(Advanced) Computer Security!

Network & Web Security

Cross-Site Scripting (XSS) Attacks & Countermeasures

(part I)

Prof. Travis Peters

Montana State University

CS 476/594 - Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

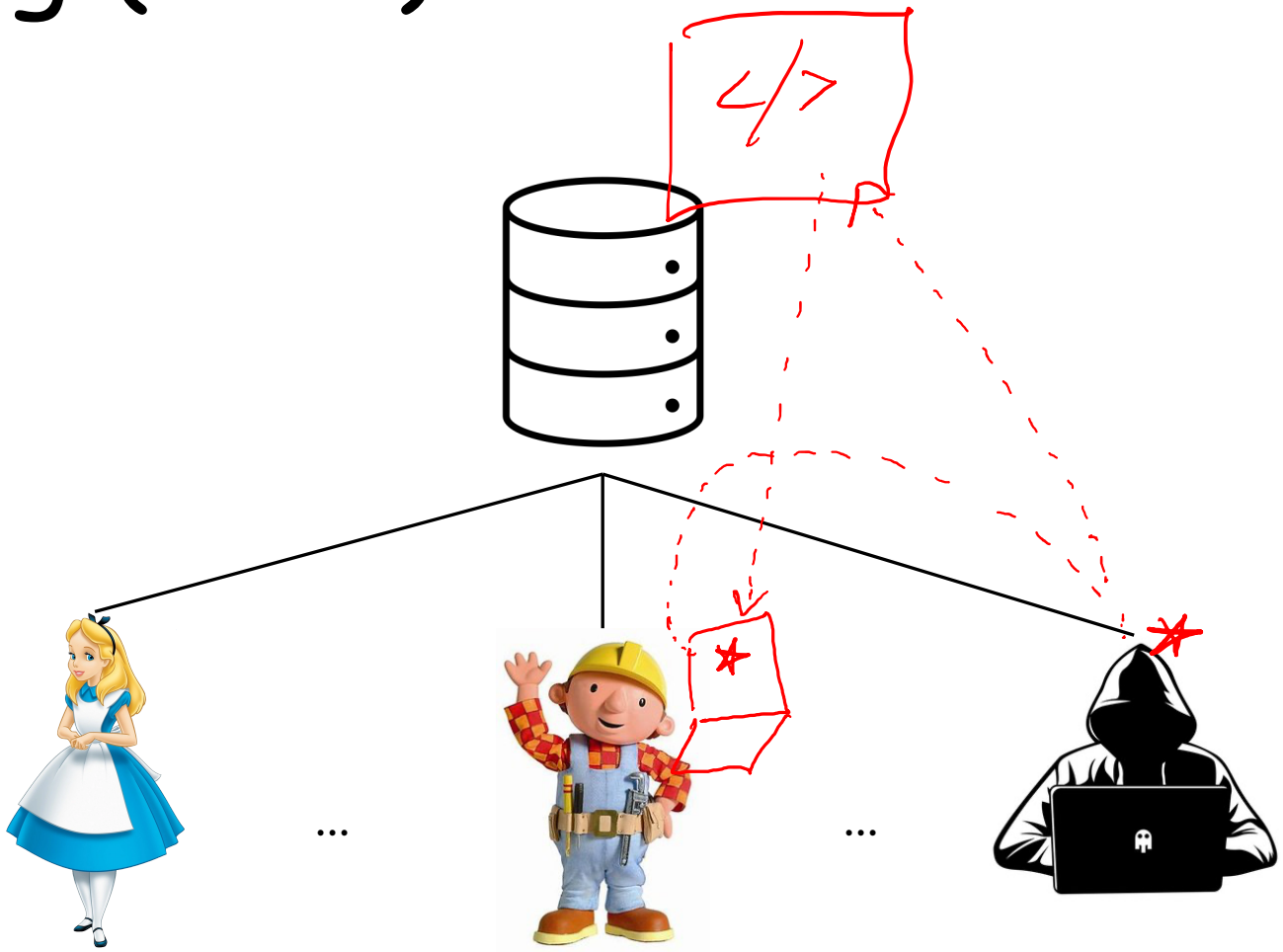
- Announcements
  - Lab 04 due today
  - Lab 05 released
- Learning Objectives
  - Pre-reqs
    - Session cookies
    - HTTP GET and POST requests
    - JavaScript and Ajax
    - The DOM & DOM APIs
  - What is XSS and how do XSS attacks work?
    - one-time XSS attacks
    - self-propagating XSS worms
  - XSS Countermeasures
    - Content Security Policy (CSP)

Cross-Site Scripting (XSS)

- 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 the HTML output…

- TL;DR XSS = Attacker code running in the user's browser!



An XSS Case Study

- 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

This is the context behind many of the lab tasks!

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.

ref: https://portswigger.net/web-security/cross-site-scripting & SEED (Chapter 11)

Types of XSS Attacks

Types of XSS Attacks

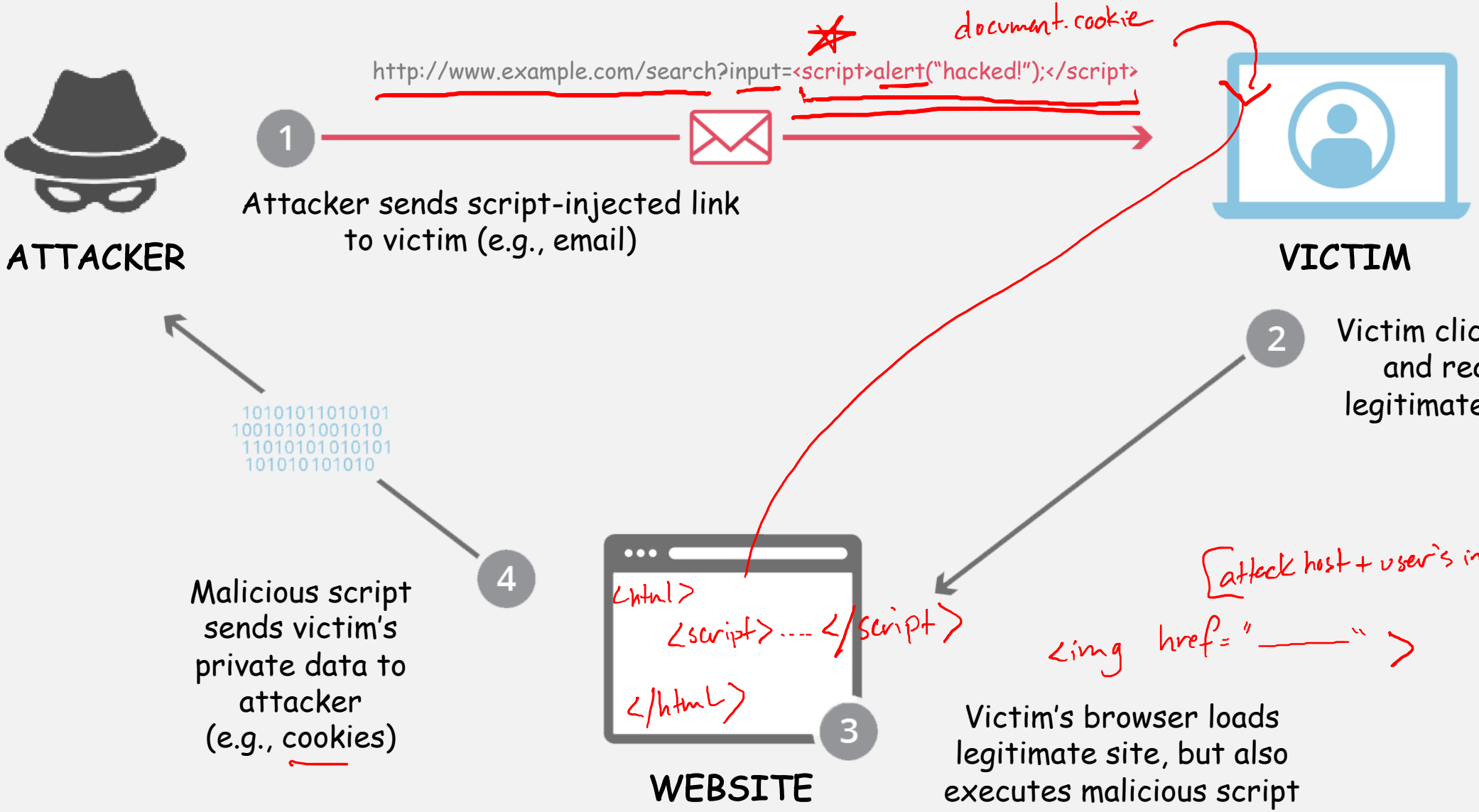
- **Reflected XSS**
  The malicious script comes from the current HTTP request

- **Stored XSS**
  The malicious script comes from the website's database

- **DOM-based XSS**
  The vuln. exists in client-side code rather than server-side code

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

Reflected XSS



Reflected XSS -> Non-Persistent

- Why does this happen?!

- Many websites are **reflective**:

  user input -> website -> (modified) user input sent back to browser

- If an application receives data from an HTTP request...
  ...and includes that data within the immediate response in an unsafe way...
  ...reflective XSS may be possible!

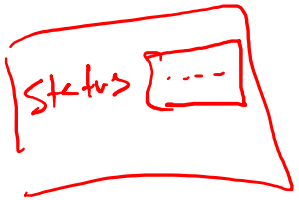
ref: https://portswigger.net/web-security/cross-site-scripting

Reflected XSS -> Non-Persistent

https://insecure-website.com/status?message=All+is+well.

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

Reflected XSS -> Non-Persistent



https://insecure-website.com/status?message=All+is+well.

↓

Status: All is well.

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

Reflected XSS -> Non-Persistent

https://insecure-website.com/status?message=<script>...Bad+stuff+here...</script>

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

Reflected XSS -> Non-Persistent

https://insecure-website.com/status?message=<script>...Bad+stuff+here...</script>

Status: <script>...Bad+stuff+here...</script>

Attacker script executes
in the user's browser!

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

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.

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

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>'>

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

Our XSS Sandbox

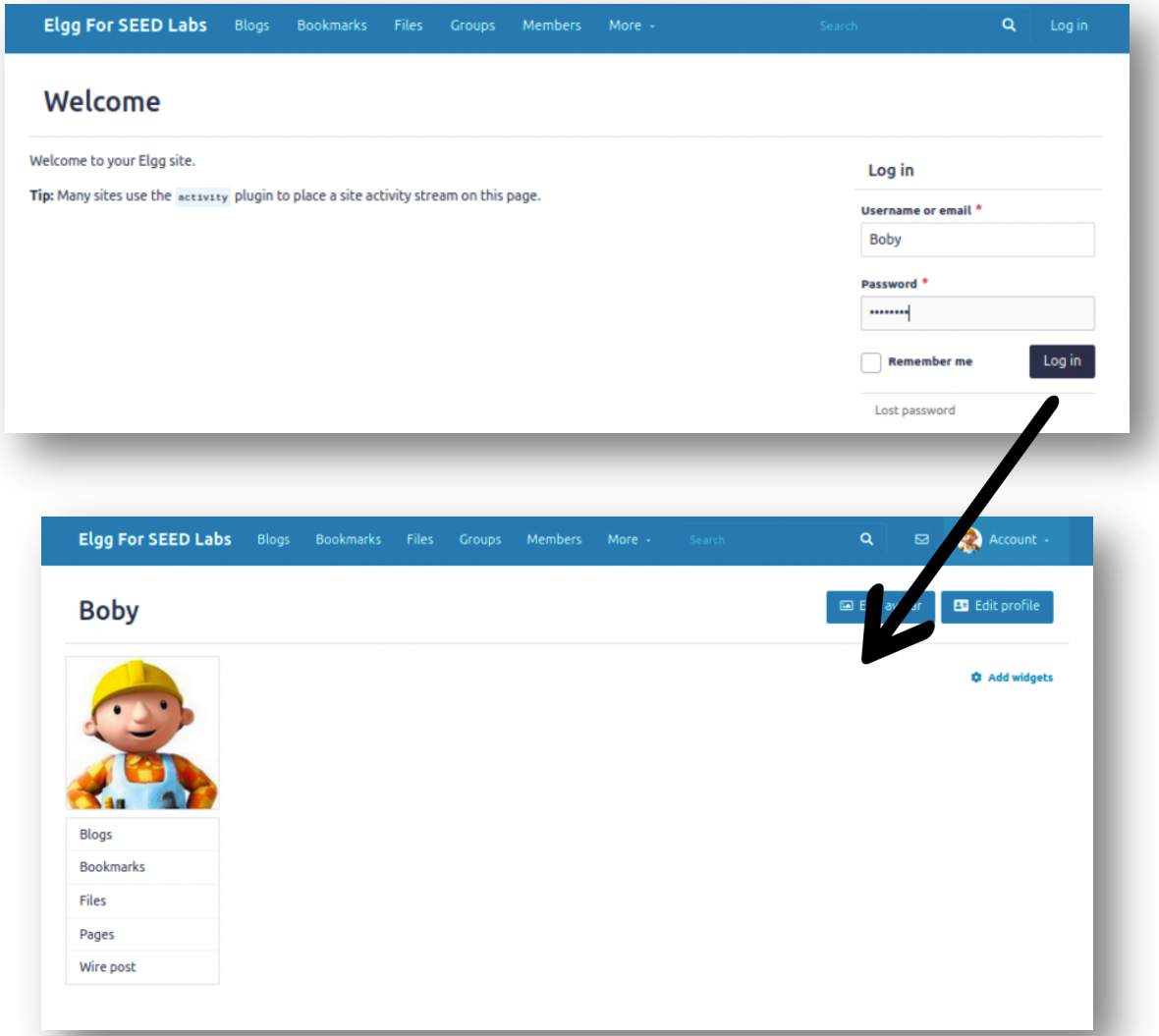
The Elgg webapp

Elgg

- An open source social networking engine
  - Adapted for SEED Labs
  - XSS countermeasures disabled

- Within the VM…
  - visit http://www.xsslabelgg.com (10.9.0.5)
  - see lab setup for info on existing user credentials (e.g., Alice, Boby, Charlie, Samy)

- Within the container…
  - website resides at /var/www/elgg
  - website config at /etc/apache2/sites-available/



You Try!

1. Log in as a user and get familiar with the website
   - Edit your profile...
   - Search for people...
   - Add a friend...

↳ make sure you dcdown from last lab

↳ you have to be in the folder w/ docker-compose.yml

Injecting JavaScript

- WHERE could we inject JavaScript code?

Name
About ME

- WHAT could we inject? (Let's start simple...)

alert(...)

You Try! (Task 1)

1. Log in as Boby
2. Add some JS code to the Boby's profile to make a pop-up appear (e.g., alert(...))
3. Verify the pop-up shows when YOU visit your own profile
4. Log out and log in as another user (e.g., Alice)
5. Visit Boby's profile

Interesting observations?

Going Further…

- Access sensitive info, such as user cookies (Task 2)
- Send user data to attacker (Task 3)

The lab basically walks you through these tasks…

Pro Tip: Use the browser's "console" to debug JavaScript and check for errors!