CSCI 476: Computer Security

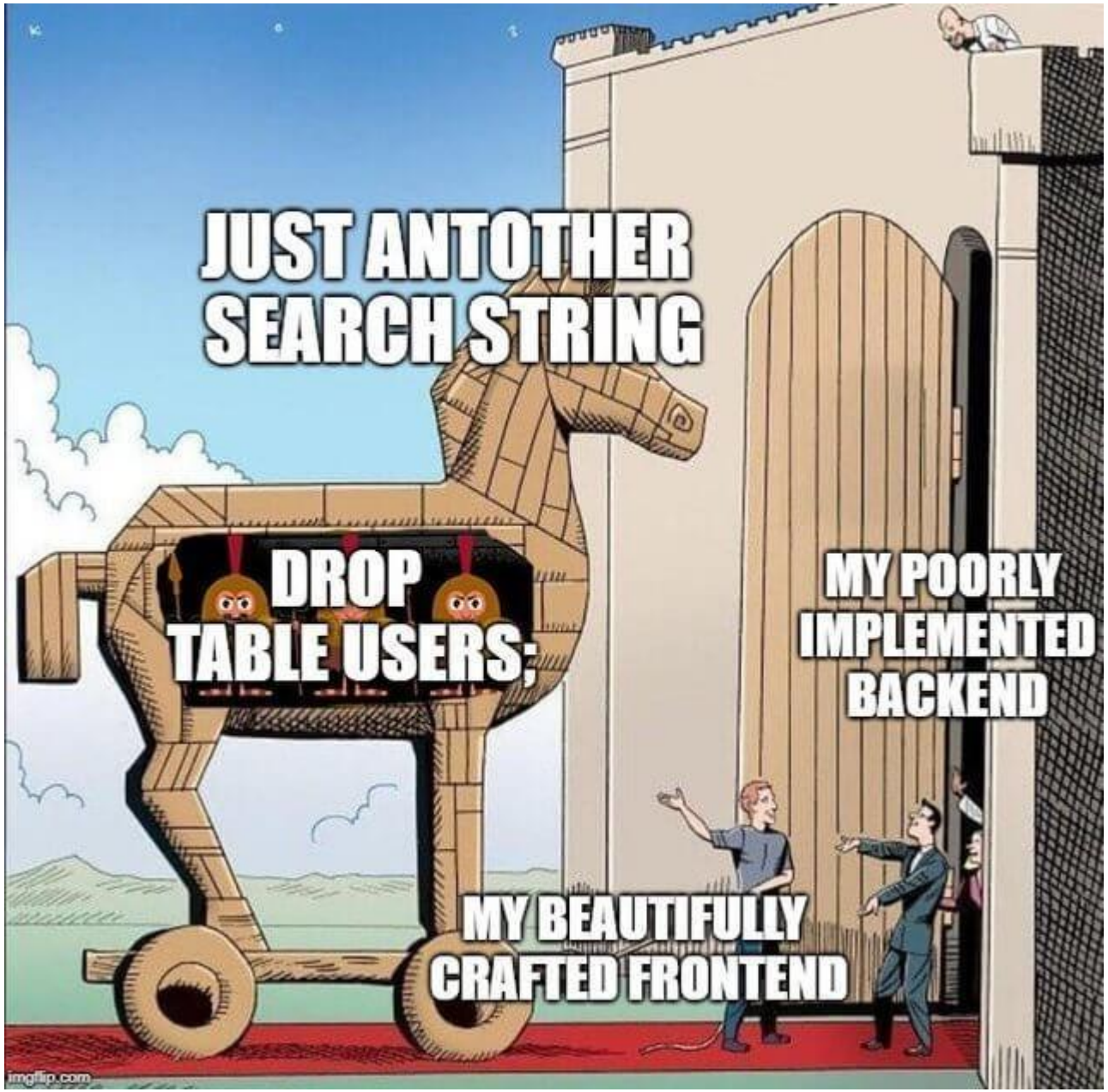
Cross Site Scripting (XSS) Attack (Part 1)

Reese Pearsall

Fall 2023

Announcement

Lab 4 (SQL injections)
Due Sunday 10/12 @
11:59 PM



Lab 4

SQL Injection

Critical SQL Injection Flaws Expose Gentoo Soko to Remote Code Execution

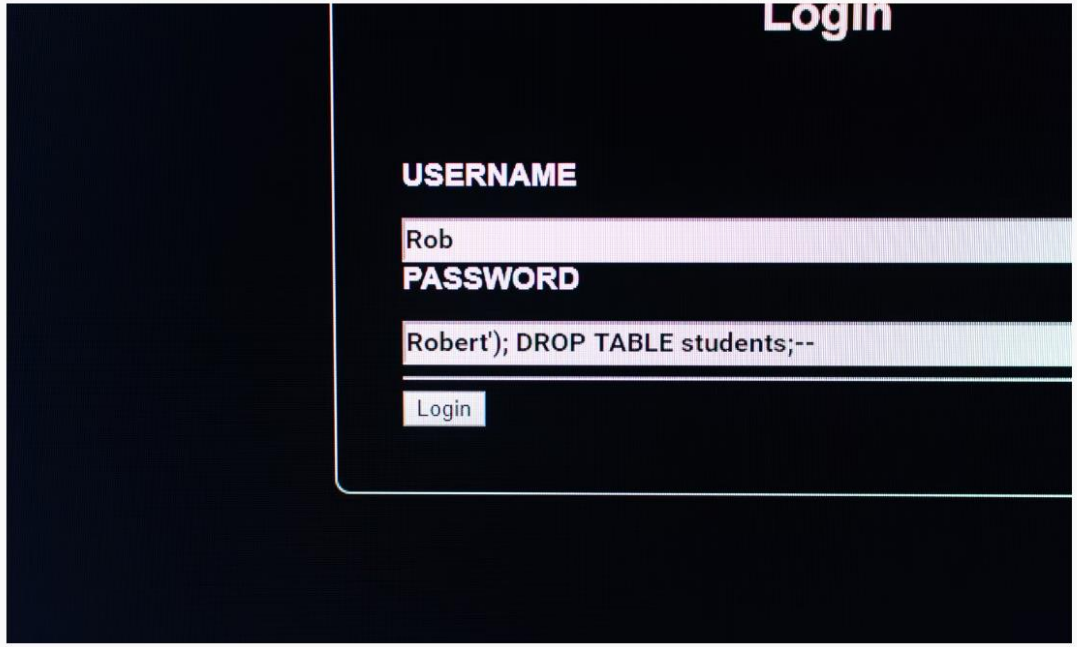
Jun 28, 2023 Ravie Lakshmanan Endpoint Security / RCE

```
-func BuildSearchQuery(searchString string) string {
-   var searchClauses []string
+func BuildSearchQuery(query *pg.Query, searchString string) *pg.Query {
    for _, searchTerm := range strings.Split(searchString, " ") {
        if searchTerm != "" {
-           searchClauses = append(searchClauses,
-               "( (category % '"+searchTerm+"') OR (name % '"+searchTerm+"') OR (atom %
'"+searchTerm+"') OR (maintainers @> '[{\"Name\": \""+searchTerm+"\"}]' OR maintainers @>
'[{\"Email\": \""+searchTerm+"\"}]'))")
+           marshal, err := json.Marshal(searchTerm)
+           if err == nil {
+               continue
+           }
+           query = query.WhereGroup(func(q *pg.Query) (*pg.Query, error) {
+               return q.WhereOr("category % ?", searchTerm).
+                   WhereOr("name % ?", searchTerm).
+                   WhereOr("atom % ?", searchTerm).
+                   WhereOr("maintainers @> ?", `[{"Name": "`+string(marshal)+`"}]`).
+                   WhereOr("maintainers @> ?", `[{"Email": "`+string(marshal)+`"}]`), nil
+           })
        }
    }
-   return strings.Join(searchClauses, " AND ")
```

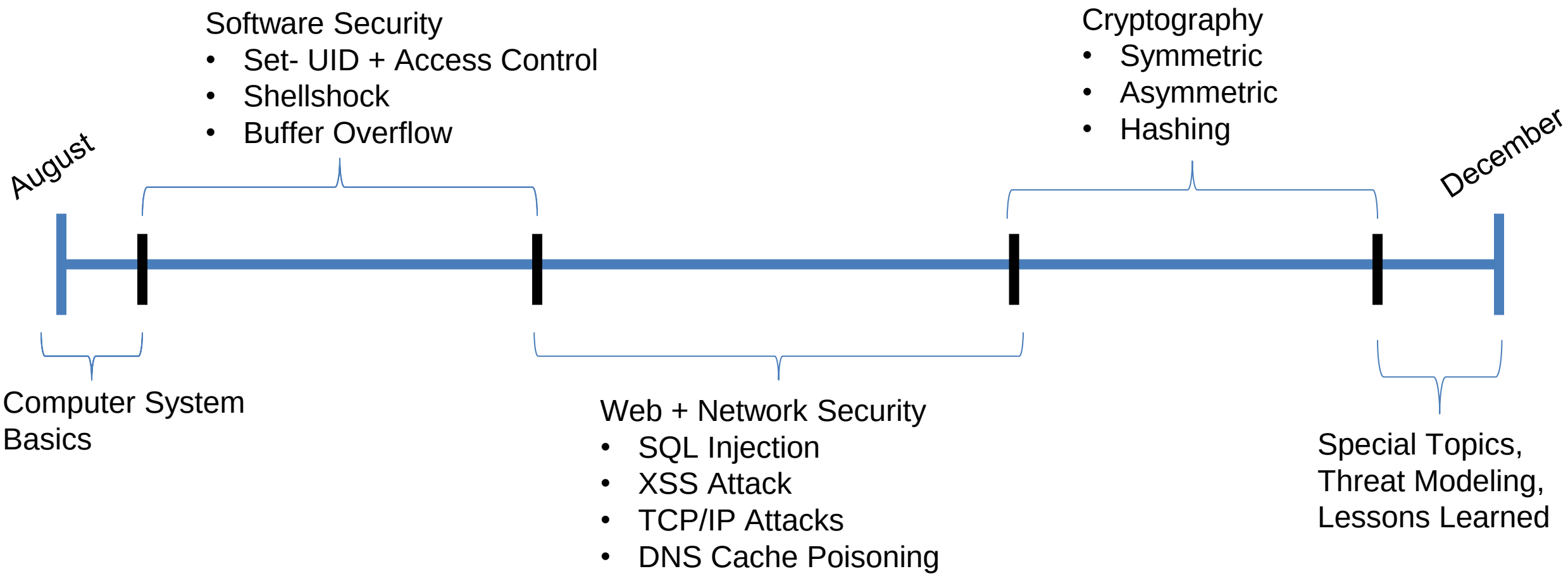
WIZ

SQL injection vulnerability in MOVEit Transfer leads to data breaches worldwide

Sponsored by Invicti June 21, 2023



Timeline and TODO



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:

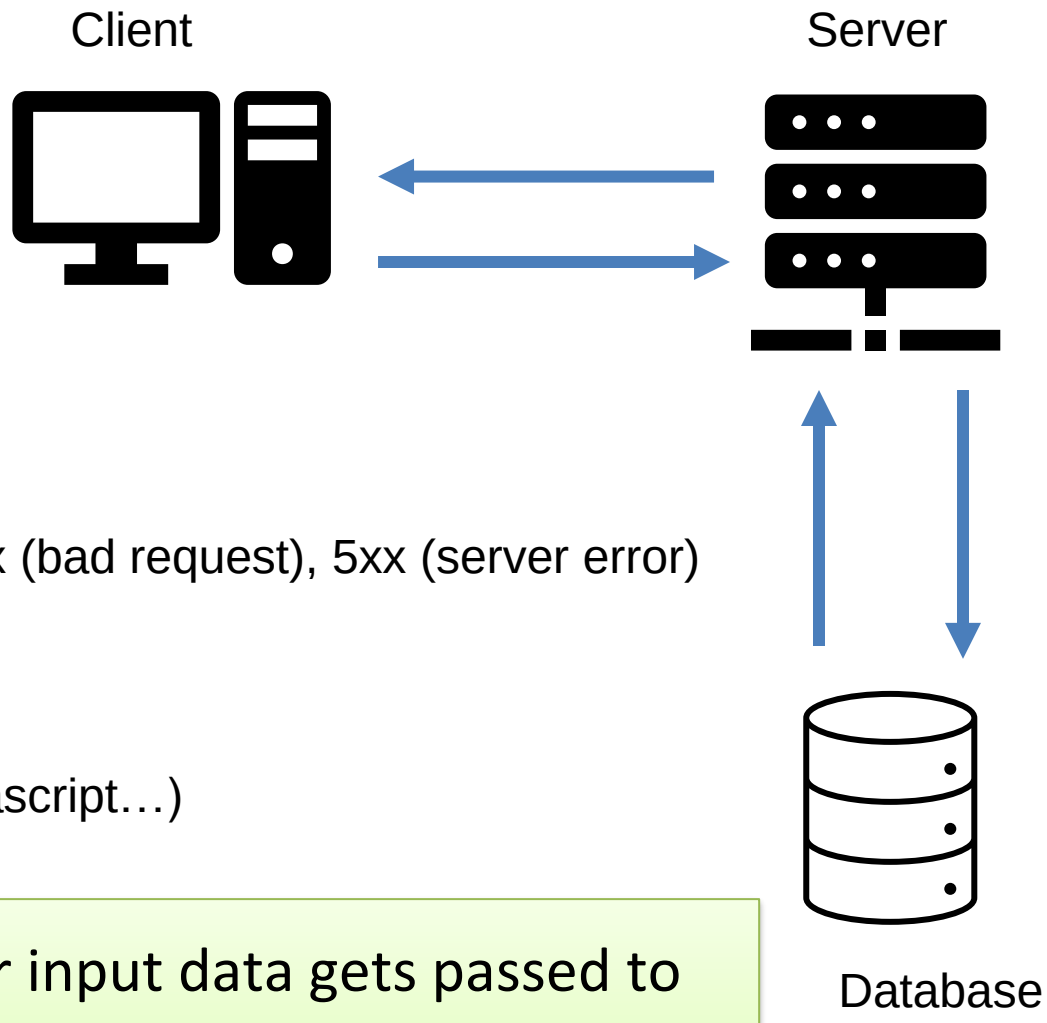
- **Format**: Method, Headers, Body
- **Methods**: GET, POST, HAD, UPDATE
- Headers: Host, referrer, User-agent, Cookie…

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**

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]

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 u span link img ul •li

```
<h3>
  Hello there~!
</h3>

<tt> This is some HTML typed up in a text editor </tt>

<h1>
  It gets rendered in your browser!
</h1
```

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

```
<h3>
  Hello there <script> getName() </script>!
</h3>

<tt> This is a list of animals pulled from an SQL database
</tt>

<script> getListOfAnimals() </script>

<h1>
  Javascript is great!
</h1>
```

Hello there reese !

This is a list of animals pulled from an SQL database

- Goat
- Dog
- Lizard

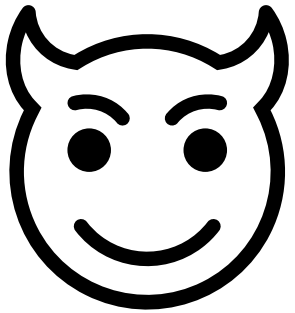
Javascript is great!

```html
<!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("AHHHHHHHHHHHHHHH");
  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**



 Hello there $value

First name: reese

PREVIEW (html)

hello there reese

 Hello there $value

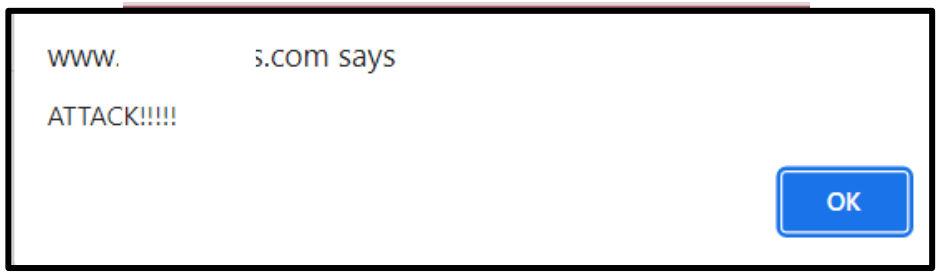
http://unsafe-website.com?**value=reese**

PREVIEW (html)

hello there reese

 Hello there $value

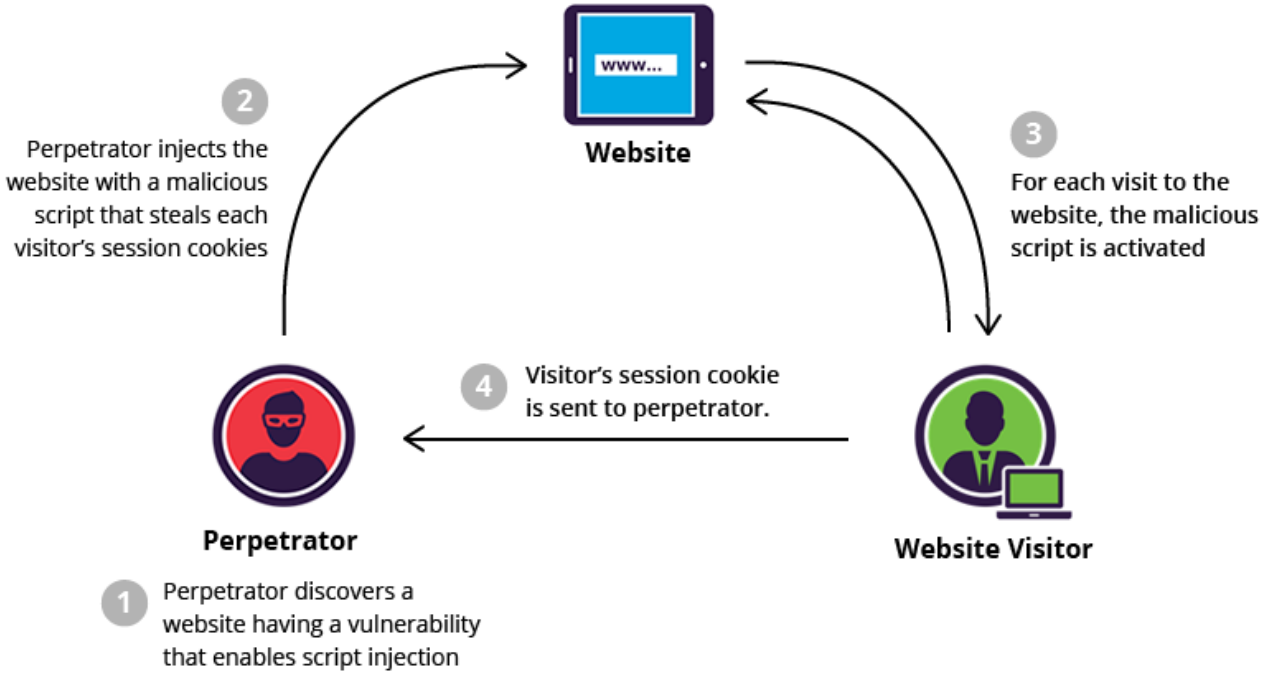
First name: reese <script> alert("ATTACK!!"); </script>



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

Cross Site Scripting (XSS) is a type of web vulnerability that allows an attack to inject their own malicious client-side scripts into benign webpages that will be loaded by other users



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

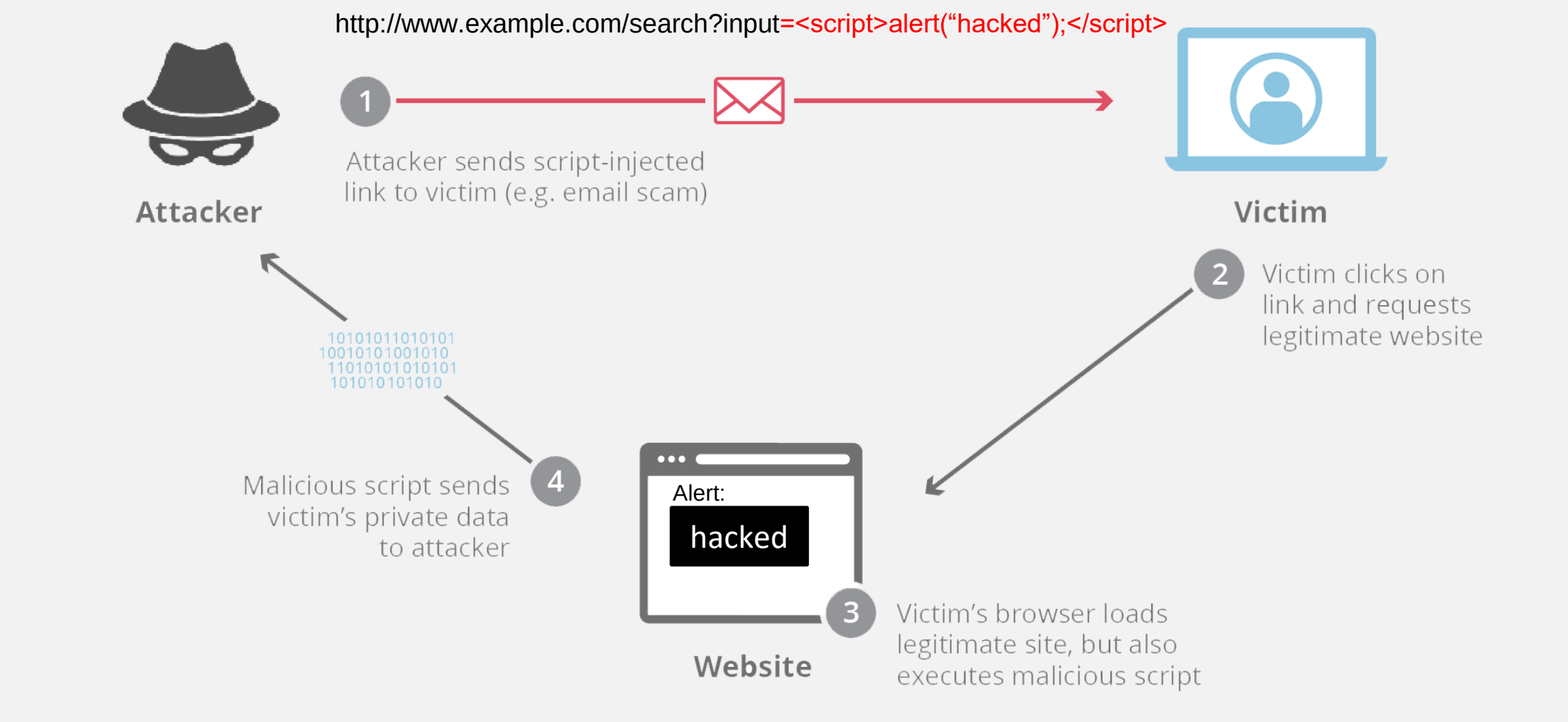
- **Reflected XSS**
  The malicious script comes from the current HTTP request

- **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

Types of XSS

Reflective



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!

Reflected XSS (Non-Persistent)

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

Reflected XSS (Non-Persistent)

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

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

↓

Status: All is well.

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!

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.

SSN

{USER-MESSAGE-HERE}

↓

Hello, this is my message!

↓

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

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
```

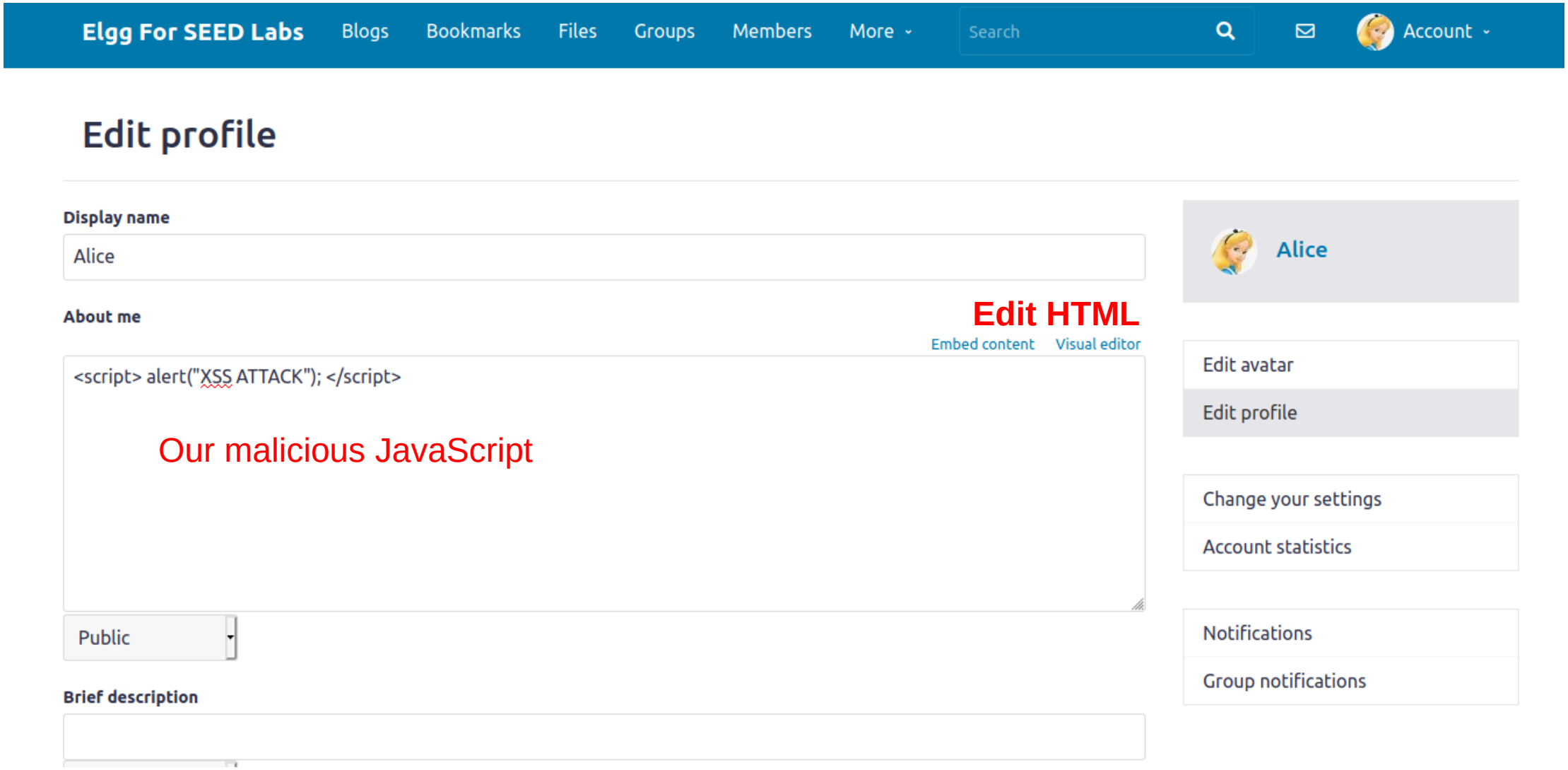
Elgg is an open source web framework for creating social media sites

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

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

(do not visit this site elsewhere)

Basic XSS Attack to display a message



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 (!!!)