

# Secure User Interfaces

## CS571: Building User Interfaces

**Cole Nelson**

# Today's Warmup

1. Grab some cookies 
2. Download & install [Mozilla Firefox](#)

# Announcements

Today's content *will **not** be* on the final exam.

Please complete [the end-of-semester survey](#) if you haven't done so already!

Work **must** be turned in by *Friday, May 3rd @ 11:59 pm*

# Final Exam

50 questions, all multiple choice. Half on design, half on implementation. Cumulative, but with a focus on more recent content. 90 minutes to complete.

- Alternative on Thur. 5/9 @ 5:05 pm in Chem S429
- General on Fri. 5/10 @ 5:05 pm
  - SEC001 in Chamberlin 141
  - SEC002 in Chem S429

A confirmation email will be sent by EOD tomorrow!

# Learning Objectives

1. Be able to adhere to ethical considerations when doing application security.
2. Be able to attack with and defend against SQLi, XSS, and XSRF vulnerabilities.
3. Be able to understand the value in dependency management and server-side validation.

# A Disclaimer

*a badger wearing a red shirt with a w on it in jail for committing cybercrimes, pixel art*

generated using DALL-E



[VIEW SOURCE](#) —

# Viewing website HTML code is not illegal or “hacking,” prof. tells Missouri gov.

Professor demands that governor halt "baseless investigation" and apologize.

[JON BRODKIN](#) - 10/25/2021, 3:09 PM

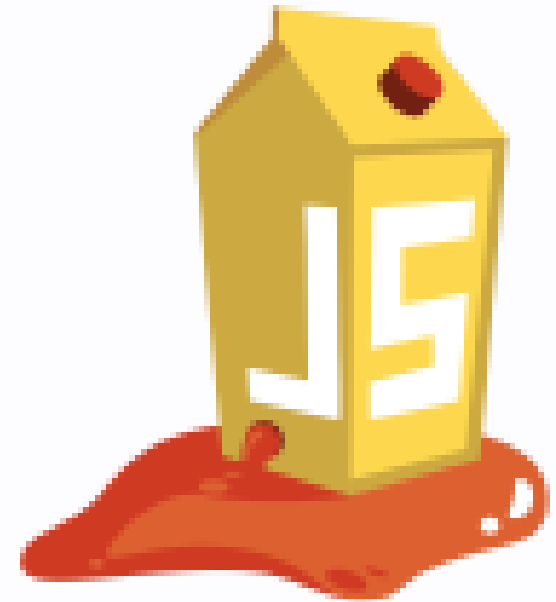
# Building Secure User Interfaces

Make sure you have permission to resources before performing security audits!

Start with a **sandbox** then work with **safe harbors**.

- [HackerOne](#)
- [OpenBugBounty](#)
- [BugCrowd](#)





Not an endorsement of any particular service.



Not an endorsement of any particular service.

# What is a vulnerability?

A vulnerability is a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application.

— OWASP

# Common Vulnerabilities

Vulnerabilities affect both frontends *and* backends!

Today we will look at...

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (XSRF)
- Vulnerable and Outdated Components
- Software and Data Integrity Failures

# SQL Injection (SQLi)

```
' OR 1=1; DROP TABLE Grades; --
```

# SQL Injection (SQLi)

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data...

## OWASP Definition

# SQLi

Imagine we have the following SQL query...

```
`INSERT INTO Messages(title, content) VALUES('${title}', '${content}')`
```

When the **title** is "hello" and **content** is world...

```
INSERT INTO Messages(title, content) VALUES('hello', 'world')
```

All is good!

# SQLi

Imagine we have the following SQL query...

```
`INSERT INTO Messages(title, content) VALUES('${title}', '${content})`
```

When the `title` is `','`); DROP TABLE Messages; --  
and the `content` is anything else...

```
INSERT INTO BadgerMessage(title, content) VALUES('',''); DROP TABLE Messages; --', 'anything')
```

All is **NOT** good!



# SQLi Mitigations

Sanitize your inputs **on the backend!**

- Frontend sanitization is *never* sufficient!

Use parameterized queries instead!

[Image Source](#)



# Cross-Site Scripting (XSS)

```

```

# Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

## OWASP Definition

# HW2 XSS

Render each student using `innerHTML`

```
let html = "<div>";  
html += `<h2>${student.name}</h2>`;   
html += "</div>";  
return html;
```

# HW2 XSS

When input is Michael ...

```
<div>  
  <h2>Michael</h2>  
</div>
```

<i>Michael</i> ...

```
<div>  
  <h2><i>Michael</i></h2>  
</div>
```

# HW2 XSS

```
<script>alert("oops!")\</script> ...
```

```
<div>  
  <h2><script>alert("oops!")</script></h2>  
</div>
```

```
 ...
```

```
<div>  
  <h2></h2>  
</div>
```

# DOM-based vs Persistent XSS

## DOM-based XSS

```
https://example.com/search?q=%3Cimg%20src=%22%22%20onerror=%22alert(1)%22/%3E
```

## Persistent XSS

### Bascom Chatroom

Post Title

Post Content

Create Post

# XSS Demo w/ BadgerChat

BadgerChat is *NOT* a safe harbor -- please ask for permission before pentesting.



# XSS Mitigations

Sanitize your displays!

Do not create a sanitizer yourself!

React **performs sanitization** for you.

[Image Source](#)



# Cross-Site Request Forgery (XSRF)

`nefarious.example.com` making a request on behalf of  
a user to `usbank.com`

# XSRF

You can `fetch` anything! You can `credentials:`  
`'include'` anything! It just depends on whether or not  
the server will accept your request.

# XSRF Demo w/ BadgerChat

BadgerChat is *NOT* a safe harbor -- please ask for permission before pentesting.

# XSRF Mitigations

Use of a randomly-generated nonce that is included with every request (called an anti-xsrftoken).

Use of stricter Cross-Origin Resource Sharing (CORS) policies.

Use of SameSite cookies set as `Lax` or `Strict`.

Use of the `Origin` request header; this cannot be manipulated by arbitrary JavaScript!

# XSRF Mitigations

Use a stricter CORS policy...

```
app.use((req, res, next) => {  
  res.header("Access-Control-Allow-Origin", req.headers.origin);  
  res.header("Access-Control-Allow-Headers", req.headers["access-control-request-headers"]);  
  res.header('Access-Control-Allow-Methods', req.headers["access-control-request-method"]);  
  res.header('Access-Control-Allow-Credentials', 'true');  
  res.header('Access-Control-Expose-Headers', 'Set-Cookie');  
  res.header('Vary', 'Origin, Access-Control-Allow-Headers, Access-Control-Allow-Methods')  
  next();  
});
```

...this is about as loose as a policy gets!

# Use of Outdated and Vulnerable Components

Keep them up to date (if you can!)

# Software and Data Integrity Failures

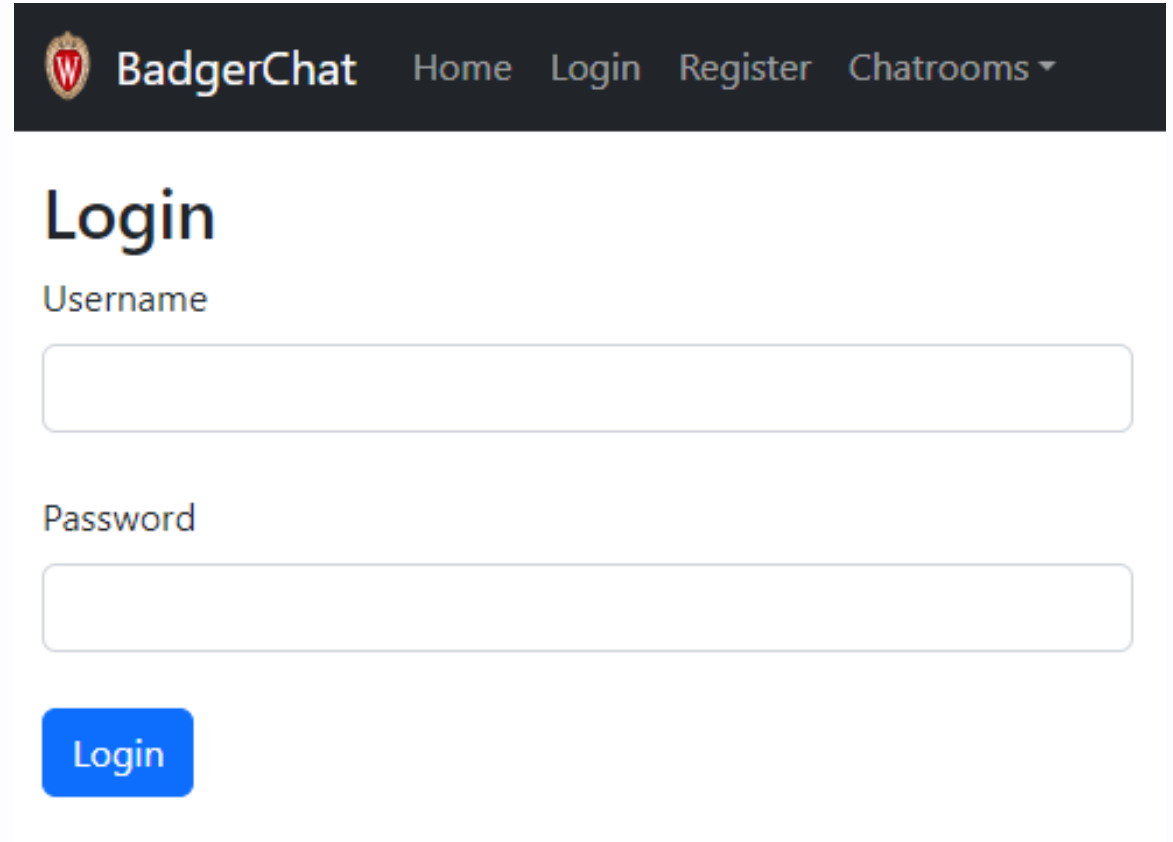


# Validation

Frontends are just a way of getting to the backend!

**Do not rely solely on frontend validation.**

The user can send more than you allow them to.

A screenshot of the BadgerChat login page. At the top is a dark navigation bar with the BadgerChat logo (a red shield with a white 'W') and the text 'BadgerChat'. To the right of the logo are links for 'Home', 'Login', 'Register', and 'Chatrooms' with a dropdown arrow. Below the navigation bar, the word 'Login' is displayed in a large, bold, dark font. Underneath 'Login' are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Both fields are empty and have a light gray border. Below the password field is a blue button with the word 'Login' in white text.

BadgerChat Home Login Register Chatrooms ▾

## Login

Username

Password

Login



Image Source

# Software and Data Integrity Failure Demo

On [OWASP JuiceShop](#)

# The Bonus CTF Flag

## JSON

msg: "Purchased!"

```
additional_msg: "Congrats! You have broken your first API! 🎉"
```

[illegible]

You'll paste this in the last quiz question.

**Difficulty:** ★★

# The Bonus++ CTF Flag

```
{  
  "msg": "Congrats! You completed the bonus-bonus CTF! The PIN code is below. 💎",  
  "code": "###"  
}
```

Unlocks the treasure chest! 🔑

**Difficulty:** ★★ ★ — ★★ ★★

# Your Turn!

Get hacking! :)

# What can we do in defense?

# Mitigation Strategies

Technical strategies may include...

- Obfuscation
  - e.g. don't produce [a sourcemap](#)
- Web Application Firewall (WAF)
- Containerization (Using [Docker](#) or [VMs](#))
- Defense in Depth (Swiss Cheese Approach)



# Mitigation Strategies

Non-technical strategies may include...

- Threat Modeling
- Least Privilege
- Scary Messages
- Ask Nicely :)

# </CS571>

What's next?

**CS570** Introduction to Human-Computer Interaction  
→ explore UX methods (research, design, evaluation)

**CS770** Human-Computer Interaction  
→ core topics and research methods in HCI research

See also [hci.cs.wisc.edu](http://hci.cs.wisc.edu).

# Thank you! :)