Building Secure UIs

CS571: Building User Interfaces

Cole Nelson & Yuhang Zhao

Announcements

- Please complete your course evaluations! :)
- All HWs **must** be turned in by tomorrow evening.
- Office hours end Friday evening, by appointment only following.
- Today's content will *only* be on the Bonus Quiz.
  - "Due" on 12/13, can be completed until 12/18.

Final Exam

Final Exam is on Monday, December 18th at 10:05 am.

- SEC001 in Bascom 272
- SEC002 in Humanities 3650

You will have 90 minutes for 40 MC (20 on design and 20 on implementation)

- Alternative final exam confirmation and information has been sent out. Fill out form by **EOD TODAY**.

Don't forget your double-sided notesheet!

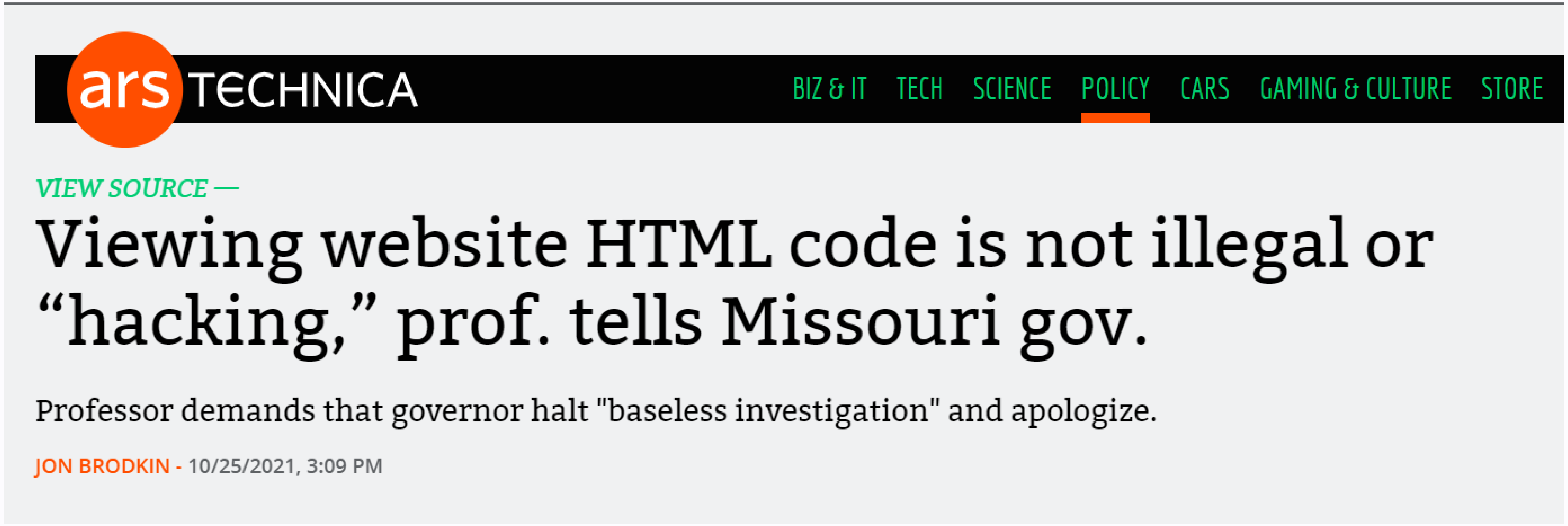
What will we learn today?

- How to (not) end up in jail?
- What are bug bounty programs?
- What are the OWASP Top 10?
- How do SQLi attacks work?
- How do XSS attacks work?
- How do vulnerable dependencies create risk?
- Why is client-side validation insufficient?
- What are common mitigation strategies?

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!

- HackerOne
- OpenBugBounty
- BugCrowd

Look for **safe harbors** or **local resources**.

OWASP

TOP 10

OWASP Top 10 (2021)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration

OWASP Top 10 (2021)

6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

Common Vulnerabilities

Vulnerabilities affect both frontends *and* backends!

Today we will look at...

- SQL Injection
- Cross-Site Scripting (XSS)
- Vulnerable and Outdated Components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 *not* sufficient!

Use parameterized queries instead!

Image Source

SQLi Demo

CS220 | Download here! (requires anaconda)

Cross-Site Scripting (XSS)

```
<img src="0" onerror="alert(document.cookie)"/>
```

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>
```

`<img src="0" onerror="alert(document.cookie)"/>` …

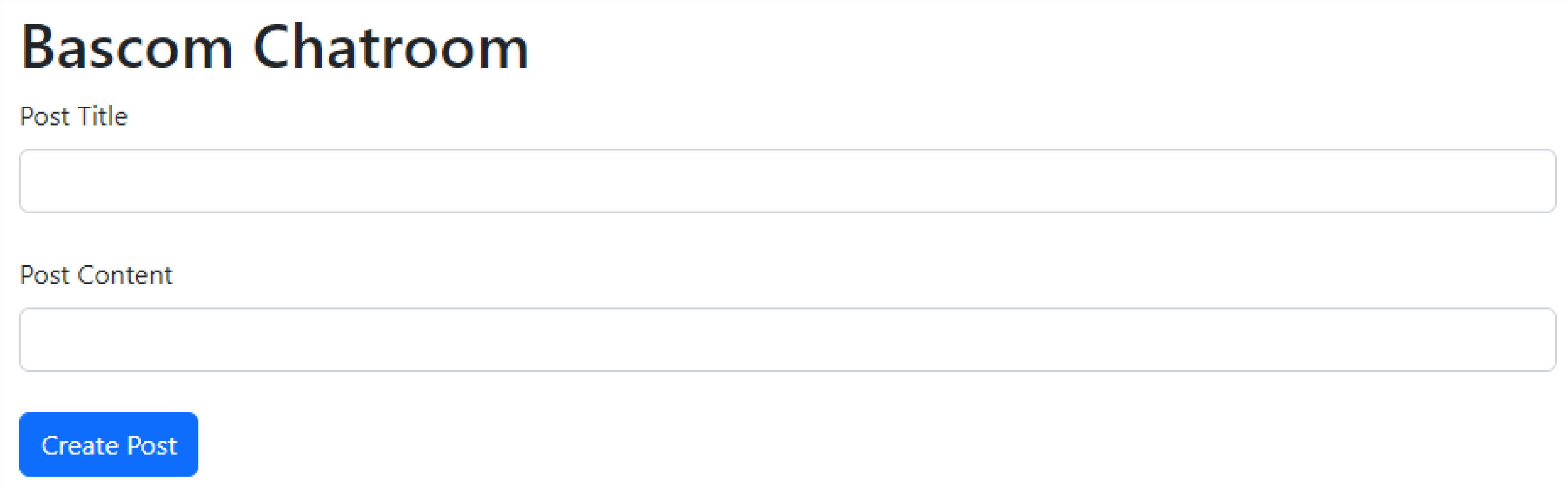
```
<div>
  <h2><img src="0" onerror="alert(document.cookie)"/></h2>
</div>
```

DOM-based vs Persistent XSS

DOM-based XSS

```
https://example.com/search?q=%3Cimg%20src=%220%22%20onerror=%22alert(1)%22/%3E
```

Persistent XSS



XSS Mitigations

Sanitize your displays!

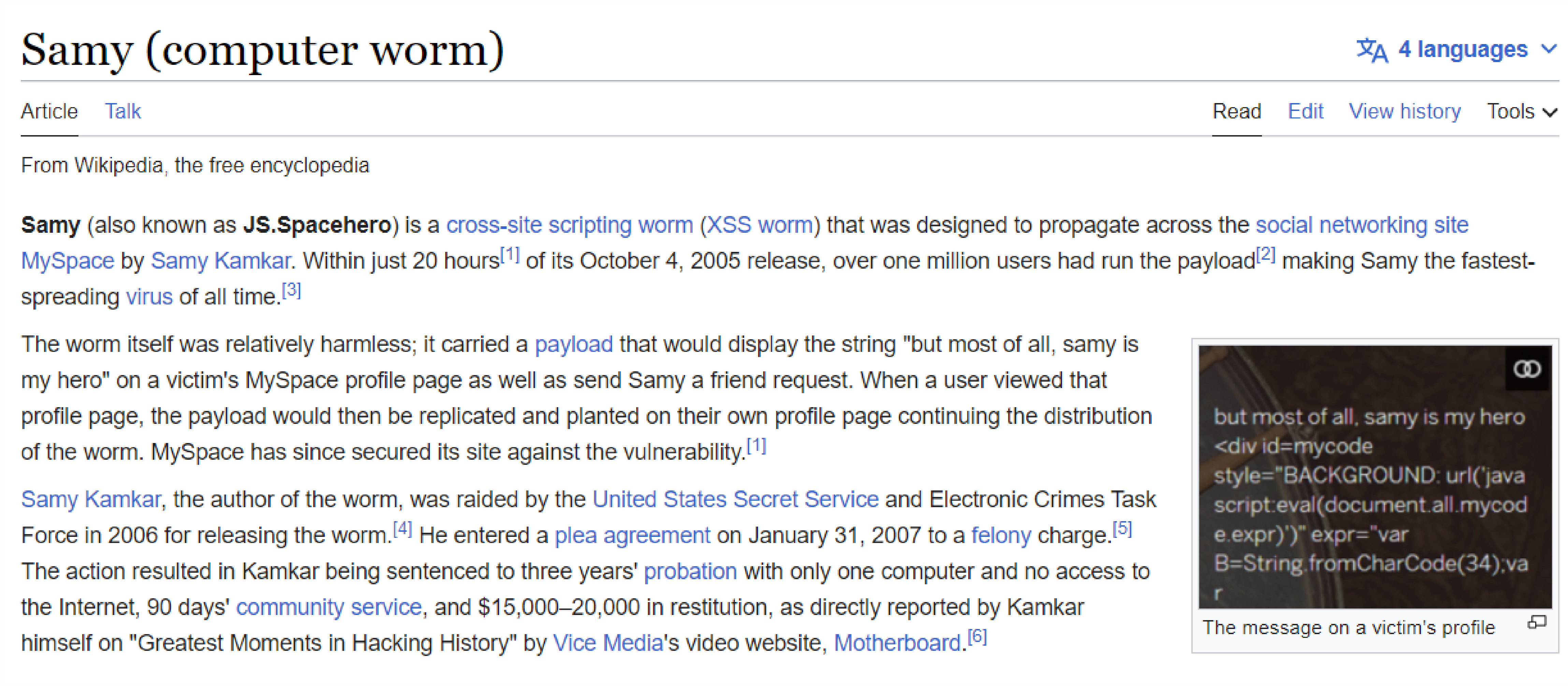
Do not create a sanitizer yourself!

React performs sanitization for you.

Image Source

XSS Demo w/ BadgerChat

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



Samy (computer worm)

4 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Samy (also known as **JS.Spacehero**) is a cross-site scripting worm (XSS worm) that was designed to propagate across the social networking site MySpace by Samy Kamkar. Within just 20 hours[1] of its October 4, 2005 release, over one million users had run the payload[2] making Samy the fastest-spreading virus of all time.[3]

The worm itself was relatively harmless; it carried a payload that would display the string "but most of all, samy is my hero" on a victim's MySpace profile page as well as send Samy a friend request. When a user viewed that profile page, the payload would then be replicated and planted on their own profile page continuing the distribution of the worm. MySpace has since secured its site against the vulnerability.[1]

Samy Kamkar, the author of the worm, was raided by the United States Secret Service and Electronic Crimes Task Force in 2006 for releasing the worm.[4] He entered a plea agreement on January 31, 2007 to a felony charge.[5] The action resulted in Kamkar being sentenced to three years' probation with only one computer and no access to the Internet, 90 days' community service, and $15,000–20,000 in restitution, as directly reported by Kamkar himself on "Greatest Moments in Hacking History" by Vice Media's video website, Motherboard.[6]

```
but most of all, samy is my hero
<div id=mycode
style="BACKGROUND: url('java
script:eval(document.all.mycod
e.expr)')" expr="var
B=String.fromCharCode(34);va
r
```

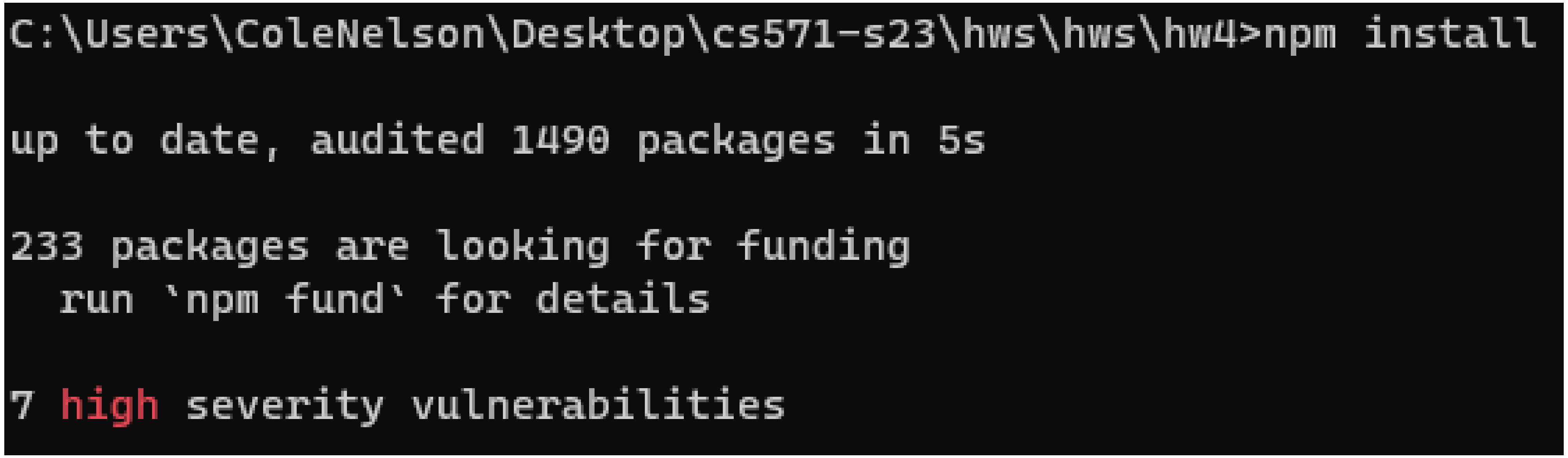
The message on a victim's profile

SQLi & XSS Demos

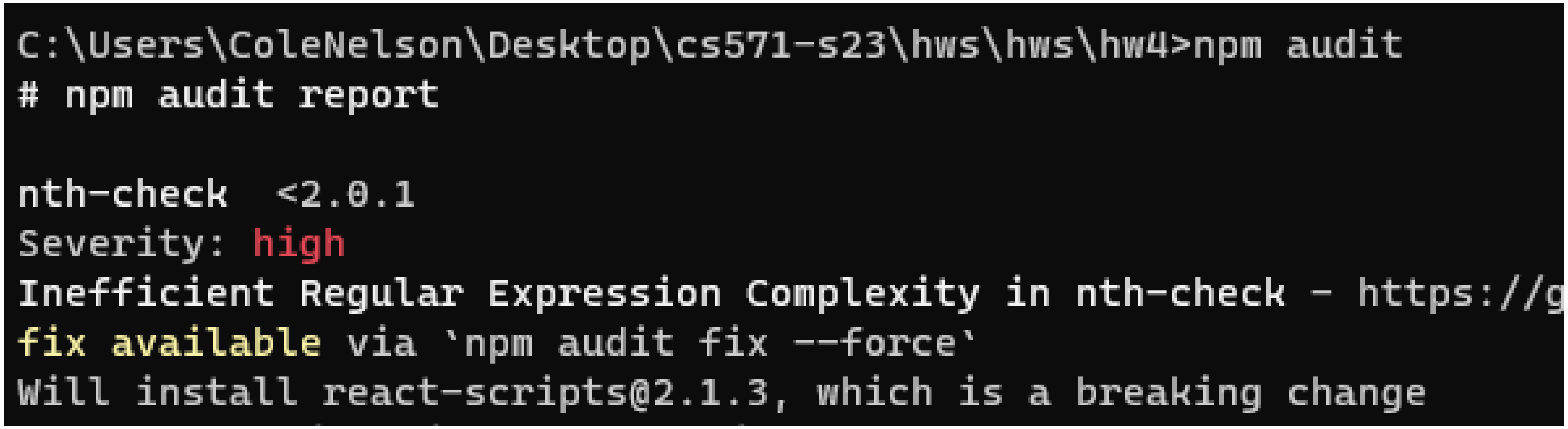
Using OWASP JuiceShop | Download here!

Use of Outdated and Vulnerable Components

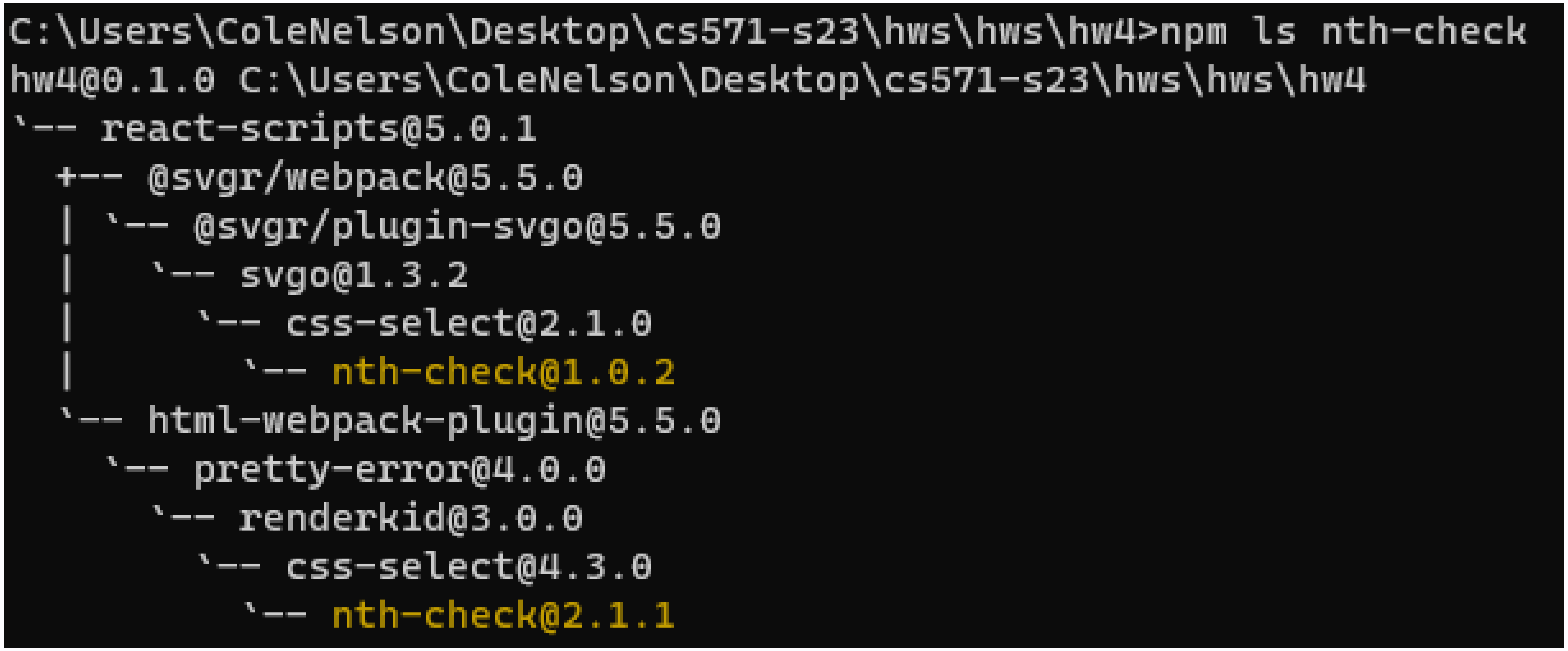
`angular@1.8.3` among many, many others...



```
C:\Users\ColeNelson\Desktop\cs571-s23\hws\hws\hw4>npm install

up to date, audited 1490 packages in 5s

233 packages are looking for funding
  run `npm fund` for details

7 high severity vulnerabilities
```



```
C:\Users\ColeNelson\Desktop\cs571-s23\hws\hws\hw4>npm audit
# npm audit report

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check - https://g
fix available via `npm audit fix --force`
Will install react-scripts@2.1.3, which is a breaking change
```



```
C:\Users\ColeNelson\Desktop\cs571-s23\hws\hws\hw4>npm ls nth-check
hw4@0.1.0 C:\Users\ColeNelson\Desktop\cs571-s23\hws\hws\hw4
`-- react-scripts@5.0.1
  +-- @svgr/webpack@5.5.0
  | `-- @svgr/plugin-svgo@5.5.0
  |   `-- svgo@1.3.2
  |     `-- css-select@2.1.0
  |       `-- nth-check@1.0.2
  `-- html-webpack-plugin@5.5.0
    `-- pretty-error@4.0.0
      `-- renderkid@3.0.0
        `-- css-select@4.3.0
          `-- nth-check@2.1.1
```

Outdated and Vulnerable Components

A vulnerable dependency does not *necessarily* mean the application is vulnerable.

Likewise, an application without vulnerable dependencies *could still* be vulnerable.

Vulnerable dependencies are *flags* to look into.

Checking Vulnerabilities

Check the CVE (Common Vulnerabilities and Exposures)

e.g. CVE-2021-3803...

- NIST
- GitHub
- Snyk

Does it have to be in the production build? Can it be specified as a dev dependency?

LOG4J

ANGULARJS

by Google

Outdated Dependency Mitigations

Keep your dependencies up-to-date!

- e.g. create-react-app is no longer maintained!

- Static Analysis (SAST) Tools
  - OWASP DependencyCheck
- Continuous Maintenance
- Minimize Surface Area

Beware of changing technologies!

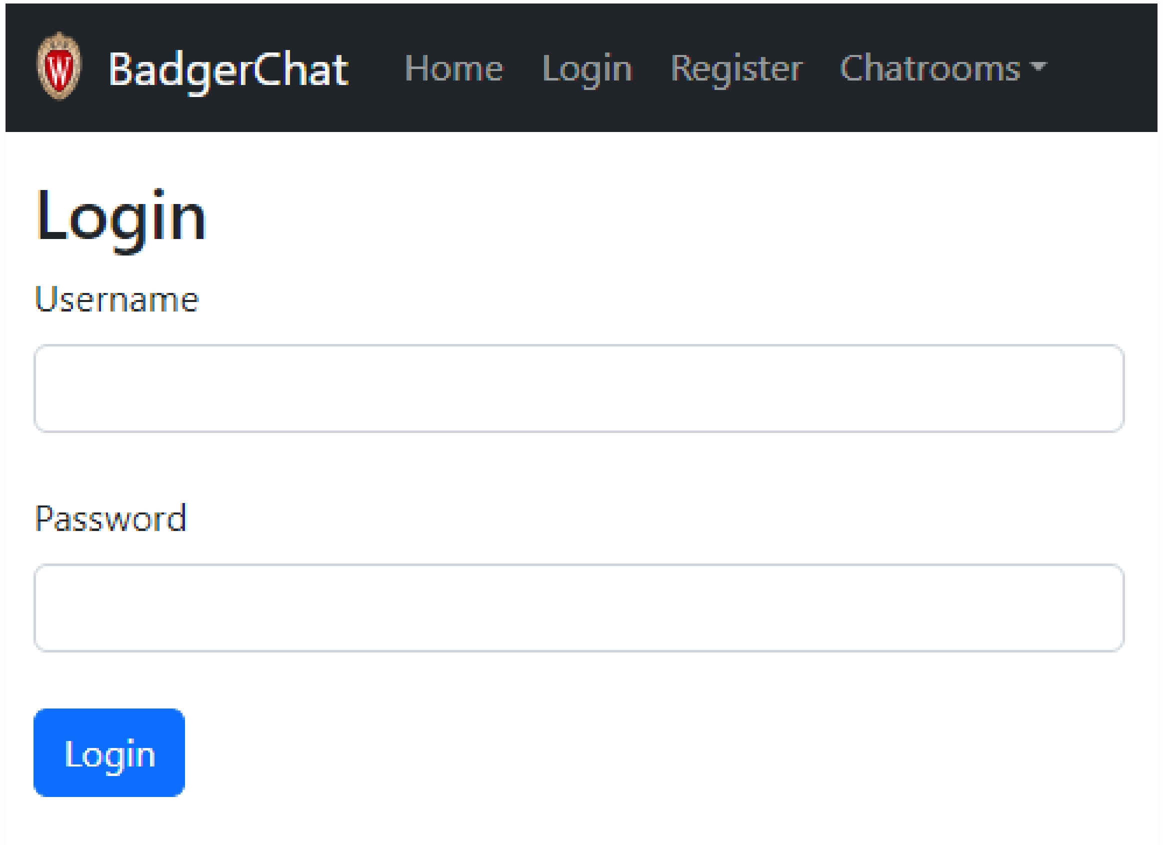
Software and Data Integrity Failures

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.



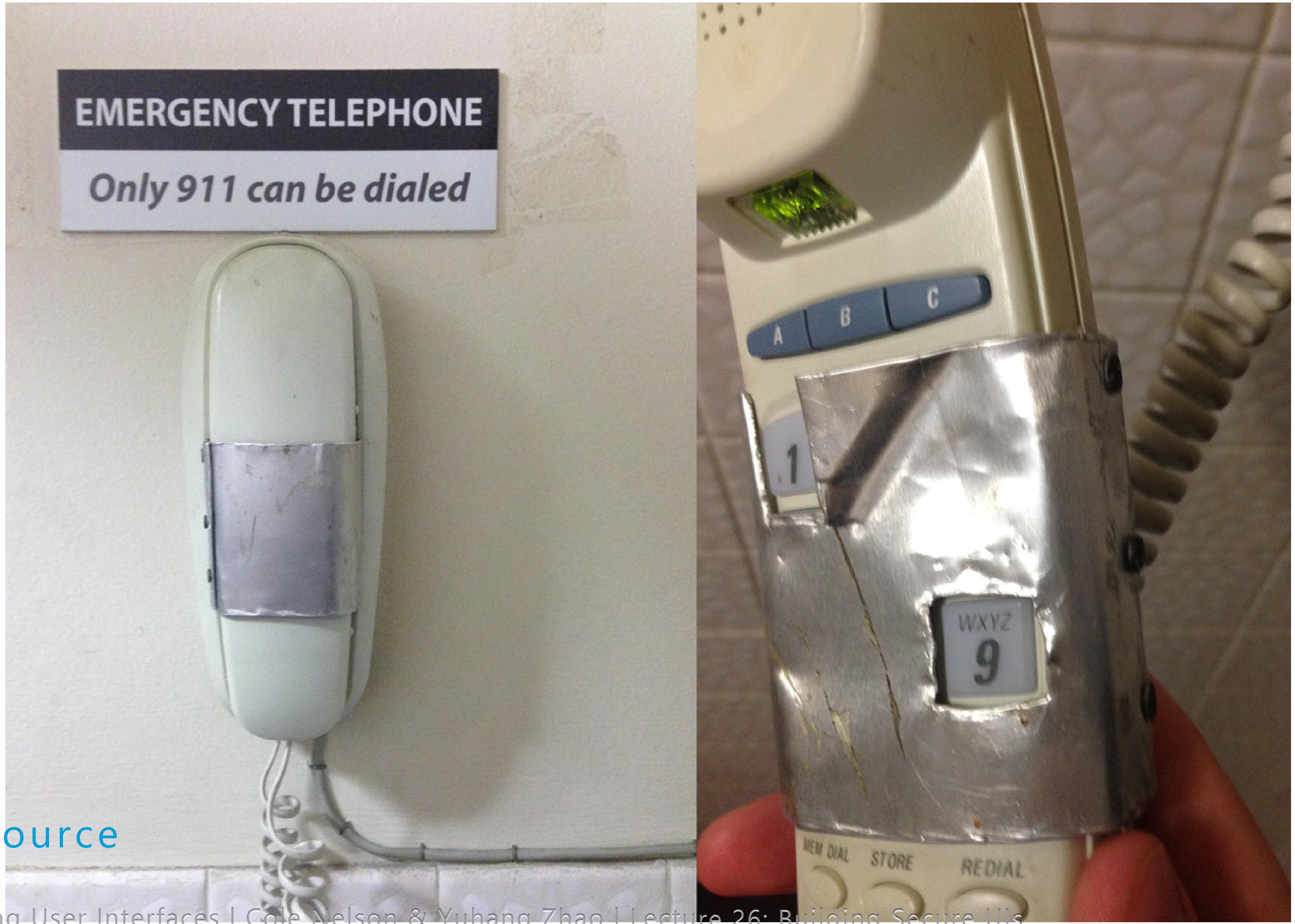


Image Source

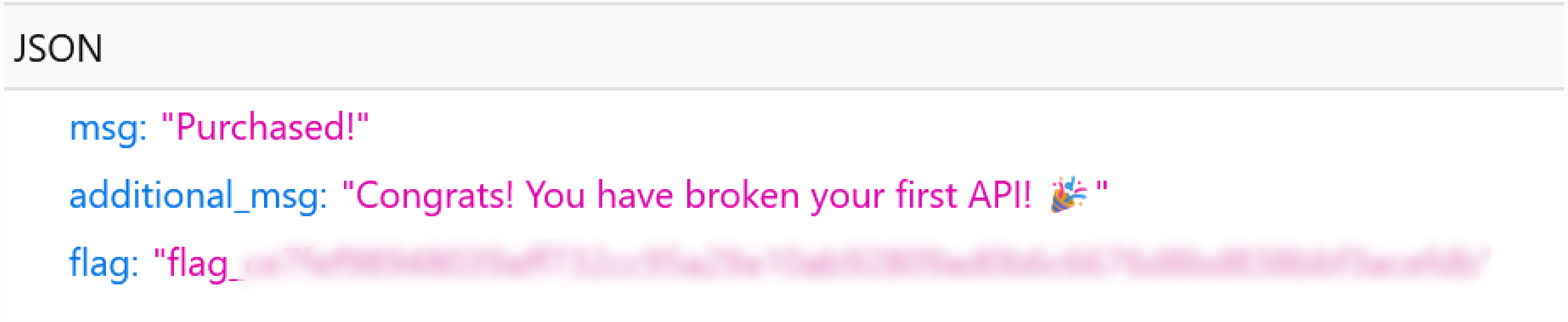
Software and Data Integrity Failure Demo

On OWASP JuiceShop

Bonus Quiz

- Two multiple-choice questions on today's content.
- A capture-the-flag on Badger Backers
  - This is the **ONLY** API we allow you to "hack"
  - **Hint:** Its a "Software and Data Integrity Failure"

The Flag


JSON

```
msg: "Purchased!"
additional_msg: "Congrats! You have broken your first API! 🎉"
flag: "flag_[redacted]
```

You'll paste this in the last quiz question.

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.

Thank you! :)