ECS 162
WEB PROGRAMMING

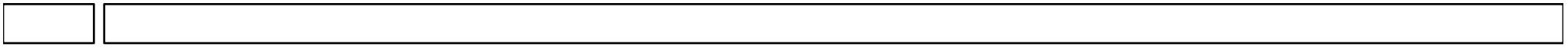
Good questions

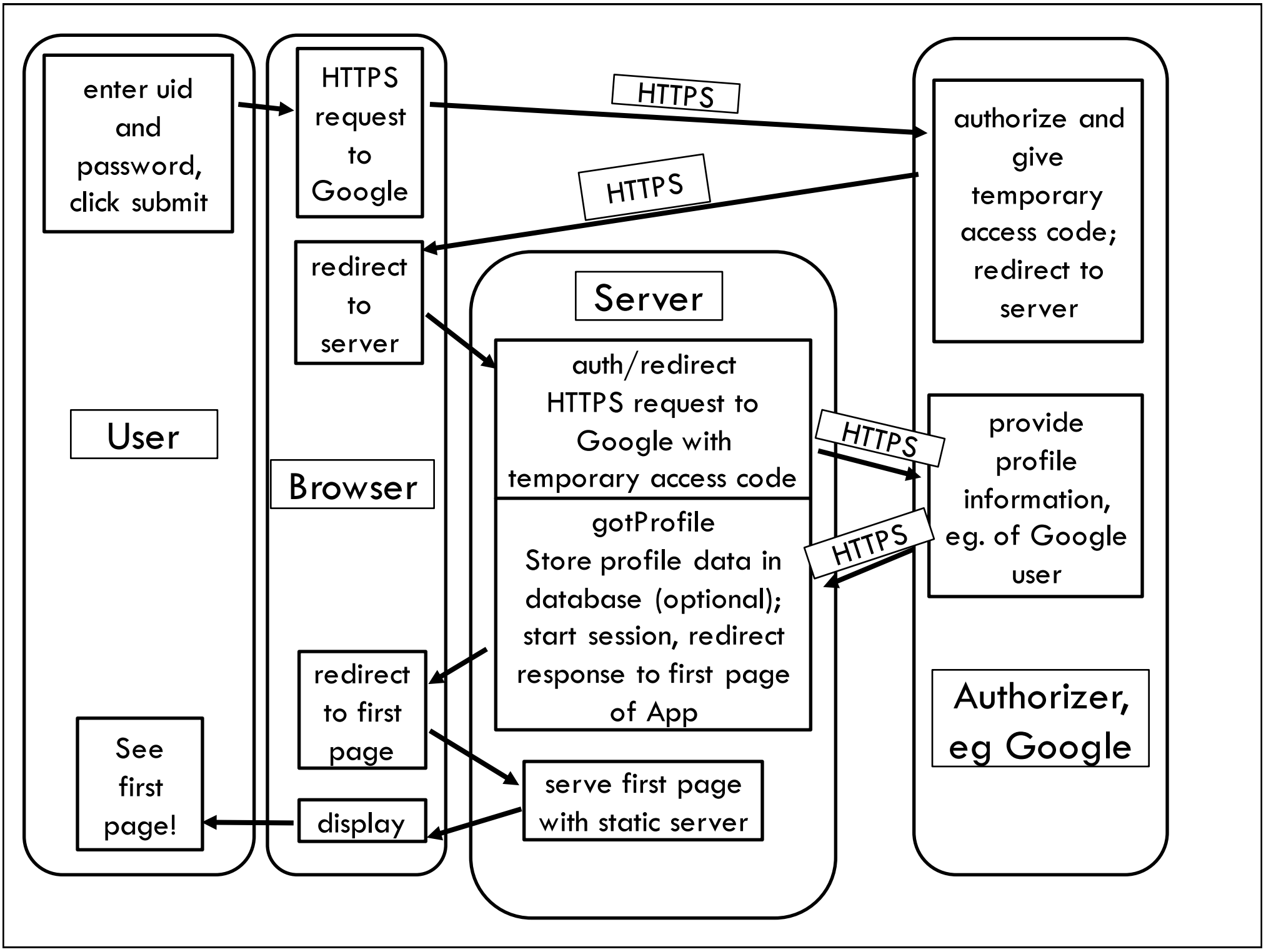
- Why does it fail when I try to start off login by an AJAX request to:

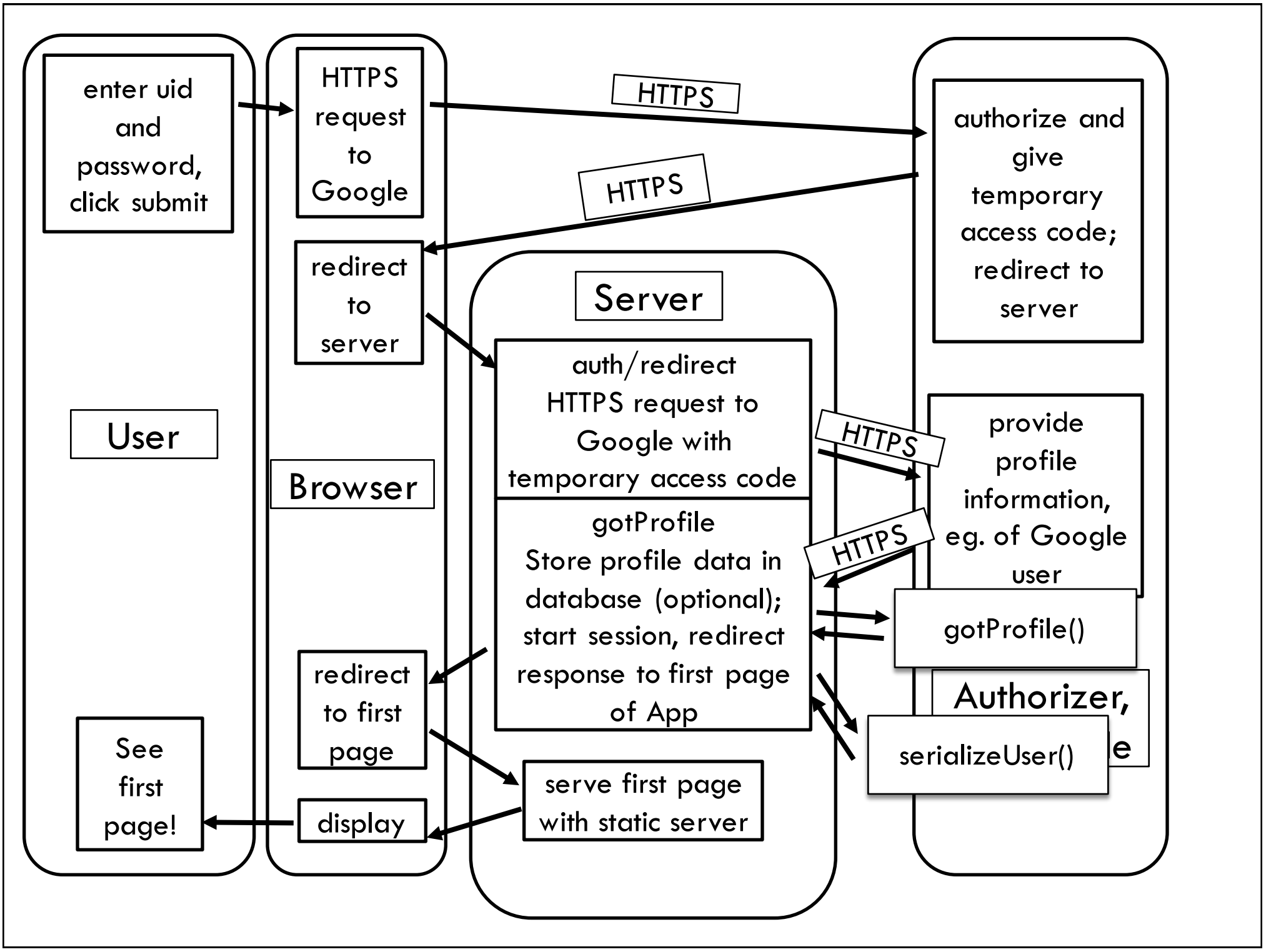
  server162.site:[port]/auth/google

The end result of this command is a redirect to Google. The Browser won't redirect an AJAX request because she applies the SOP to AJAX requests.

Comic login part 2




enter uid and password, click submit
HTTPS request to Google
HTTPS
authorize and give temporary access code; redirect to server
HTTPS
redirect to server
Server
auth/redirect HTTPS request to Google with temporary access code
User
Browser
HTTPS
provide profile information, eg. of Google user
gotProfile Store profile data in database (optional); start session, redirect response to first page of App
HTTPS
redirect to first page
Authorizer, eg Google
See first page!
display
serve first page with static server


enter uid and password, click submit
HTTPS request to Google
HTTPS
authorize and give temporary access code; redirect to server
HTTPS
redirect to server
Server
User
Browser
auth/redirect HTTPS request to Google with temporary access code
HTTPS
provide profile information, eg. of Google user
gotProfile
Store profile data in database (optional); start session, redirect response to first page of App
HTTPS
gotProfile()
redirect to first page
Authorizer,
serializeUser()
See first page!
display
serve first page with static server

gotProfile

- Called as soon as HTTPS request to Google returns the profile. The Server has to store the profile info in the Database. This is up to us; Passport knows nothing of our Database.
- The function done(), passed in by Passport, is it's callback – it passes control back to Passport.

```
done(null, dbRowID);
```

- Passport will pass the second argument of done() as the input to serializeUser().

serializeUser

- Prepares Passport for session cookie handling.
- Main question it has to answer: when Passport sees the session cookie for this user, what data should it attach to the req object?
- Generally, some pointer to the user information; in this case the primary key for the user in the User database table would be good.

deSerializeUser

- Called every time an HTTP request from this user is processed.
- Called by the passport.session() middleware function.
- Can use the user's primary key to get data out of database at this point, or can just pass the primary key on to later Server pipeline stages that might want it.
- Either way, the second argument to done() becomes req.user, where later stages can find it.

Kinds of attacks

- Man-in-the-middle
  - use HTTPS
- Cross-site request forgery
  - Hacker needs to evade Same Origin Policy
  - How can this happen?
- Injection
  - Cross-site scripting (HTML/JS injection)
  - SQL injection

Forms do not obey the SOP

- One common way of getting around the SOP is by using a form.

```
<form action="/query" method="get">
   Name:    <input type="text"  name="yourName">
   <input type="submit"  value="Submit">
</form>
```

Form standard behavior

```
<form action="/query" method="get">
   Name:    <input type="text"  name="yourName">
   <input type="submit"  value="Submit">
</form>
```

- Say the user types "Maude" and hits submit.
- An HTTP GET request is sent, with the following URL:

  server162.site:[port]/query?yourName=Maude
- Don't answer these unless you know they are legit!

Form standard behavior

- This is not an AJAX request.
- The Server is expected to respond with a redirect, so there is no way to set a callback for form requests
- (You can override the form standard behavior and turn it into an AJAX request, or anything else you can do in Javascript.)

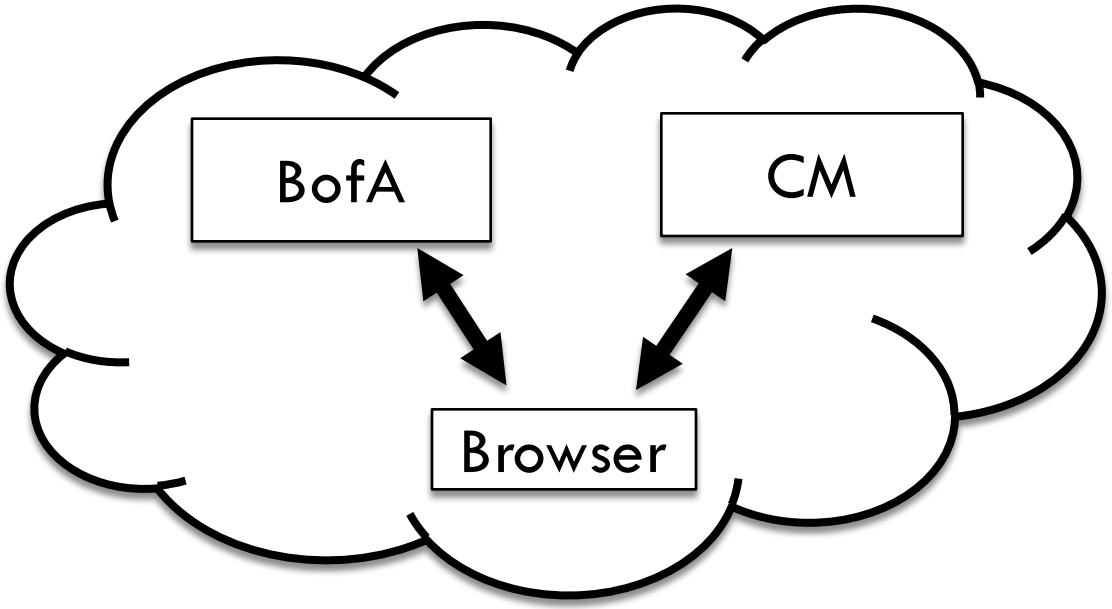
- Only AJAX requests have to obey the SOP.
- So this form request can be sent anywhere!

Example

```
<form action="https://www.google.com/">
    Search:
    <input type="text"  id="searchBox" name="q">
    <input type="submit"  value="Submit">
</form>
```

- Redirects to Google to search for whatever the user types.

Example



- Celebrity Makeovers has form with default values that mimics getting BofA to send them a check.
- The form has "display: hidden".
- The submit button is labeled "Can Mehgan lose the baby fat?"
- The form action sends it to BofA.

Defense

- We just hope that BofA is smart enough to defend itself against CSRF attacks.
- How do we do that?

Defense

- We just hope that BofA is smart enough to defend itself against CSRF attacks.
- How do we do that?

Include some kind of un-guessable token in the body of every HTTPS request from the browser, for instance the session cookie or some function of the session cookie. The server ignores the request if this token is missing. This strategy is sometimes called Csurf.

Injection attacks

- Putting text you do not completely control into any context where it will be executed is a security hole.
- Example:

```
let text =
  document.getElementById("entry").value;
let target = document.getElementById("tDiv");
target.innerHTML = text;
```

- The use of innerHTML is unnecessary here; use textContent instead. The contents of innerHTML are executed, textContent is not.

Injection attacks

```
let text =
        document.getElementById("entry").value;
let target = document.getElementById("tDiv");
target.innerHTML = text;
```

- User could type something executable in the box.
- The Browser is smart enough **not to execute**

  ```
  <script>alert("XSS Attack");</script>
  ```

- But it executes the error function in

  ```
  <img src=x onerror="alert('XSS Attack')">
  ```

Why exactly is this bad?

- Isn't the user attacking herself? Not our problem!

- Two issues:
  - A hacker can print out any internal data the Browser knows, eg. session cookie, which might help them figure out how to forge it.
  - If they can auto-fill the box somehow, and get the user to push the button, they can get the user to execute anything, eg. as a basis for a CSRF attack.

Autofill

- We can allow forms or text boxes to be filled in from the URL
- This is an issue for forms (CSRF directly) or user input that gets executed (arbitrary XSS).

```
let query = window.location.search.substring(1);
let fields = query.split("=");
if (fields[0] == "name") {
    document.getElementById("usersName").value = fields[1];
}
```

Autofill

- This URL fills the name box:

  http://server162.site:[port]/flaw.html?name=Maude

- Now consider Celebrity Makeovers, with a button labeled "Shaun White's New Look!", which is really a link:

  <a href=

  "http://server162.site:[port]/flaw.html?name=

  <img src=x onerror='anyFunctionInOurCode'>"

- All our Browser code is totally public, so a hacker can put a lot of effort into finding exploits.

Built-in browser protections

- URI encoding replaces most special characters with codes, so it is hard to get most code into a page that way.
- Some browsers have additional checks.
- There are a number of React add-ons for checking user input.

Auto-fill *by browser*

- Browsers fill in often very confidential information, to make filling out forms easier. Possible fields from HTML5 Standard:

`"cc-name"`	Full name as given on the payment instrument	Free-form text, no newlines	Tim Berners-Lee	Text
`"cc-given-name"`	Given name as given on the payment instrument (in some Western cultures, also known as the *first name*)	Free-form text, no newlines	Tim	Text
`"cc-additional-name"`	Additional names given on the payment instrument (in some Western cultures, also known as *middle names*, forenames other than the first name)	Free-form text, no newlines		Text
`"cc-family-name"`	Family name given on the payment instrument (in some Western cultures, also known as the *last name* or *surname*)	Free-form text, no newlines	Berners-Lee	Text
`"cc-number"`	Code identifying the payment instrument (e.g. the credit card number)	ASCII digits	4114360123456785	Text
`"cc-exp"`	Expiration date of the payment instrument	Valid month string	2014-12	Month

Other code injections

- SQL injection is a long-standing problem, always sanitize database inputs.
- CSS is code.
- Some graphics are code, eg. SVG.

- We use many modules. Security holes in modules are security holes in our apps. Using sketchy apps increases this danger.