Building Secure User Interfaces using JWTs

Robert Damphousse

JavaScript Lead, Stormpath

Talk Overview

Security considerations for web applications

Cookies: need to know

Infrastructural issues with session identifiers

JWT to the rescue!

Demo Time - Angular + JWT

User Interface = Web App

(if it's running in your browser)

- HTML, CSS and Javascript that makes up an application that runs in your browser, aka **the client**
- They need a server to serve them!
- The server may be dynamically rendering some of the content, aka **"Server Side Rendered"**
- The application may be completely static, aka **"Single page Apps"**

Security Concerns For Web Apps

- I need to prevent malicious code from running in my users' browsers
- My user's credentials need to be kept confidential and not exposed to attackers
- My server endpoints (the API) need to be protected from unauthorized access
- Access control information is required by the client application

I need to prevent malicious code from running in my users' browsers

This is a HUGE threat, and referred to as Cross-Site Scripting (XSS) Attacks

I need to prevent malicious code from running in my users' browsers

"Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. 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."

https://www.owasp.org/index.php/XSS

Demo

https://www.google.com/about/appsecurity/learning/xss/#BasicExample

XSS - What can I do??

Read EVERYTHING on this page:

https://www.owasp.org/index.php/XSS

XSS - What can I do??

If you are dynamically rendering HTML server-side:

Use well-known, trusted libraries to help you with escaping content, do NOT try to roll this yourself

XSS - What can I do??

If you have a Single Page App, your framework of choice probably does a lot of the work for you (preventing DOM based attacks by escaping user input).

But you should still read up on it.

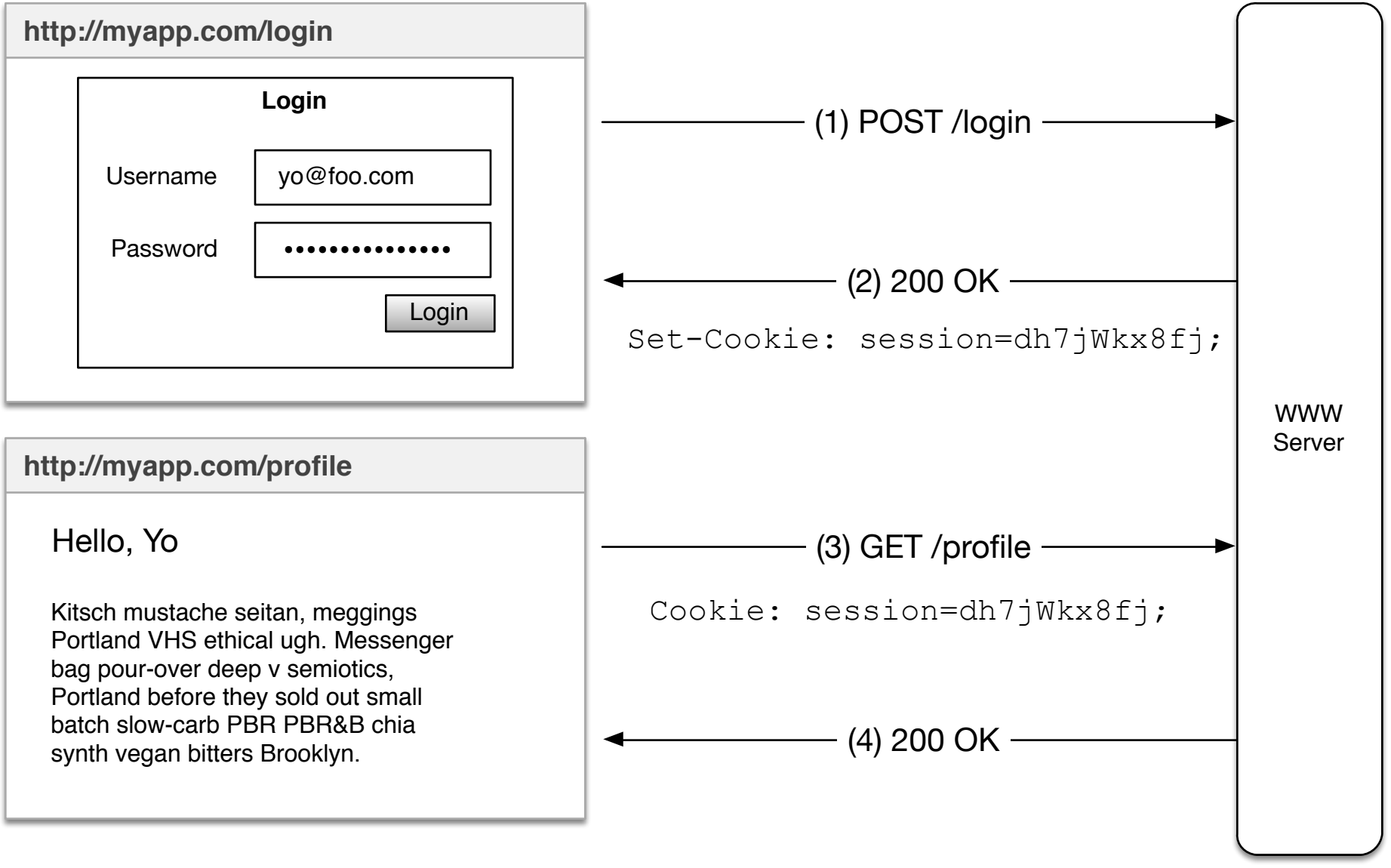
The user's credentials need to be kept confidential

The traditional solution is to use a cookie-based session identifier

This is okay, so long as you do cookies right

Later on, we'll tell you why JWTs are better :)

Cookie-Based Session Identifiers



The server endpoints (the API) need to be protected from unauthorized access

Again, the traditional solution is to use a cookie-based session identifier

Access control information is required by the client

Traditional solution:

- Store this information in your DB and link it to the session ID
- Provide a “/me” route which reflects this information

We’ll show you how access tokens can solve these problems in a simpler fashion

Let’s talk about cookies..

Cookies are OK! If you do them right

It's really easy for them to be compromised, the common attacks being:

- Man-in-the-middle attacks
- Cross-Site Request Forgery (CSRF)

Man in The Middle (MITM) attacks

When someone is “listening on the wire” to the requests between the browser and the server.

Solutions:

- Use HTTPS everywhere
- Secure your internal networks, too!
- Love your IT & DevOps teams

Cross-Site Request Forgery (CSRF)

“Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious Web site, email, blog, instant message, or program causes a user’s Web browser to perform an unwanted action on a trusted site for which the user is currently authenticated”

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Cross-Site Request Forgery (CSRF)

In other words…

Foreign websites can make requests of your server, causing your user's cookies to be automatically sent yo your server, thus allowing the Foreign initiator to perform any action that your server allows with that cookie

(Yes, you should be freaking out)

Cross-Site Request Forgery (CSRF)

So how do we deal with this?

While the browser WILL send your cookies, even from other pages, those pages CANNOT see what's inside the cookies on your domain.

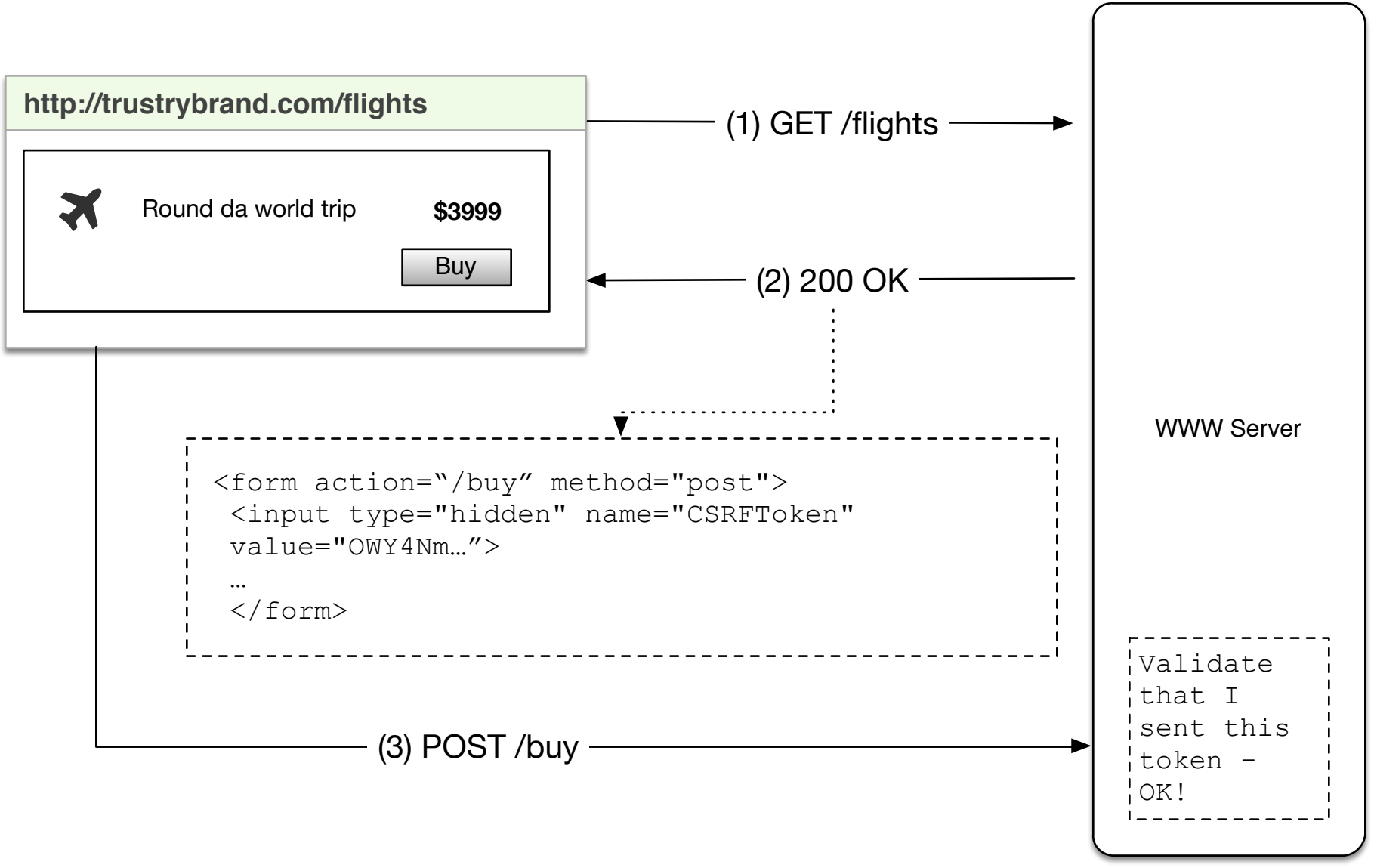
Alas! You want to do something which ensures that it's a page on YOUR domain that is making requests with your cookies

Cross-Site Request Forgery (CSRF)

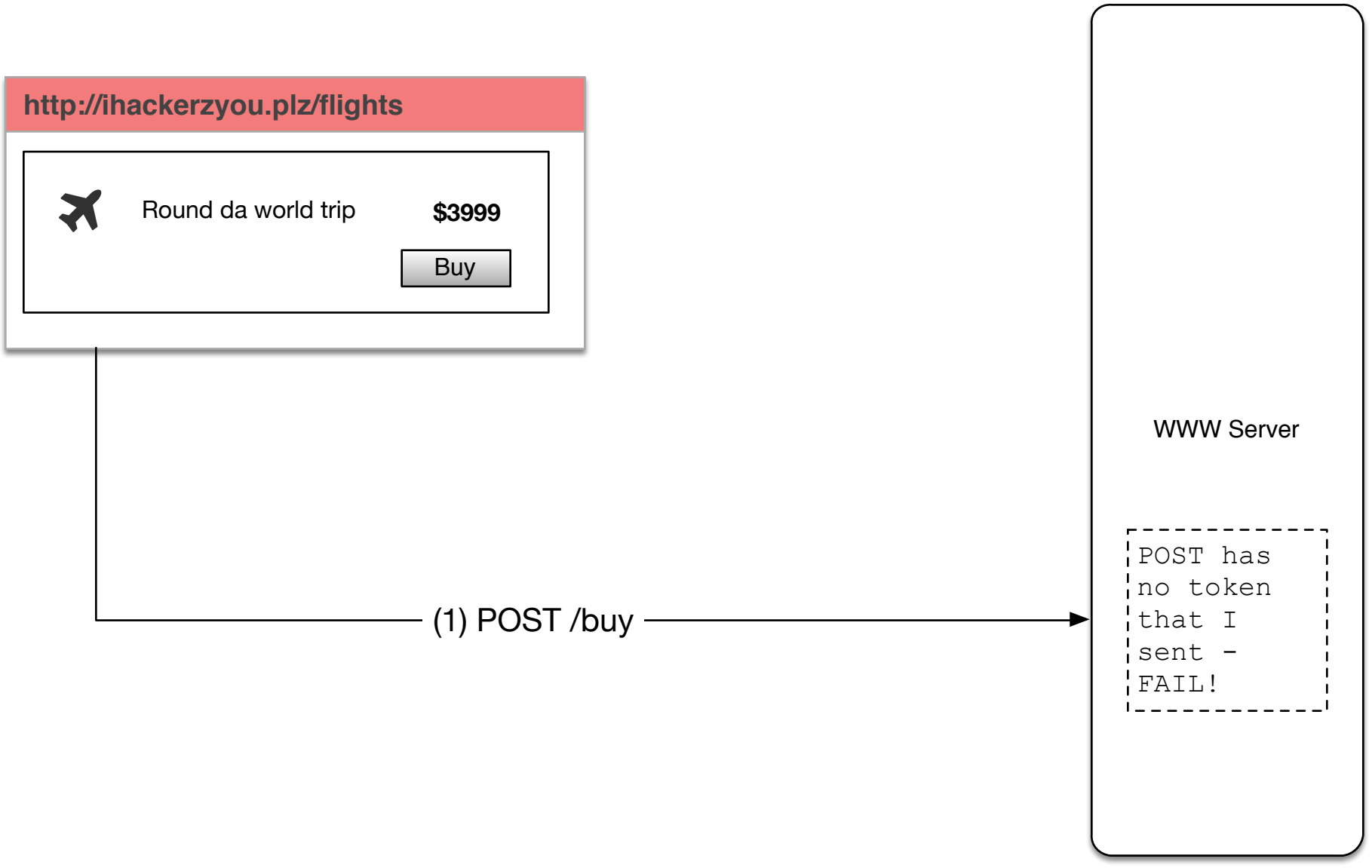
Solutions:

- Synchronizer Token
- Double-Submit cookie
- Referer header check
- Origin header check

Synchronizer Token - Trusted Page



Synchronizer Token - Foreign Page



Synchronizer Token - Considerations

Requires cooperation from your rendering layer

Requires you to store tokens in a data store

Difficult to do with static SPA content

Only protects against forged POST requests, not GET requests!

Pro tip: don't allow GET requests to modify server state!

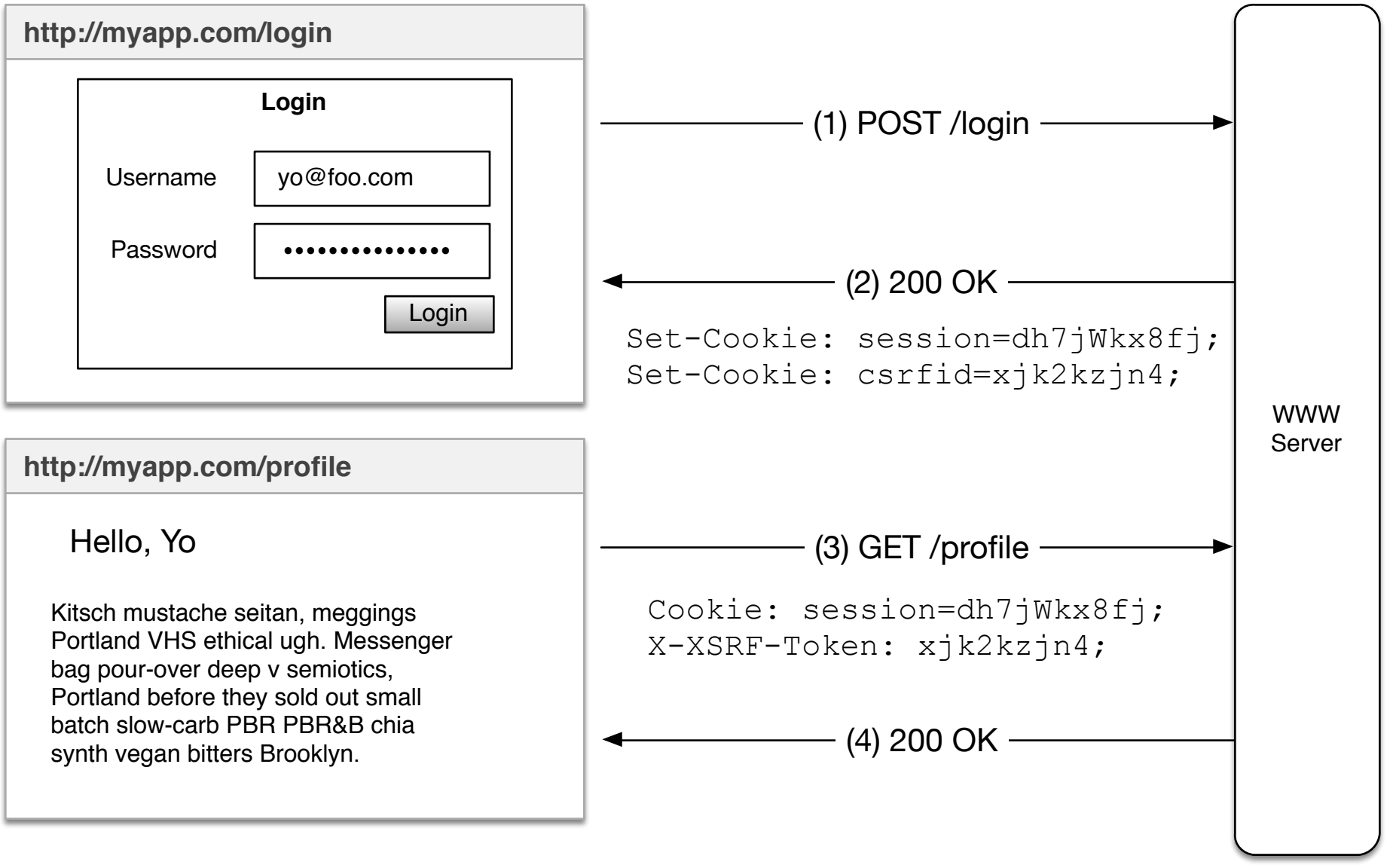
Double Submit Cookie

You send two cookies: the session ID and a random value

Your client application should send this random value, using a channel which cannot be used by a foreign site. It should be a mechanism that follows the browsers Same-Origin-Policy

Reading a cookie from your domain and sending it's value as a custom header (or field form) is the best way to do this

Double Submit Cookie



Double Submit Cookie - Considerations

You can send the token in a custom HTTP header as part of the request payload, do what makes sense for your architecture

Still vulnerable to XSS! Because you need to expose the token to the JS environment in order for this to work.

Protect against XSS!

Referer Check

Browsers send a `Referer` header when they make requests of your server. It reports what page you are looking at right now.

This value is easy to hack in the browser

Origin Check

Browsers send a `Origin` header when they make requests to a domain that is not the same domain of the current page. It reports the domain of the page that is making the request

Cannot be hacked by the browser, but could be modified by a malicious HTTP proxy server.

Midpoint Summary

The typical solution to authentication concerns for web apps is to put a session identifier in a cookie.

Cookies can be used against you, but there are known solutions to the problem.

Talk Overview

Security considerations for web applications

Cookies: need to know

Infrastructural issues with session identifiers

JWT to the rescue!

Demo Time - Angular + JWT

Session Identifiers and your Infrastructure

You need to store that session ID somewhere! And you need to look it up for every request. Uber :(for high rate API services

Session Identifiers and your Infrastructure

Your client application has NO IDEA what it's allowed to access or not. It needs to make another request of your server for this information. Super annoying when you need to quickly bootstrap an application in a "Admin" vs. "User" view

Session Identifiers and your Infrastructure

They're opaque! They have no meaning elsewhere. This becomes a problem in a service-oriented architecture: you need a centralized ID de-referencing service.

It's also a problem for SaaS companies: how do you know what my ID means to your system?

JSON Web Tokens To The Rescue

(JWT)

JSON Web Tokens (JWT)

They are the “implementation detail” of the “access tokens” that are referred to by the Oauth2 specification

They JWT spec is still a draft, but that doesn’t ever stop us now does it??

They are a self-contained string that is signed with a secret key. They contain a set of “claims” which assert an identity and a scope of access

JSON Web Tokens (JWT)

You can store them in cookies! But all those cookie rules still apply.

In fact, you can simply replace your opaque session identifier with a JWT

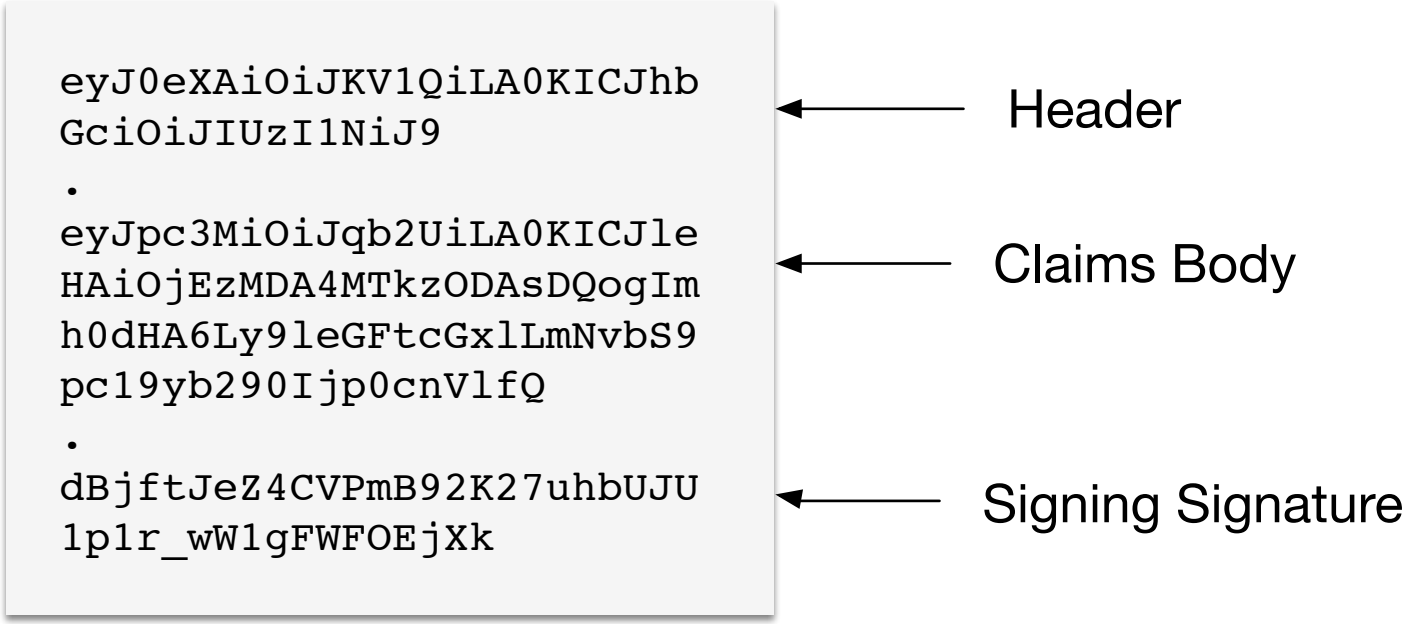
JSON Web Tokens (JWT)

In the wild they look like just another ugly string:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJ
pc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQo
gImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnV
lfQ.dBjftJeZ4CVPmB92K27uhbUJU1p1r_wW1gFWFOEj
Xk
```

JSON Web Tokens (JWT)

But they do have a three part structure (Each part is a Base64 Encoded string)



JSON Web Tokens (JWT)

Decode them to find the juicy bits

```
{
    "typ":"JWT",
    "alg":"HS256"
}
```

Header

```
{
    "iss”:”http://trustyapp.com/”,
    "exp":1300819380,
    “sub”:””users/8983462,
    “scope”: “self api/buy”
}
```

Claims Body

```
tß´—™à%O˜v+nî…SZu¯µ€U…8H×
```

Signing Signature

JSON Web Tokens (JWT)

The claims body is the best part! It's telling you..

```
{
    "iss":"http://trustyapp.com/",
    "exp":1300819380,
    "sub":""users/8983462,
    "scope": "self api/buy"
}
```

◄····· Who issued this token

JSON Web Tokens (JWT)

The claims body is the best part! It's telling you..

```
{
    "iss":"http://trustyapp.com/",
    "exp":1300819380,
    "sub":""users/8983462,
    "scope": "self api/buy"
}
```

◄····· Who issued this token

◄····· When it expires

JSON Web Tokens (JWT)

The claims body is the best part! It's telling you..

```
{
    "iss":"http://trustyapp.com/",
    "exp":1300819380,
    "sub":""users/8983462,
    "scope": "self api/buy"
}
```

◄····· Who issued this token

◄····· When it expires

◄····· Who this token is for

JSON Web Tokens (JWT)

The claims body is the best part! It's telling you..

```
{
    "iss":"http://trustyapp.com/",
    "exp":1300819380,
    "sub":""users/8983462,
    "scope": "self api/buy"
}
```

◄····· Who issued this token

◄····· When it expires

◄····· Who this token is for

◄····· What this person can do

JSON Web Tokens (JWT)

Great! Why is it useful to pack all this into an ugly string?

- The information can be implicitly trusted because it's signed - it saves roundtrip communication to your authentication service
- It's structured, enabling inter-op between services
- It can inform your client about basic access rules e.g. "I'm in the admin group because I have admin scope

JSON Web Tokens (JWT)

Okay, what's the catch?

- **Implicit trust is a tradeoff** - how long should that token be good for, how you will revoke it? For another talk: refresh tokens
- You still have to secure your cookies!
- You have to be mindful of what you store in the JWT if you don't use encrypted JWTs - i.e. don't put a full account object with a shipping address in there!

Demo Time!

And end-to-end demo of a JWT infrastructure with the following technologies:

- AngularJS
- ExpressJS (Node.js)
- Stormpath API