Web Development

COMP 431 / COMP 531

Lecture 22: Third Party Authorization

Instructor: Mack Joyner

Department of Computer Science, Rice University

mjoyner@rice.edu

http://www.clear.rice.edu/comp431

Part II – Back End Development

Homework Assignment 6
(Draft Back-End)
Due Thursday 11/16

COMP 531
Paper and Presentation
Due Tuesday 11/28

In-Class Exercise: Integrate front and back

- Spin up your backend. Spin up your frontend. Open the JS console
- From your *frontend* try to access your *backend*, e.g., **`GET /headlines`**
  - Log in on landing page, main page gets headline from backend
- We need CORS activated for the frontend to talk to the backend
  - The browser is connected to frontend, therefore backend is a different origin (port)
- CORS is enabled with headers: Access-Control-Allow-…
- Create a *middleware* function that sets these headers:

-Allow-Origin	origin of the requestor
-Credentials	true
-Methods	… what do you think we'll want to allow?
-Headers	Authorization, Content-Type, perhaps others?

- If the request method is **`OPTIONS`** (preflight) then we return status **`200`**
- Now try **`GET /headlines`** and see if it works from the frontend

`/inclass-21/index.js` *contains your CORS middleware*

In-Class Exercise: Integrate front and back solution

https://www.clear.rice.edu/comp431/sample/index_sol21.js

COMP 531 Paper and Presentation

- Topic

  Web Development or Design
  - New technology
  - Technology comparison
  - Site design analysis
  - Enterprise in the Web
  - E-commerce
  - User experience
  - User interfaces
  - ReflectJS, Security, BigData

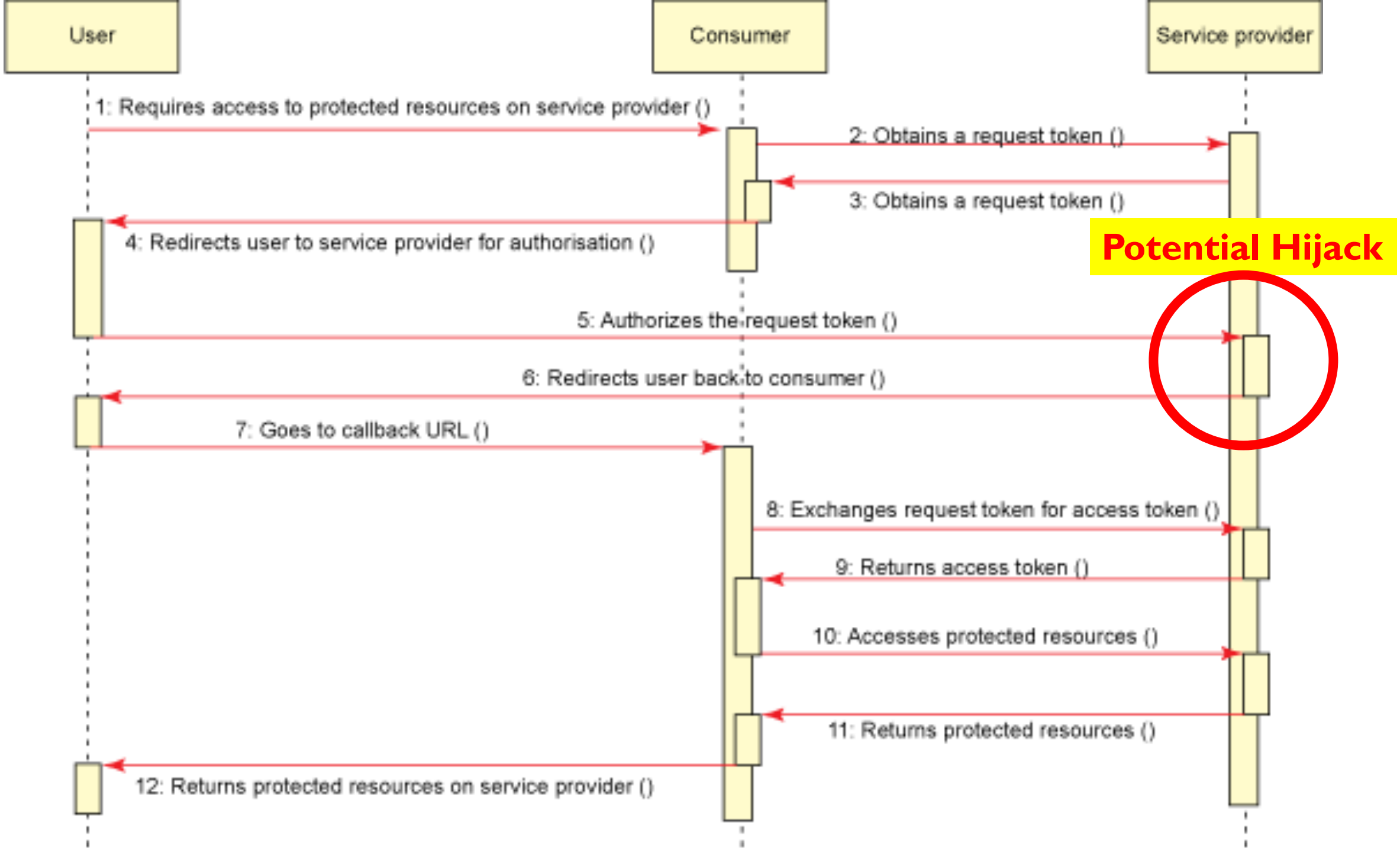
- Paper
  - 1000 to 2000 words
  - Proof read
  - Review and revise
  - Spelling and grammar
  - Think of it as a *blog post* that your future boss will read

- Presentation
  - No more than 5 minute talk
  - slides, web sites, demos, props, etc...

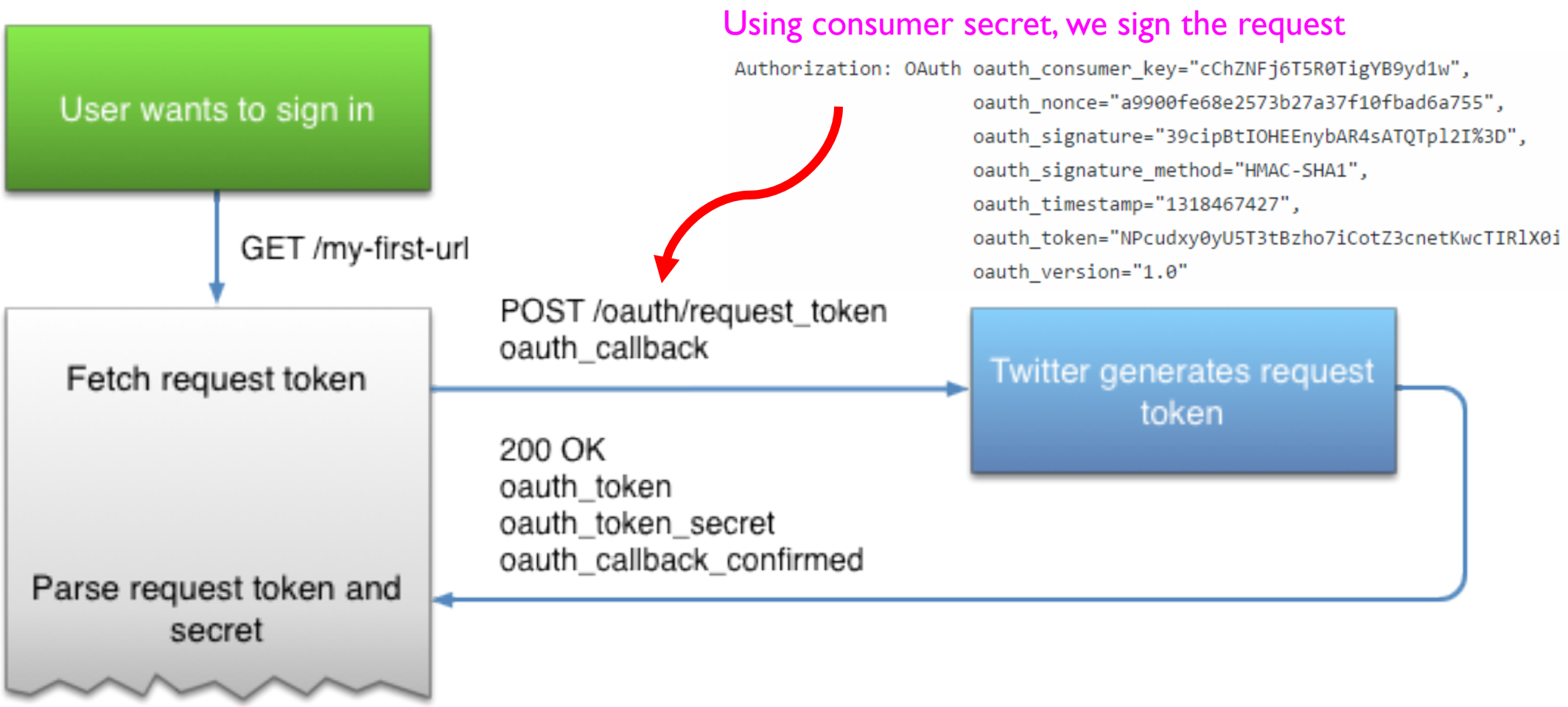
post your idea on Piazza – there will be no duplicate topics

Open standard for Authorization (OAuth)

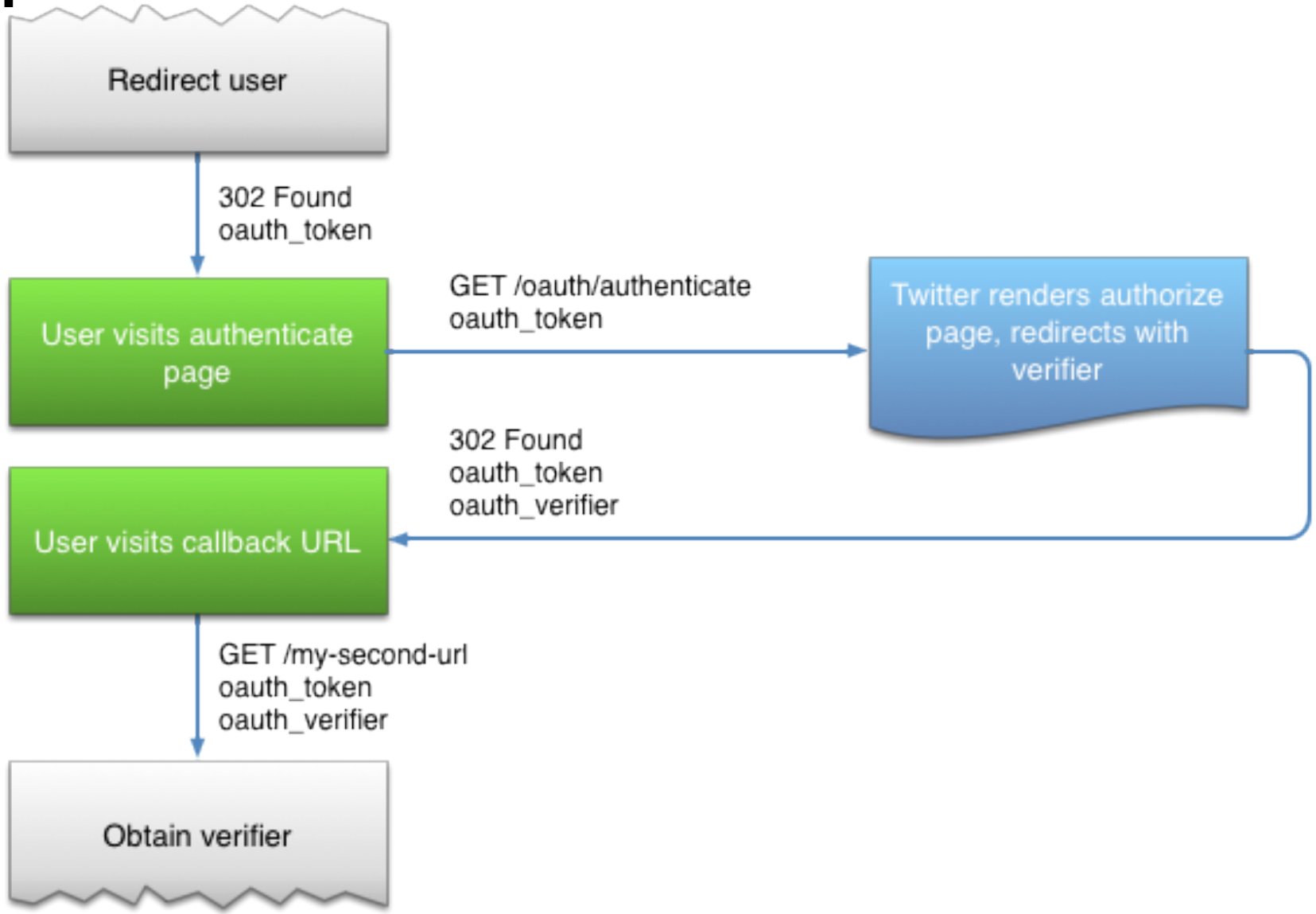
- Credential delegation
- Resource Owner has relationship with Third-Party Authorization Server
- User authenticates with Third-Party
- Third-party issues access token for user to access Resource


User
Consumer
Service provider
1: Requires access to protected resources on service provider ()
2: Obtains a request token ()
3: Obtains a request token ()
4: Redirects user to service provider for authorisation ()
Potential Hijack
5: Authorizes the request token ()
6: Redirects user back to consumer ()
7: Goes to callback URL ()
8: Exchanges request token for access token ()
9: Returns access token ()
10: Accesses protected resources ()
11: Returns protected resources ()
12: Returns protected resources on service provider ()

Example with Twitter: 1. Get Request Token



Example with Twitter: 2. Redirect User for Login



https://api.twitter.com/oauth/authenticate?oauth_token=ihOnQAAAAAAAh7E0AAABU

Sign up for Twitter ›

Authorize webdev-dummy to use your account?

webdev-dummy

webdev-dummy.herokuapp.com

Dummy Web Server

Username or email

Password

Remember me · Forgot password?

Sign In | Cancel

This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Will not be able to:

- Access your direct messages.
- See your Twitter password.

https://<CALLBACK_URL>?oauth_token=ihOnQAAVA71E0AAABUMR9phM&oauth_verifier=lhav0JQnDtV4APqfhKxk15ZB4wwB

Example with Twitter: 3. Convert to Access Token

<u>Sweet and simple</u>

Now store the token … in a cookie?

What did we do?

- Traded username and password for token key and secret
- Why is this good?
  - Because the user doesn't have to remember another password
- How does this help us?
  - Encrypt the token key and secret and store in browser (e.g., cookie)
  - This at least alleviates us from caching these values on the server
  - MITM/CSRF can still hijack the user's tokens

OAuth2

- v2 is always better than v1?

hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/

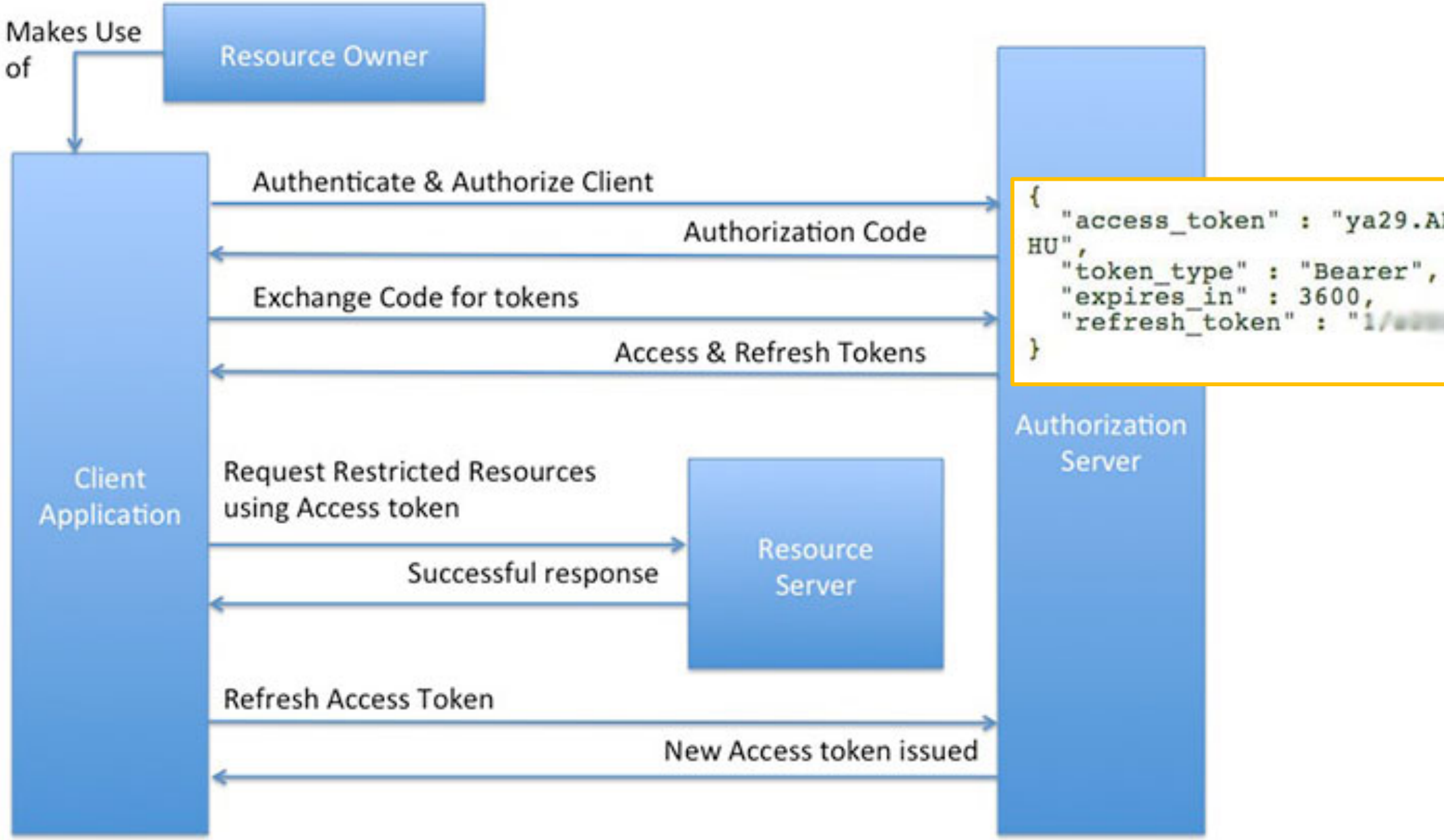
OAuth 2.0 and the Road to Hell

They say the road to hell is paved with good intentions. Well, that's OAuth 2.0.

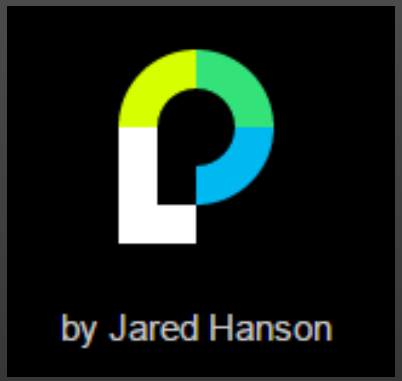
Last month I reached the painful conclusion that I can no longer be associated with the OAuth 2.0 standard. I resigned my role as lead author and editor, **withdraw my name from the specification**, and left the working group. Removing my name from a document I have painstakingly labored over for three years and over two dozen drafts was not easy. Deciding to move on from an effort I have led for over five years was agonizing.

Big Differences

- OAuth 2 should sit inside TLS/SSL
  - *Therefore no signing of every request (curl)*
- Native Applications (mobile apps, desktop apps, TV / game consoles)
  - *OAuth 1 was designed for browsers*
- OAuth 1 had "inherent complexity" and was "hard to use"
- Separation of Authentication Server from Resource Server
- Tokens have finite lives and must be refreshed


Makes Use of
Resource Owner
Client Application
Authenticate & Authorize Client
Authorization Code
Exchange Code for tokens
Access & Refresh Tokens
Request Restricted Resources using Access token
Successful response
Resource Server
Refresh Access Token
New Access token issued
Authorization Server
{
"access_token" : "ya29.A
HU",
"token_type" : "Bearer",
"expires_in" : 3600,
"refresh_token" : "1/
}

PassportJS



- One stop shopping:

  HTTP Basic, HTTP Digest, OAuth, OAuth 2.0, OpenID, Facebook, Twitter, Google, … over 300 strategies…

Use const not var!
Only include what you need

```
var request          = require('request')
var qs               = require('querystring')
var express          = require('express')
var cookieParser     = require('cookie-parser')
var session          = require('express-session')
var passport         = require('passport')
var FacebookStrategy = require('passport-facebook').Strategy;

app = express()
app.use(session({ secret: 'thisIsMySecretMessageHowWillYouGuessIt'}))
app.use(passport.initialize());
app.use(passport.session())
app.use(cookieParser());
```

```
// serialize the user for the session
passport.serializeUser(function(user, done) {
  users[user.id] = user
  done(null, user.id)
})

// deserialize the user from the session
passport.deserializeUser(function(id, done) {
  var user = users[id]
  done(null, user)
})

passport.use(new FacebookStrategy(config,
  function(token, refreshToken, profile, done) {
    process.nextTick(function() {
      // register the user in our system
      return done(null, profile)
    })
  })
)
```

users[] is a proxy for db.

Serialize converts userObj to id
Deserialize converts id to userObj

```
function logout(req, res) {
  req.logout();
  res.redirect('/')
})

function isLoggedIn(req, res, next) {
  if (req.isAuthenticated()) {
    next()
  } else {
    res.redirect('/login')
  }
}

function profile(req, res) {
  res.send('ok now what?', req.user)
}
```

by Jared Hanson

```
app.use('/login',    passport.authenticate('facebook', { scope:'email' }))
app.use('/callback', passport.authenticate('facebook', {
  successRedirect: '/profile', failureRedirect :'/fail' }))
app.use('/profile', isLoggedIn, profile)
app.use('/fail', fail)
app.use('/logout', logout)
app.use('/', hello)
```