Authentication

UNC COMP 523

October 30, 2019

Jeff Terrell

What is authentication?

Authentication: the process of proving an identity.

- Example: you must prove that you're really you before you can access your email inbox.

Types of authentication

- Token-based
  - Driver's License
  - Passwords
- Biometric
  - TouchID or fingerprint scanning
  - FaceID on newer iPhones
  - Retinal scan, voice print, etc.
- Multi-factor

Authentication vs. Authorization

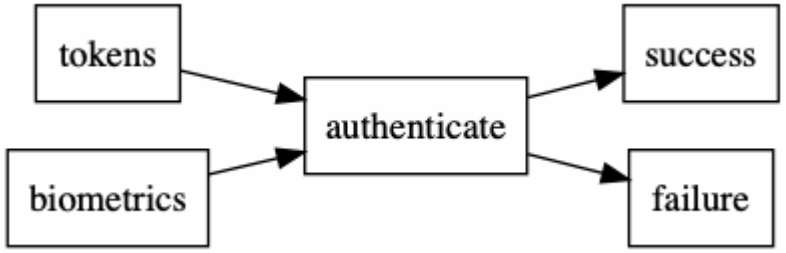
- *Authentication* is proving an identity
- *Authorization* is enforcing access restrictions (e.g. "employees only")
- Access to protected areas usually requires authentication

Authentication vs. Security

- Security is preventing unintended access or uses of the data or the software
- Authentication [should] involve security
- But security is broader

Authentication as a decision

- Authentication can be thought of as a *decision*



- We'll learn more today about what's in the middle

Identifiers

- A key component of auth[entication] is an *identifier*
- Examples: SSN, server-assigned numeric ID, username, email address
- Recommended: email address
- Recommended: `citext` data type in PostgreSQL (case-insensitive text)
- Pro tip: `jeff@email.com`, `jeff+admin@email.com`, and `jeff+anythingelse@email.com` all route to the same place
- Food for thought: you might should *verify* an email address before sending (more than 1) email to it

Passwords

- Don't store passwords as plain text (why?)
- Also, don't just store a hash of the password
- *Rainbow tables* are pre-computed lookup tables for common passwords and their hashes
- Countermeasure: *salts*: random info added to each password, incorporated into the hashed value
- Recommendation: use `bcrypt` (bonus: use a password manager! e.g. Bit Warden)

bcrypt

Example bcrypt password:

> `$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy`

- `$2a$` - bcrypt identifier and version
- `$10$` - *cost*: how hard must I work to check a password against this? (hinders guessing attacks)
- `N9qo8uLOickgx2ZMRZoMye` - the salt for this password (prevents rainbow table attacks)
- `IjZAgcfl7p92ldGxad68LJZdL17lhWy` - the hash

After the authentication decision

- What comes after the authentication decision?
- If identity not proven, start authentication process over.
- If identity proven, typically create a *session* for the user.

Sessions

- Upon successful authentication, server creates a random token and shares it with the client
- Client provides the token along with subsequent requests
- HTTP has a mechanism for this: cookies
- Cookies are scoped to a domain (e.g. `cs.unc.edu`), can have an expiration, and are added by the browser to requests to that domain
- Server side must remember the tokens (or *session IDs*) it provided, and what user is associated with that session
- Don't let session IDs get stolen: use HTTPS!

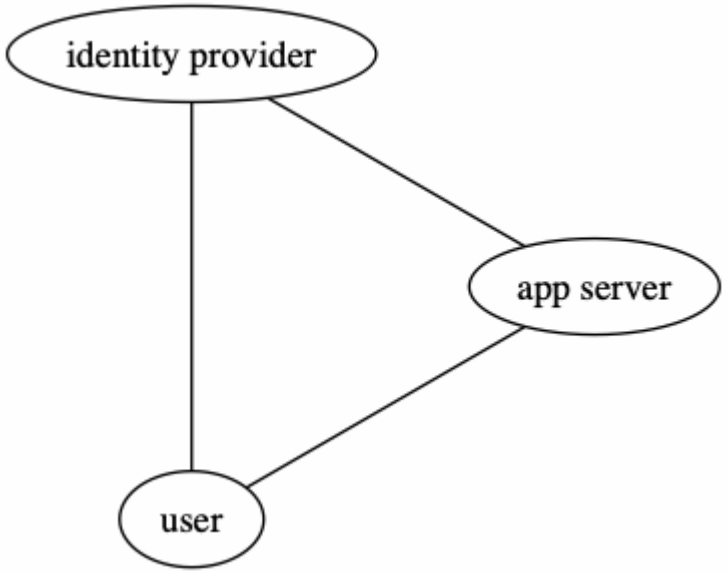
Difficulties of DIY auth

- Merely implementing the auth *decision* isn't usually enough
- What about verifying an email address? (Including email sending and storing/checking email verification tokens)
- What about forgotten passwords? (Including email sending, storing/checking password reset tokens, and password reset screens and endpoints)
- What about extra features, like multi-factor authentication?

Identity services

- You can also use a third-party identity service (e.g. "Log in with GitHub")
- Advantage: let somebody else worry about identity management and extra security features
- Disadvantage: the complicated protocol dance
- OAuth is a common protocol here (v1.0a and v2)
- UNC uses SAML for SSO (single sign-on)

OAuth parties



OAuth for users

- Click the "login with __" button
- Authenticate with the identity provider (if not already logged in)
- Consent to give the app access to your identity (and maybe also your data)
- Now you're logged in to the app

OAuth for identity providers

- Think-pair-share: what does the identity provider (e.g. GitHub) need to care about?
- Let users authenticate and manage their identity and authentication mechanisms
- Tell users about the app that's asking for their info, and which info it's asking for
- Provide random tokens to apps, remembering the token, the app, and the user
- Provide access to user info upon request, when the request is authenticated with a valid token

OAuth for app developers

- Register your app with the identity provider (incl. app name, app domain, permissions, and a callback URL), and save client ID and secret
- Add UI for "Log in with __" button, which redirects users to identity provider's site and includes your client ID, a request token you generated, and proof that you have access to the client secret
- Wait for user to get redirected to your callback URL with either rejection or approval and a token
- Store the token, and include it in subsequent requests for info for that user from the identity provider

Cross-site scripting attacks

- Remember how the browser automatically attaches cookies to requests? This is a potential security hole.
- Example: figure out which endpoint gmail uses to retrieve messages, then add code to your web app to hit that endpoint. If the user is logged in to gmail, you can retrieve their messages with your code!
- This is called a "cross-site scripting" (XSS) attack

Cross-origin resource sharing

- Ever wrestle with this error in your app?

> Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://some-url-here. (Reason: CORS header ‘Access-Control-Allow-Origin’ missing).

- This is part of CORS, the cross-origin resource sharing standard, designed to prevent XSS attacks
- Note: CORS only applies to HTTP requests initiated by Javascript code. Other HTTP requests don't need or use CORS.
- Also, CORS only affects *cross-origin* requests, i.e. requests from one domain to a different domain

CORS details

- The browser adds an `Origin` HTTP header indicating the protocol + domain that the request originated from, e.g. `https://myapp.com`
- For "simple" requests, the browser sends the actual request and looks at the response headers. If the `Access-Control-Allow-Origin` header isn't `*` and doesn't include the origin, the response is considered rejected, and no data from the response is available to Javascript.
- For non-"simple" requests, the browser sends a "preflight" request to check on permissions, using the HTTP `OPTIONS` method

CORS summary for devs

- Think about whether you want other web sites to be able to access your user's data.
- If so, adding a `Access-Control-Allow-Origin: *` header to all responses should suffice.
- Otherwise, change the `*` to something more specific, e.g. `https://myapp.com`
- You can get more specific about what kinds of cross-origin requests you allow.