Enterprise Programmering 2

Lesson 09:
Security in MicroServices

Prof. Andrea Arcuri

Goals

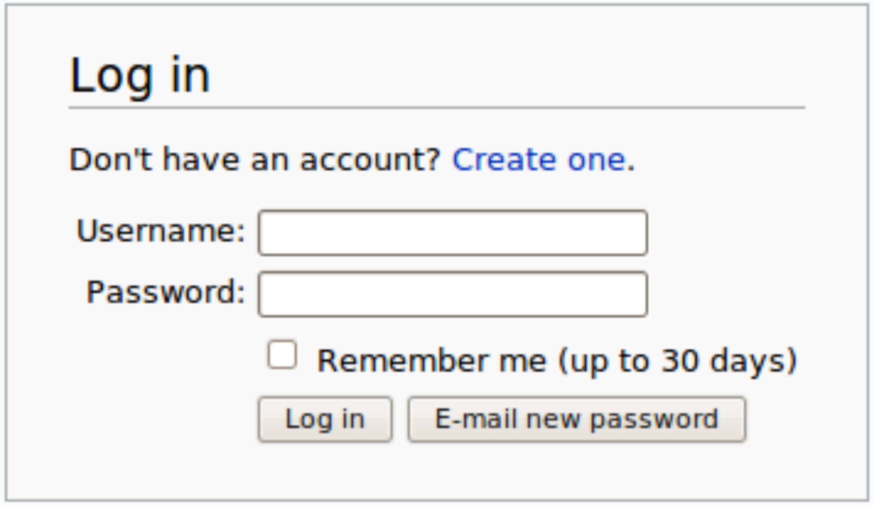
- Refresh knowledge on how to authenticate via HTTP
- Deal with *distributed sessions* in a microservice, where all instances share the same session for a given user

Authentication

- Server does not know who the user is
- Server only sees incoming HTTP/S messages
  - not necessarily from a browser… user can do direct TCP connections from scripts
- HTTP/S is stateless
- Need a way to tell that sequence of HTTP/S calls comes from same user
- User has to send information of who s/he is at **EACH** HTTP/S call
- But users can **lie**… (eg, hackers)

Ids and Passwords

- A user will be registered with a *unique* id
- Need also secret password to login
  - Otherwise anyone could login with the ids of other users…
- HTTP/S does not prevent attempts to login to accounts of other users



Log in

Don't have an account? Create one.

Username:

Password:

□ Remember me (up to 30 days)

Sending *id/pwd*

- Need to send it at **EACH** request
- Can put them inside the HTTP header *Authorization*
- Can be different formats to specify how *id/pwd* should be encoded
- *Basic* (RFC-7617): string "*id:pwd*" in Base64 encoding
- Ex *id=test* and *pwd=123£*, then header on **EACH** request:

Authorization: Basic dGVzdDoxMjPCow==

Problems

- Base64 is NOT encrypted… it is just a mapping from bits into printable ASCII codes
- When sending *id/pwd*, must use HTTP**S**
  - otherwise, anyone on the network can read them
  - anyway, always use HTTP**S** instead of HTTP…
  - but we do not use HTTP**S** in code examples, just due to complications of creating certificates
- What if someone intercepts a HTTP in clear, or has direct access to the browser (eg, via a malware)?
  - s/he will get the password

Authentication Token

- “Login” with *id/pwd* only **once**
- Server will return a *token* associated with that user *id,* stating s/he authenticated (assuming pwd was correct)
- From now on, instead of sending *id/pwd,* rather send the *token*
- Token will be valid only for a certain amount of time, after that, need to get new one via *id/pwd*
- *Benefits???*

Stolen Token

- If token is stolen, hacker can use it only for a *limited* amount of time, until it expires
- If user does **logout**, then token becomes invalid, and server will reject any further HTTP request with such token
  - so, even if hacker has the token, it will become *useless* for him/her
- *Critical* operations like changing password or transfer money could require a new login with *id/pwd*
  - and so hacker with stolen token cannot use it

Creating a Token

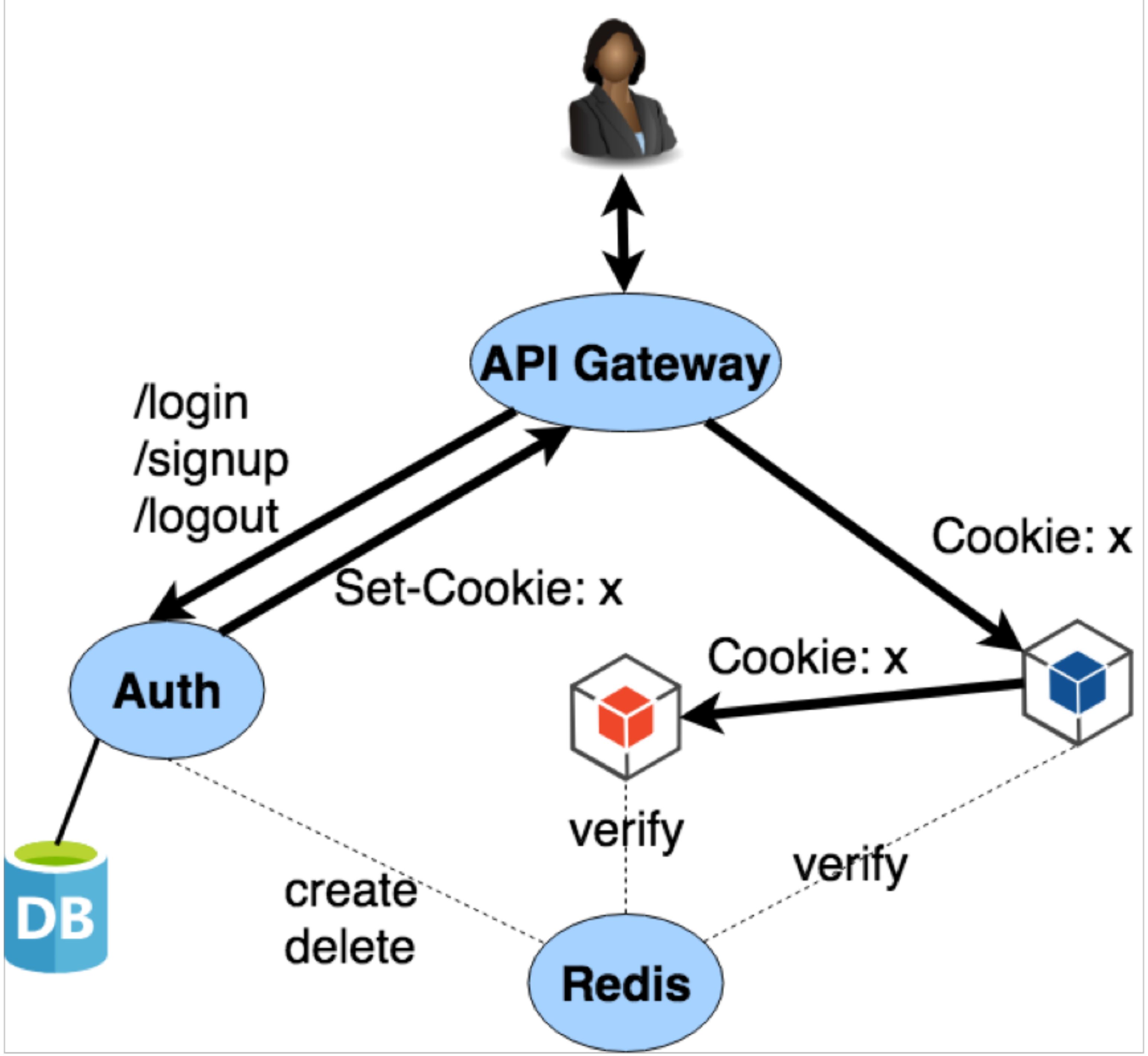
- Server could be instructed to create a token when receiving a HTTP request with header “*Authorization: Basic …*”
- This could be on any endpoint…
- … and/or could have a specific endpoint, e.g. “*/login*”
- But, in that case, I could choose how I want to send the *id/pwd* pair
- *POST /login {“id”: id, “password”: pwd}*
    - in SPAs, wants to send in *JSON* instead of *x-www-form-urlencoded* to help protecting from CSRF attacks

Sending/Receiving Tokens

- Browser needs to store authentication *tokens* somewhere
- Tokens need to be added at each HTTP request
- *Best* way to store tokens is HTTP **Cookies** marked with *HttpOnly*
  - automatically added on each HTTP request
  - cannot be read by JavaScript
- If you do *not* store authentication tokens in *HttpOnly* cookies, you are *more* vulnerable to **XSS** attacks!!!
  - Complex story… even with cookies, still vulnerable to XSS, but it would stop as soon as you close the browser… without cookies, token could be sent to malicious server via AJAX, and attacks continue from there
  - Note: this is a **huge** problem if you make the mistake of using JWT with no stateful whitelist/blacklist logout…

MicroService: Distributed Session

- Requests from user can go to many different services behind the gateway
- Should use a single authentication token, and not one for each service
- When service X speaks with Y, need to use the same kind of authentication that user would use when connecting to X and Y directly
- Session tokens stored in a database (eg *Redis*), accessible by all the services
    - *Authentication*: verify if received token is on *Redis*
    - *Login:* add token to *Redis*
    - *Logout:* remove token from *Redis*



- User first gets token from Auth (via *Set-Cookie*)
- Added at each request with *Cookie*
- At each request, services checking with *Redis*
- *Redis* must be fast

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **advanced/security/basic**
- **advanced/security/session**
- **advanced/security/distributed-session**
- Study relevant sections in RFC-7235 and RFC-7617