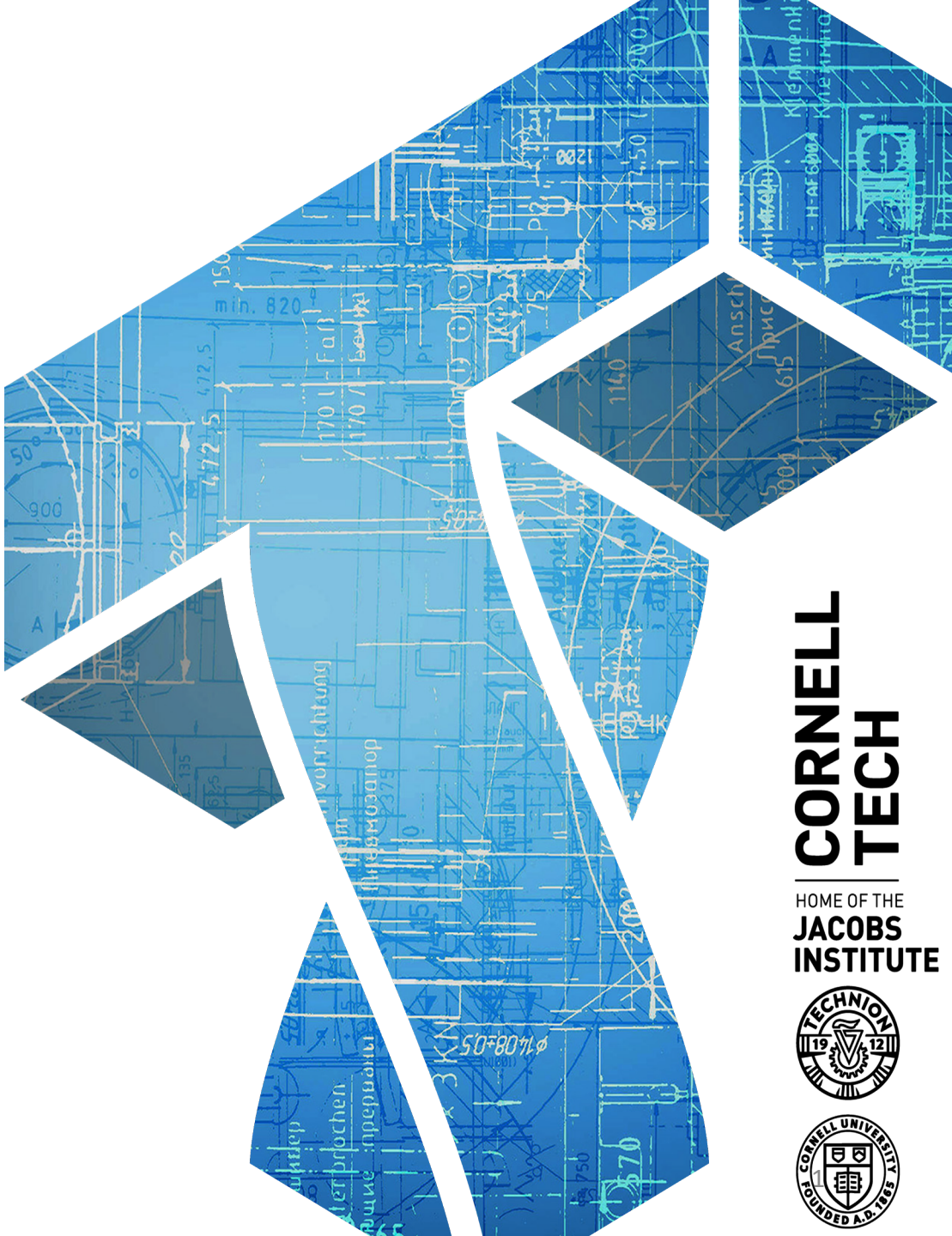
# CS 5435: Authentication beyond passwords & authorization

Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-fall2019



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

# Airbnb as an example service

Hosts

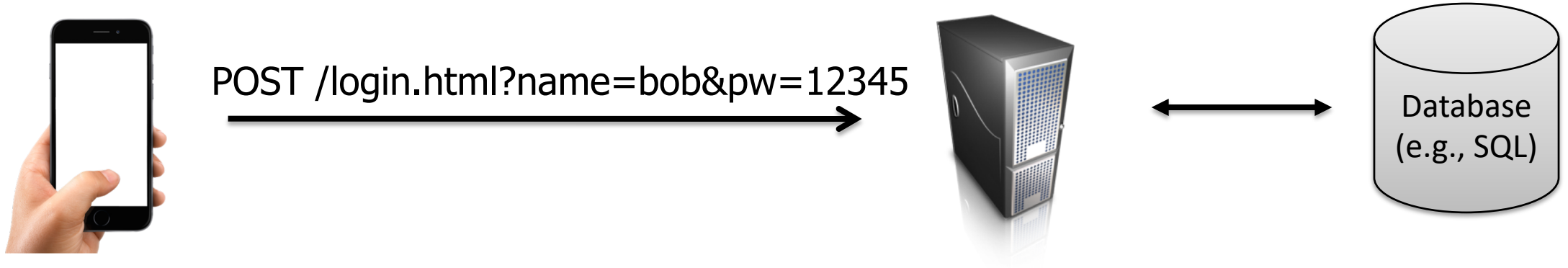
Database
(e.g., SQL)

Guests

airbnb

What threats does Airbnb need to worry about?

# Password management

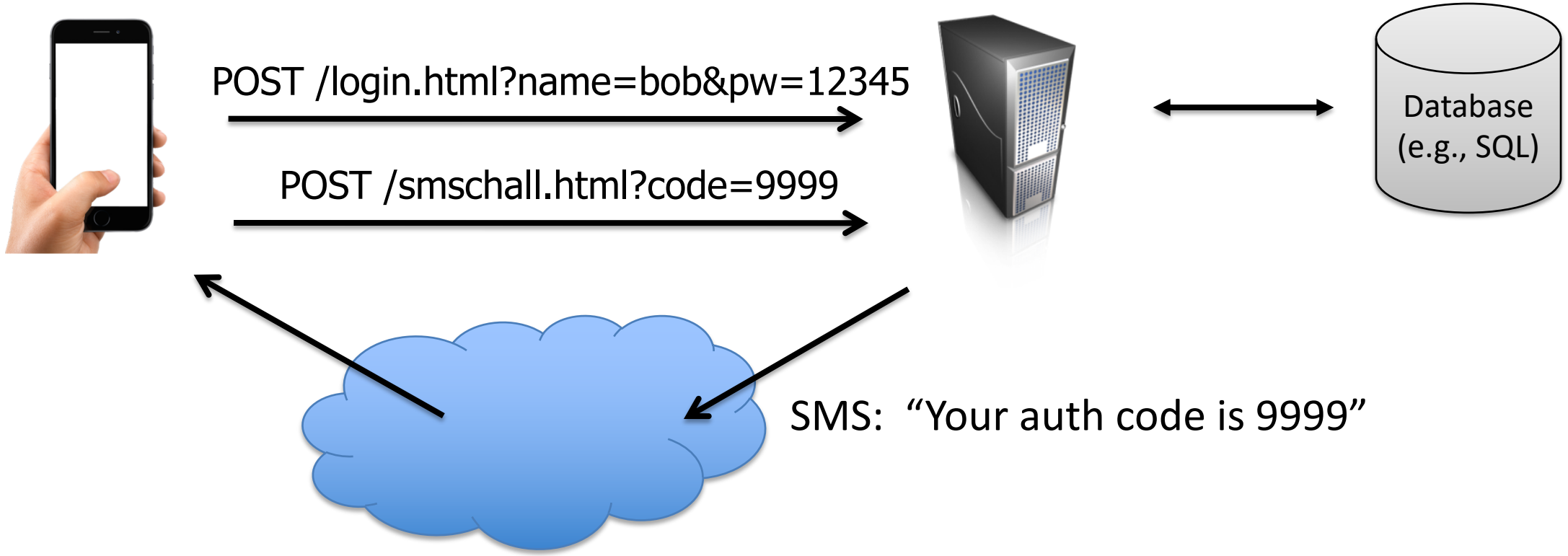



| Countermeasure | Purpose |
|---|---|
| Password hashing | Database leak doesn't immediately reveal user passwords. Slows ***offline guessing attacks*** |
| Strength meters | Nudge / force users to pick stronger passwords to mitigate guessing attacks |
| Lockout after X failed attempts | Prevent remote guessing attacks (X typically 10, 100, 1000); slows down / prevents ***online guessing attacks*** |
| Compromised credential checks | Check if password is in known breaches |

# Second factor authentication (2FA)

- Combine passwords with another way to authenticate user
- Second factor usually proof of ownership of:
  - Email address
  - Telephone number (via SMS)
  - Device (via authenticator app)
  - Hardware token (One-time-password token, universal second factor U2F token)

# Second factor authentication (2FA)




Suppose you know someone's password (e.g., due to breach) but their account is protected by SMS-based 2FA. ***What can you do as an attacker?***

# Circumventing SMS-based 2FA

Suppose you know someone's password (e.g., due to breach) but their account is protected by SMS-based 2FA

- Have physical access to device that receives SMS
- Phishing attacks – confuse user into disclosing SMS to you
- SIM swap – trick phone company into registering victim's phone # to your device
- SMS hijacking: exploit vulnerabilities in cellular network (called SS7)
  https://berlin.ccc.de/~tobias/31c3-ss7-locate-track-manipulate.pdf

[Doerfler et al. 2019]: SMS 2FA circumvented in ~4% of phishing attacks
~26% of targeted attacks

Best practice: use authenticator app and/or hardware token

# Over 90 percent of Gmail users still don't use two-factor authentication

*The security tool adds another layer of security if your password has been stolen*

By Thuy Ong | @ThuyOng | Jan 23, 2018, 8:30am EST

Usability remains a key issue preventing adoption

# Other authentication signals

- Location-based authentication
  - IP-based geolocation
- Device identification
  - Cookies, device fingerprinting
- Behaviorial cues
  - Typical actions on platform (even after authenticated)
- Biometrics
  - Fingerprints, etc.

# User authentication is huge pain

- Simple typos in passwords cause 3% of Dropbox users to be unable to login in 24 hour period [Chatterjee et al. 2016]

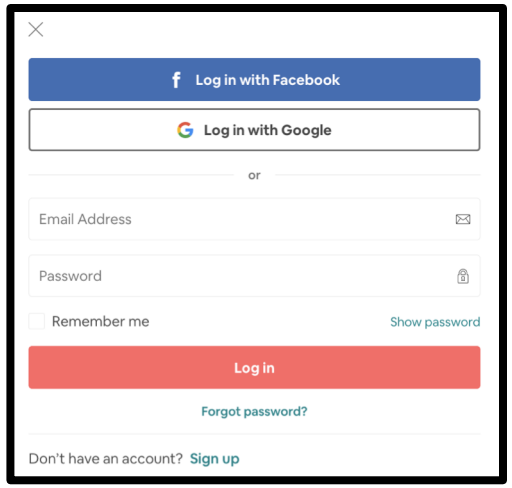
  https://www.cs.cornell.edu/~rahul/papers/pwtypos.pdf

- 52% of users fail login challenges at Google, 3% don't get in within short period of time [Doerfler et al. 2019]

  https://ai.google/research/pubs/pub48119

- How to minimize friction for honest users?

# Session handling and login




GET /login.html

Set-Cookie: AnonSessID=134fds1431

POST /login.html?name=bob&pw=12345

Cookie: AnonSessID=134fds1431

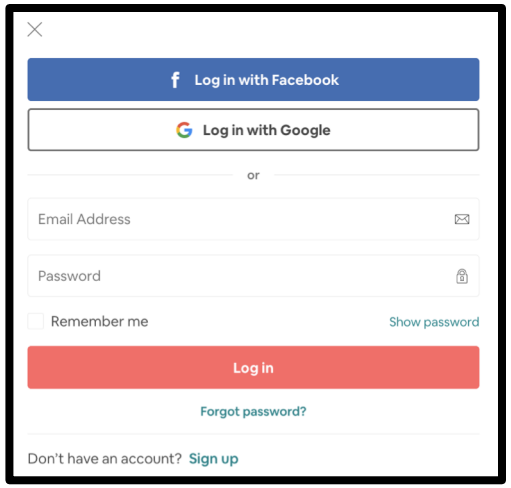
Set-Cookie: SessID=83431Adf

GET /account.html

Cookie: SessID=83431Adf

Need to check password matches registered version

# Cookies: Setting/Deleting




GET ...

HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;

path = (when to send)

secure = (only send over SSL);

expires = (when expires) ;

HttpOnly

scope

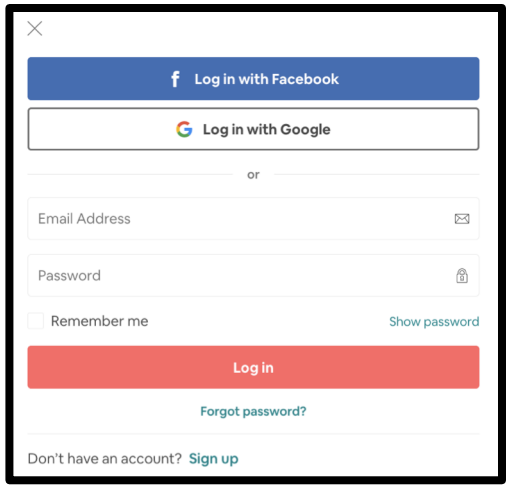
if expires=NULL:
this session only

- Delete cookie by setting “expires” to date in past
- Default scope is domain and path of setting URL
- Client can also set cookies (Javascript)

# Cookie scope rules (domain and path)

- Say we are at tech.cornell.edu
  - Any non-top level domain (non-TLD) suffix can be scope:
    - allowed: tech.cornell.edu or cornell.edu
    - disallowed: www.cornell.edu or wisc.edu
- Path can be set to anything
- Default is domain and path of request

# Cookies: reading by server

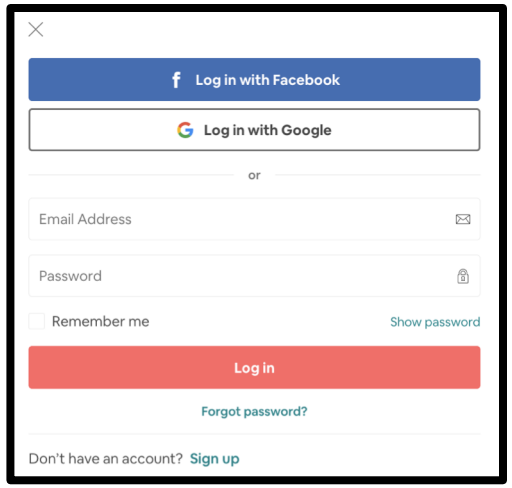



GET /url-domain/url-path

Cookie: name=value

- Browser sends all cookies such that
  - domain scope is suffix of url-domain
  - path is prefix of url-path
  - protocol is HTTPS if cookie marked “secure”

# Session handling and login




GET /login.html

Set-Cookie: AnonSessID=134fds1431

POST /login.html?name=bob&pw=12345

Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

Need to check password matches registered version

Cookie acts as ***bearer token***

GET /account.html

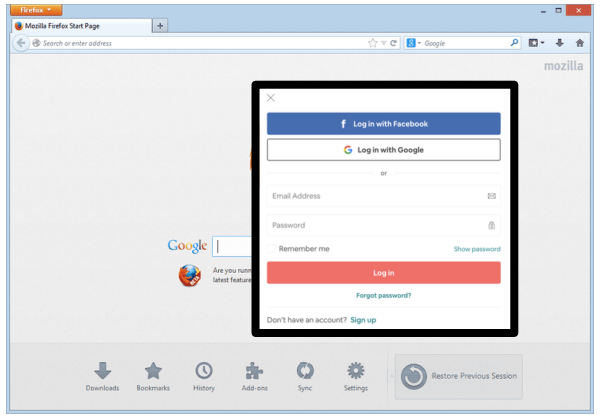
Cookie: SessID=83431Adf

# Authentication vs. authorization

- Authentication securely confirm identity
- Authorization gives (authenticated) party right to access something

# Single-sign on (SSO)

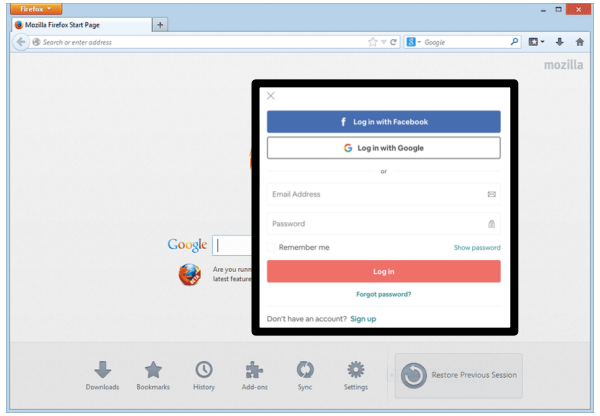
- ***Identity provider (IdP)*** handles authentication
  - (e.g., via username & password)
- Log into ***relying party (RP)*** via attestion by IdP
- Most cases: distinct web servers




Identity provider (IdP)
(e.g.:
identity.cornell.edu)

Relying party (RP)
(e.g.: account.html?user=tom)

# How might SSO work?



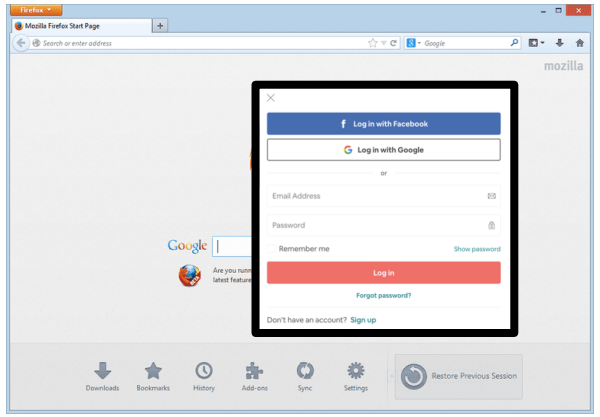

Identity provider (IdP)
(e.g.:
identity.cornell.edu)

Relying party (RP)
(e.g.: account.html?user=tom)

In-class 5-minute exercise:
How would you securely authorize browser to access www.cornell.edu/account.html?user=tom based on authentication to identity.cornell.edu?

# Single-sign on (SSO)

- Many standards for solving these problems:
  - SAML, OpenID Connect + OAuth 2.0, …
- Involve combination of logical flows plus some cryptography




Give me permissions for account.html as Tom

Auth token

IdP (e.g.: Google, Facebook, identity.cornell.edu)

Give me account.html?user=tom and here's token

Tom's account page

RP

Need to check validity of token!

# OAuth 2.0

- Widely used authorization protocol standard
- Used for web apps and mobile apps
- Two of the authorization grant types:

**Authorization code**
Used for server-side applications to obtain access to resources

**Implicit**
Used for client-side applications, such as client-side javascript or mobile app

https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

# OAuth 2.0  implicit flows

Request triggers redirect to authorization request

https://authserv.com/authorize?
response_type=code &
client_id=CLIENT_ID &
redirect_uri=https://myapp.com/callback &
scope=read

Tutorial:  https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

# OAuth 2.0  implicit flows

Request triggers redirect to authorization request

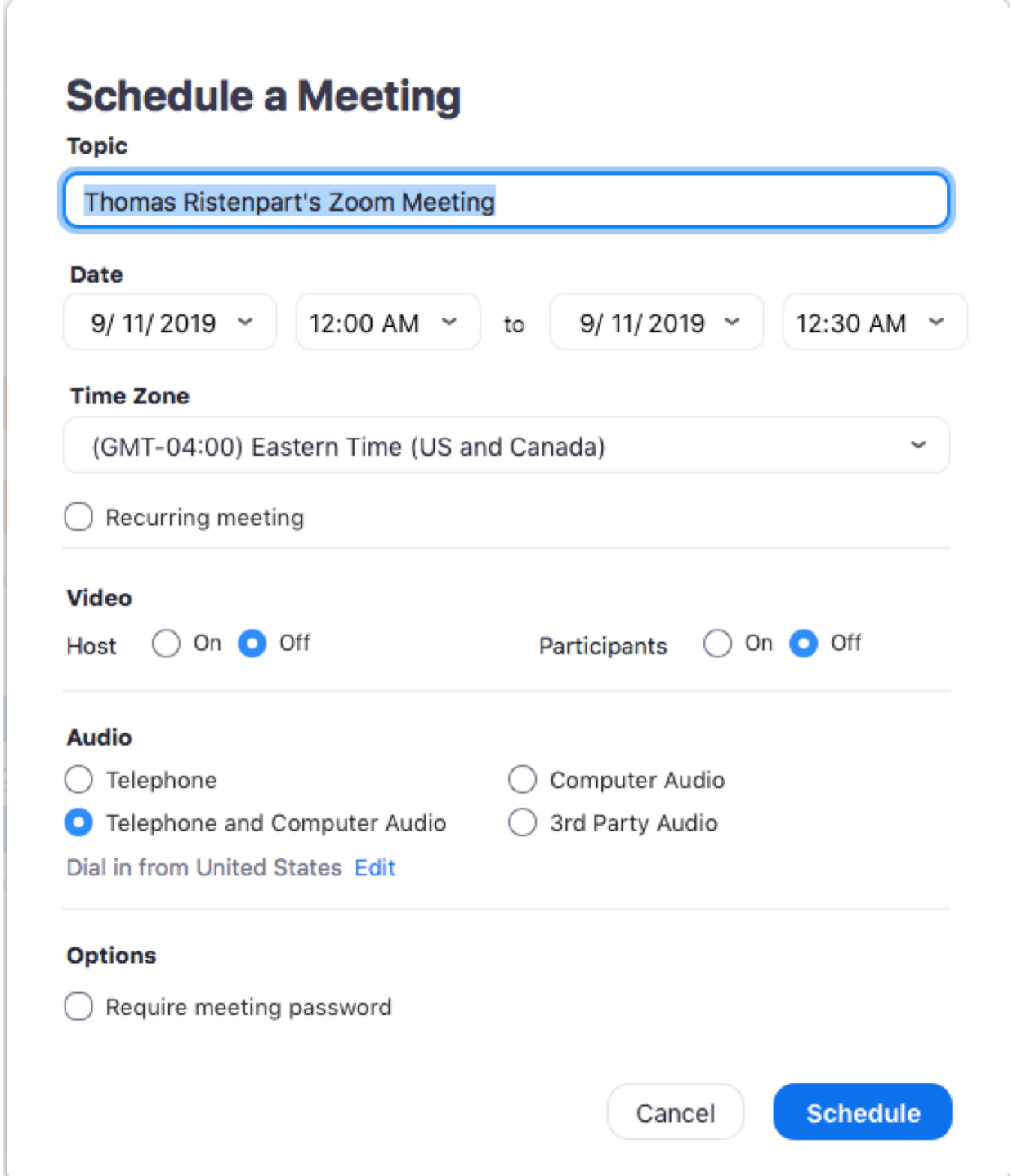
https://authserv.com/authorize?
response_type=code &
client_id=CLIENT_ID &
redirect_uri=https://myapp.com/callback &
scope=read

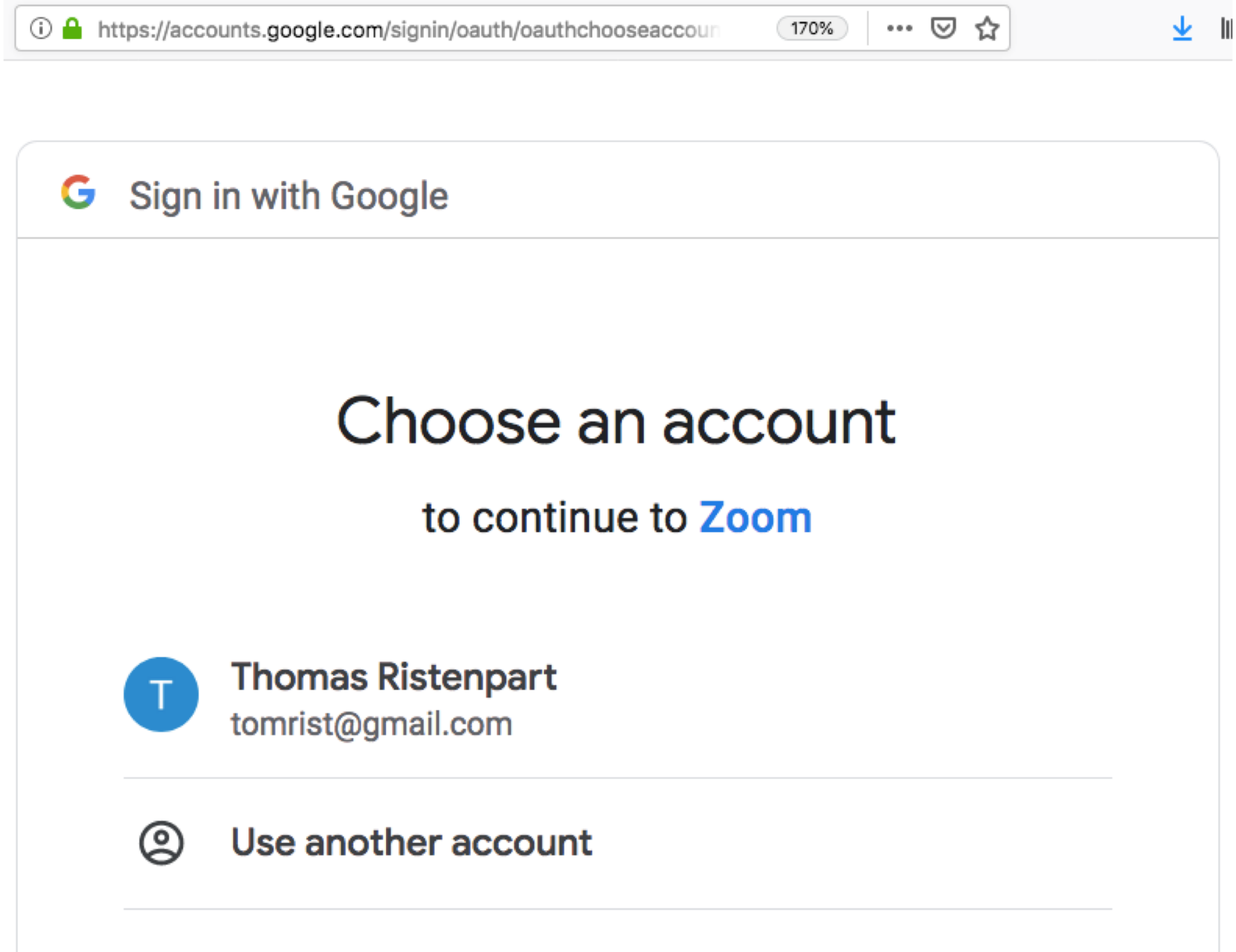
Redirects browser via redirect URI, with an auth token

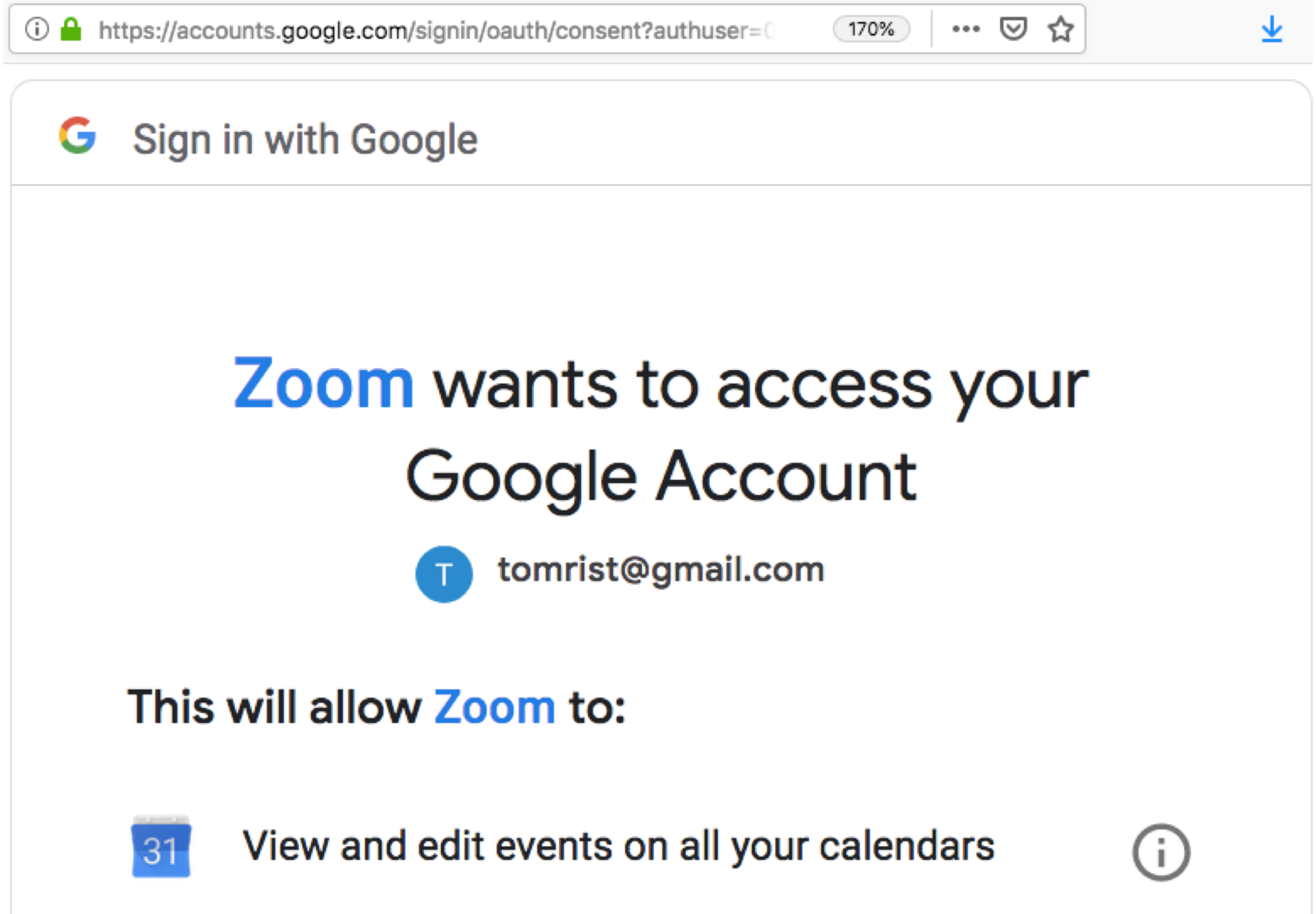
https://myapp.com/callback?token=ACCESS_TOKEN

Now client-side can use this token to access resource

Tutorial:  https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2


Schedule a Meeting
Topic
Thomas Ristenpart's Zoom Meeting
Date
9/ 11/ 2019
12:00 AM
to
9/ 11/ 2019
12:30 AM
Time Zone
(GMT-04:00) Eastern Time (US and Canada)
Recurring meeting
Video
Host
On
Off
Participants
On
Off
Audio
Telephone
Computer Audio
Telephone and Computer Audio
3rd Party Audio
Dial in from United States Edit
Options
Require meeting password
Cancel
Schedule


https://accounts.google.com/signin/oauth/oauthchooseaccoun
170%
Sign in with Google
Choose an account
to continue to Zoom
T
Thomas Ristenpart
tomrist@gmail.com
Use another account


https://accounts.google.com/signin/oauth/consent?authuser=
170%
Sign in with Google
Zoom wants to access your Google Account
T
tomrist@gmail.com
This will allow Zoom to:
31
View and edit events on all your calendars

# What can go wrong?

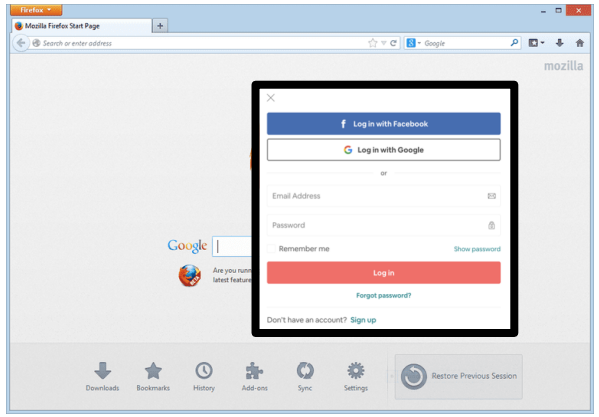
- [Wang, Chen, Wang 2012] built tool to explore OAuth security
- Found ***lots*** of vulnerabilities:

> *The web SSO systems we found to be vulnerable include those of Facebook, Google ID, PayPal Access, Freelancer, JanRain, Sears and FarmVille. All the discovered flaws allow unauthorized parties to log into victim user's accounts on the RP, as shown by the videos in [33]*

- Eg: Google's SSO could be told not to bind auth token to user's identity, and the relying party could be tricked into giving access to arbitrary victim account

https://www.microsoft.com/en-us/research/wp-content/uploads/2012/05/websso-final.pdf

# Vulnerability diagrammatically




Tweak parameters to tell Google not to bind token to "Tom"

Give me permissions for account.html as Tom

Auth token

IdP (e.g.: Google, Facebook, login.cornell.edu)

alice

Give me account.html?user=tom and here's token

Alice's account page

RP

# What can go wrong?

- [Wang, Chen, Wang 2012] built tool to explore OAuth security
- Found ***lots*** of vulnerabilities:

> *The web SSO systems we found to be vulnerable include those of Facebook, Google ID, PayPal Access, Freelancer, JanRain, Sears and FarmVille. All the discovered flaws allow unauthorized parties to log into victim user's accounts on the RP, as shown by the videos in [33]*

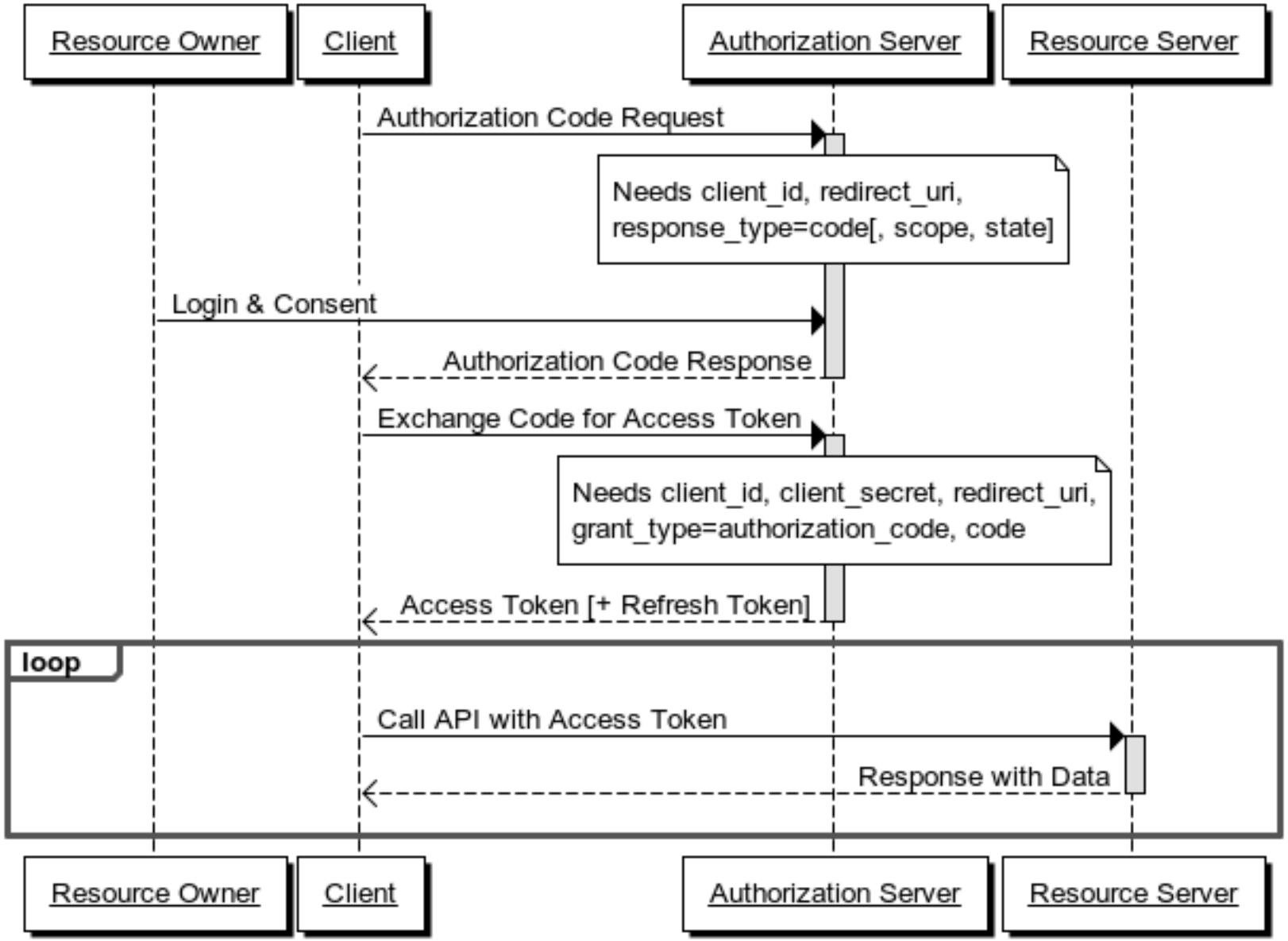
- These fixed, but logical bugs in authentication & authorization rampant in web apps
  - CSRF, XSS bugs (stay tuned) often target authorization logic

https://www.microsoft.com/en-us/research/wp-content/uploads/2012/05/websso-final.pdf

# Next time

- Abuse of (authenticated/authorized) accounts
  - Commercial abuse
- Then we'll move on to interpersonal abuse

# Authorization Code Grant Flow

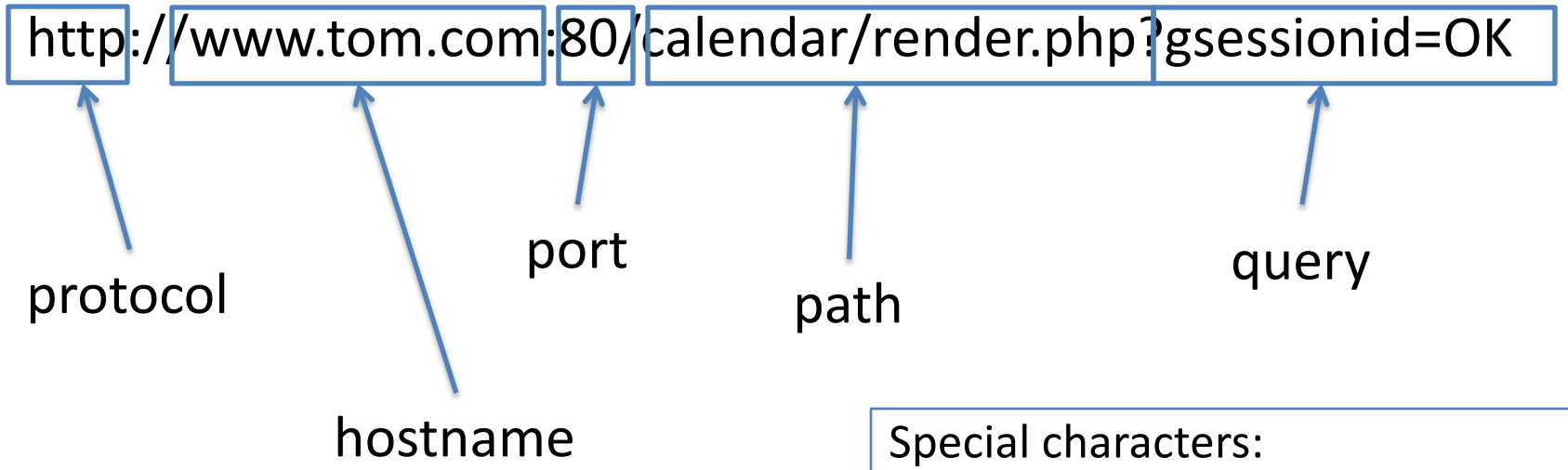



www.websequencediagrams.com

```html
<html lang="en">
  <head>
    <meta name="google-signin-scope" content="profile email">
    <meta name="google-signin-client_id" content="YOUR_CLIENT_ID.apps.googleusercontent.com">
    <script src="https://apis.google.com/js/platform.js" async defer></script>
  </head>
  <body>
    <div class="g-signin2" data-onsuccess="onSignIn" data-theme="dark"></div>
    <script>
      function onSignIn(googleUser) {
        // Useful data for your client-side scripts:
        var profile = googleUser.getBasicProfile();
        console.log("ID: " + profile.getId()); // Don't send this directly to your server!
        console.log('Full Name: ' + profile.getName());
        console.log('Given Name: ' + profile.getGivenName());
        console.log('Family Name: ' + profile.getFamilyName());
        console.log("Image URL: " + profile.getImageUrl());
        console.log("Email: " + profile.getEmail());

        // The ID token you need to pass to your backend:
        var id_token = googleUser.getAuthResponse().id_token;
        console.log("ID Token: " + id_token);
      }
    </script>
  </body>
</html>
```

# Uniform resource locators (URLs)

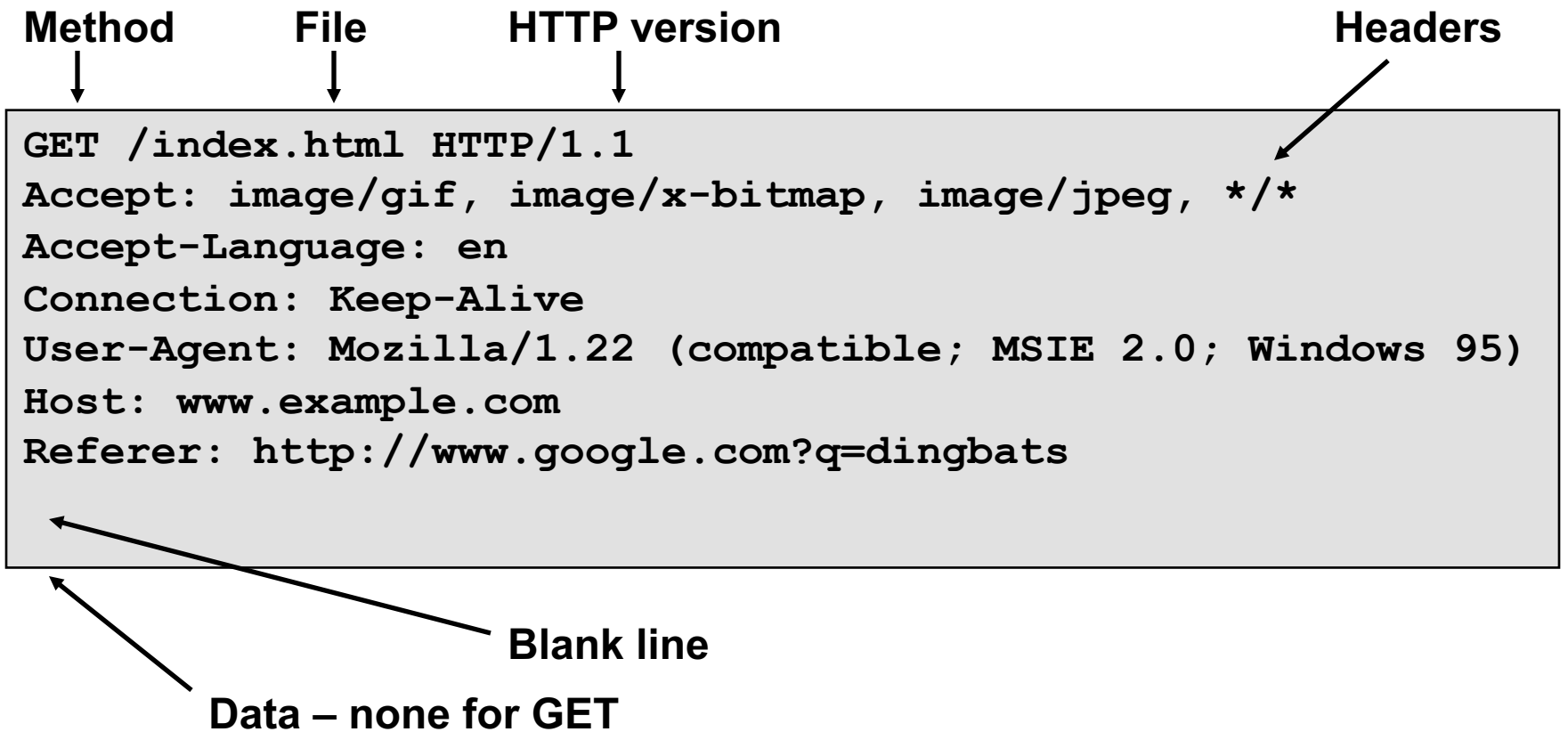



URL's only allow ASCII-US characters.
Encode other characters:

%0A = newline
%20 = space

Special characters:
+ = space
? = separates URL from parameters
% = special characters
/ = divides directories, subdirectories
# = bookmark
& = separator between parameters

# HTTP Request




| GET : no side effect | POST : possible side effect |
|---|---|

# HTTP Response