Security Issues in Web Programming (Part 4)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - Some important web pgmming security risks
  - Some mitigation techniques

- Thanks to **Joseph Eichenhofer** (COS 333 TA during the Spring 2020 semester) for major contributions to this lecture

Agenda

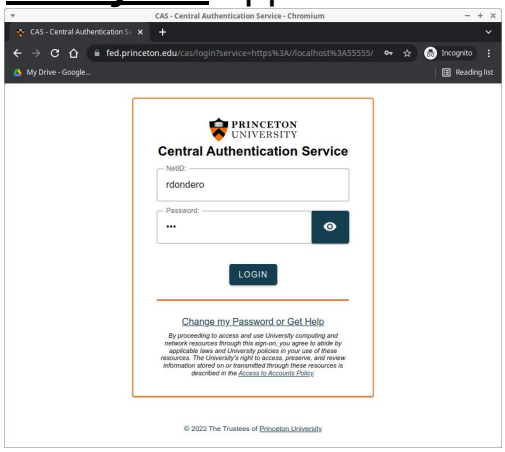
- **Broken authentication: Central Authentication Service (CAS)**
- Broken authentication: Google authentication

CAS

- ***Central Authentication Service (CAS)***:
  - Wikipedia:
    - "The Central Authentication Service (CAS) is a single sign-on protocol for the web."
    - "Its purpose is to permit a user to access multiple applications while providing their credentials (such as userid and password) only once."
    - "It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password."

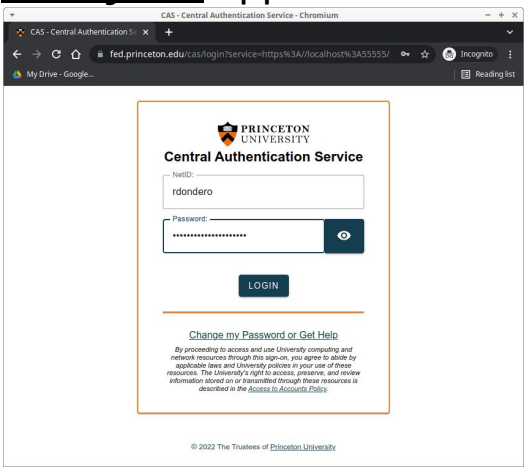
CAS: Example

- See **PennyCas** app



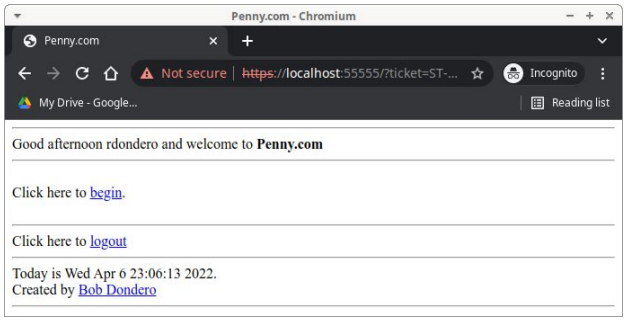
CAS: Example

- See **PennyCas** app



CAS: Example

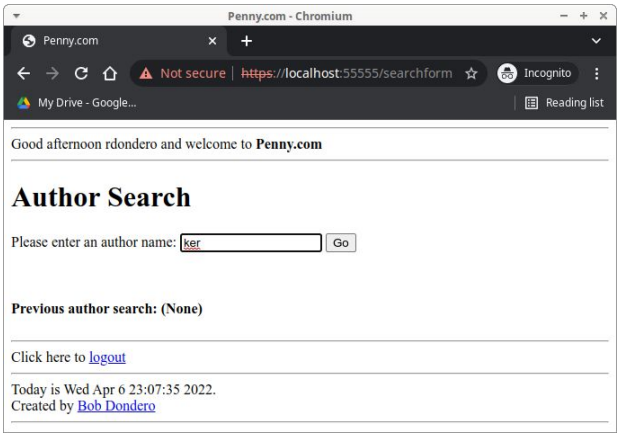
- See **PennyCas** app (cont.)



Note username in header
Note logout link in footer

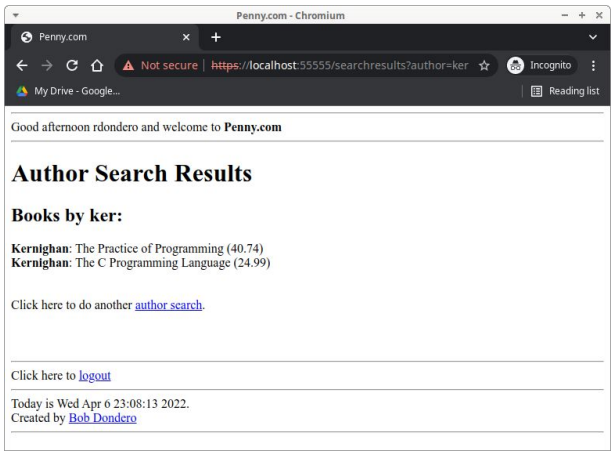
CAS: Example

- See **PennyCas** app (cont.)



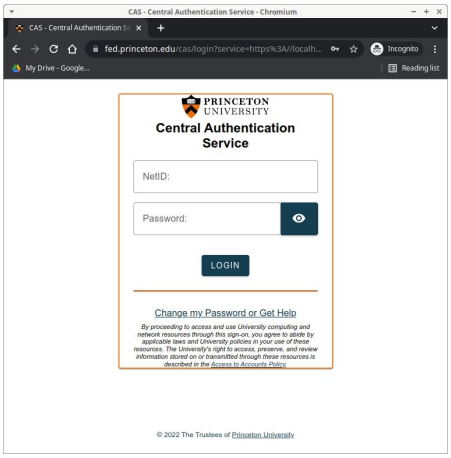
CAS: Example

- See **PennyCas** app (cont.)



CAS: Example

- See **PennyCas** app (cont.)



CAS: Flow

- See document: **CAS Flow**

CAS: Code

- See **PennyCas** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - header.html, footer.html, index.html, searchform.html, searchresults.html
  - keys.py
  - **auth.py**
  - **penny.py**

CAS: Assessment

- **Pros**
  - Application need not manage usernames or passwords
  - Application ***cannot*** access passwords!
  - Application is constrained to one user community

CAS: Assessment

- **Cons**
  - Complex
  - Adds overhead, but mostly only during first visit per browser session
  - Application is constrained to one user community!

Agenda

- Broken authentication: Central Authentication Service (CAS)
- **Broken authentication: Google authentication**

Google Auth

- ***Google authentication***
  - Delegate your app's authentication to Google

Google Auth

- The general idea…

> "You're about to write a third-party application, and it will let a user use a Google Login button to log in. To do that, Google needs to know about your application. Luckily, you can r**egister your application as a client to Google.**
>
> Once a user comes to your application and presses the Google Login button, you can send them to Google. From there, **Google needs to make sure that the user consents to pass along their email and other information to your application**. Should the user consent, **Google sends back some information to your application**. You then **store that information and can reference it later, effectively logging the user in.**"
>
> – https://realpython.com/flask-google-login/

Google Auth: Example

- Example: **PennyGoogleAuth** app...

Google Auth: Example

- Preliminary step
  - Register app with Google
    - Browse to:
      - https://console.developers.google.com/apis/credentials
    - Choose *Credentials*, *Create Credentials*, *OAuth client id*, *Web Application*
    - Add URI:
      - https://localhost:5000
      - Could add another (for Heroku)
    - Add Authorized Redirect URI:
      - https://localhost:5000/login/callback
      - Could add another (for Heroku)
    - Google provides client id and client secret key
      - Take note of them!

Google Auth: Example

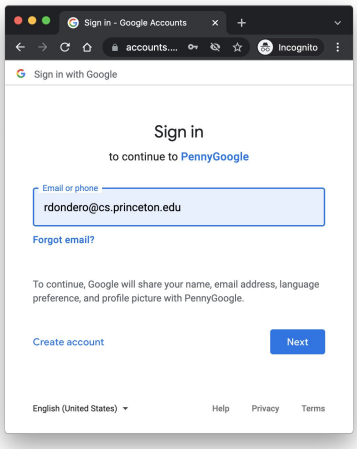
- Preliminary step
  - Install required Python modules
    - `python -m pip install requests`
    - `python -m pip install oauthlib`

Google Auth: Example

- See **PennyGoogleAuth** app
  - To run:
    - Issue command: `python runserver.py`
      - Port is hardcoded to 5000
      - Runs Flask test server with HTTPS support
    - In browser, enter URL:
      - https://localhost:5000

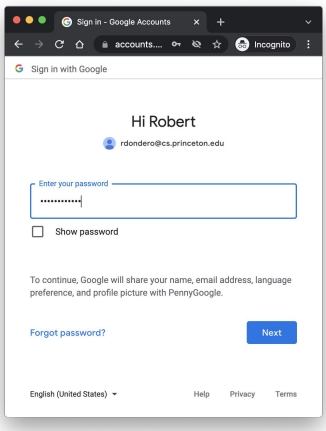
Google Auth: Example

- See **PennyGoogleAuth** app (cont.)



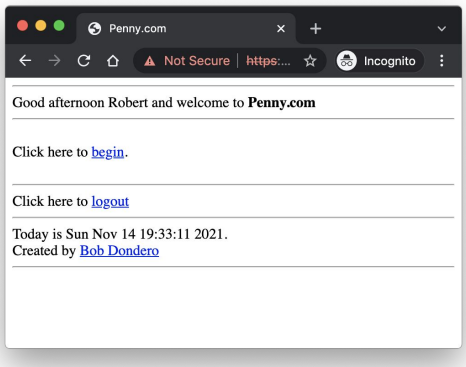
Google Auth: Example

- See **PennyGoogleAuth** app (cont.)



Google Auth: Example

- See **PennyGoogleAuth** app (cont.)



Google Auth: Flow

- See document: **Google Authentication Flow**

Google Auth: Code

- See **PennyGoogleAuth** app (cont.)
  - **runserver.py**
  - penny.sql, penny.sqlite
  - book.py, database.py
  - header.html, footer.html, index.html, searchform.html, searchresults.html
  - **keys.py**
  - **auth.py**
  - **penny.py**

Google Auth

- **See also:**
  - **Client-side** Google authentication
    - https://developers.google.com/identity/gsi/web/guides/overview

Google Auth: Assessment

- **Pros**
  - Users need not remember yet another password
  - Application need not manage usernames or passwords
    - Application need not implement related functionality (forgotten password, password reset)
  - Application ***cannot*** access passwords
  - Application can access information that user provided to Google
    - Email address, photo, …

Google Auth: Assessment

- **Cons**
  - Complex
  - Adds overhead, but mostly only during first visit per browser session
  - Application is constrained to users who have Google accounts

More Information

- For more information...

- **CAS**
  - https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol.html
- **Google authentication**
  - https://realpython.com/flask-google-login/

Summary

- We have covered:
  - Broken authentication: Central Authentication Service (CAS)
  - Broken authentication: Google authentication

Summary

- We have covered:
  - XML external entities (XXE)
  - Insufficient logging & monitoring
  - Security misconfiguration
  - Using components with known vulnerabilities
  - Broken access control
  - Injection
  - Insecure deserialization
  - Sensitive data exposure
  - Cross-site scripting (XSS)
  - Broken authentication