

Security Issues in Web Programming (Part 3)

Copyright © 2025 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
 - Some web programming security attacks
 - Some ways to thwart them

Quick Review

- Recall **PennyAdmin04Auth** app
 - Username cookie exists => user is authenticated

Agenda

- **Cookie forgery attacks**
- Cookie privacy attacks
- Cross-site request forgery (CSRF) attacks

Cookie Forgery Attacks

- **Problem:**
 - In PennyAdmin app, an attacker can forge the username cookie

Cookie Forgery Attacks

- Recall **PennyAdmin04Auth** app
 - Example:
 - Attacker runs **cookieforgeryattack.py**
 - Sends forged username cookie to PennyAdmin app

Cookie Forgery Attacks

- Recall PennyAdmin04Auth app
 - Example (cont.):

```
$ python cookieforgeryattack.py localhost 55555
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.3
Date: Sat, 31 Aug 2024 18:56:30 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 780
Connection: close

<!DOCTYPE html>
<html>
  <head>
    <title>PennyAdmin</title>
  </head>
  <body>
    <hr>Hello rdondero, and welcome to <strong>PennyAdmin</strong><hr>
    <h1>Show All Books</h1>
```

App
considers
user
to be
logged
in

Continued on next slide

Cookie Forgery Attacks

- Recall **PennyAdmin04Auth** app
 - Example (cont.):

```
123:
<strong>Kernighan</strong>:
The Practice of Programming<br>

234:
<strong>Kernighan</strong>:
The C Programming Language<br>

345:
<strong>Sedgewick</strong>:
Algorithms in C<br>

<br>
<a href="/index">Return to home page</a>
<br>
<hr>
<a href="logout">Log out</a> of the application</br>
<hr>
Created by <a href="https://www.cs.princeton.edu/~rdondero">
Bob Dondero</a>
<hr>
</body>
</html>
```

App
considers
user
to be
logged
in

Cookie Forgery Attacks

- **Solution 1:**

- ***Cookie encryption***

- Before server sends username cookie...
 - Server uses a ***secret key*** to **encrypt** the value of the username cookie
 - After server receives username cookie...
 - Server uses the same ***secret key*** to **decrypt** the value of the username cookie
 - Decryption fails => forgery

Aside: Secret Keys

- **Question:** How to generate a secret key?
- **One answer:**

```
$ python
Python 3.12.3 (main, Jul 31 2024, 17:43:48)
[GCC 13.2.0] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import os
>>> os.urandom(12).hex()
'████████████████████'
>>> quit()
$
```

Use
this
as
secret
key



Aside: Secret Keys

- **Question:** Where to store secret keys?
- **Possible answers:**
 - Source code files? **No**
 - Attacker might gain access to GitHub repo
 - Some other file? **Maybe**
 - But must make sure the file is not in GitHub repo
 - Environment variables? **Yes**
 - The common way

Aside: Secret Keys

To run subsequent versions of PennyAdmin:

Mac & Linux:

```
$ export APP_SECRET_KEY=yourappsecretkey  
$ python runserver.py 55555
```

MS Windows:

```
$ set APP_SECRET_KEY=yourappsecretkey  
$ python runserver.py 55555
```

Or use the `python_dotenv` package

```
$ cat .env  
APP_SECRET_KEY=yourappsecretkey  
$
```

Cookie Forgery Attacks

- See **PennyAdmin05aEncrypt** app
 - runserver.py
 - penny.sql, penny.sqlite
 - database.py
 - header.html, footer.html
 - index.html, show.html,
 - add.html, delete.html, reportresults.html
 - login.html, signup.html, loggedout.html
 - top.py, penny.py, **auth.py**

Cookie Forgery Attacks

- See **PennyAdmin05aEncrypt** app
 - Example:
 - Attacker runs **cookieforgeryattack.py**
 - Sends forged username cookie to PennyAdmin app

Cookie Forgery Attacks

- See **PennyAdmin05aEncrypt** app
 - Example (cont.):

```
$ python cookieforgeryattack.py localhost 55555
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.3 Python/3.12.3
Date: Sat, 31 Aug 2024 19:00:42 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 199
Location: /login
Set-Cookie: original_url=http://localhost/show; Path=/
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL:
<a href="/login">/login</a>. If not, click the link.
$
```

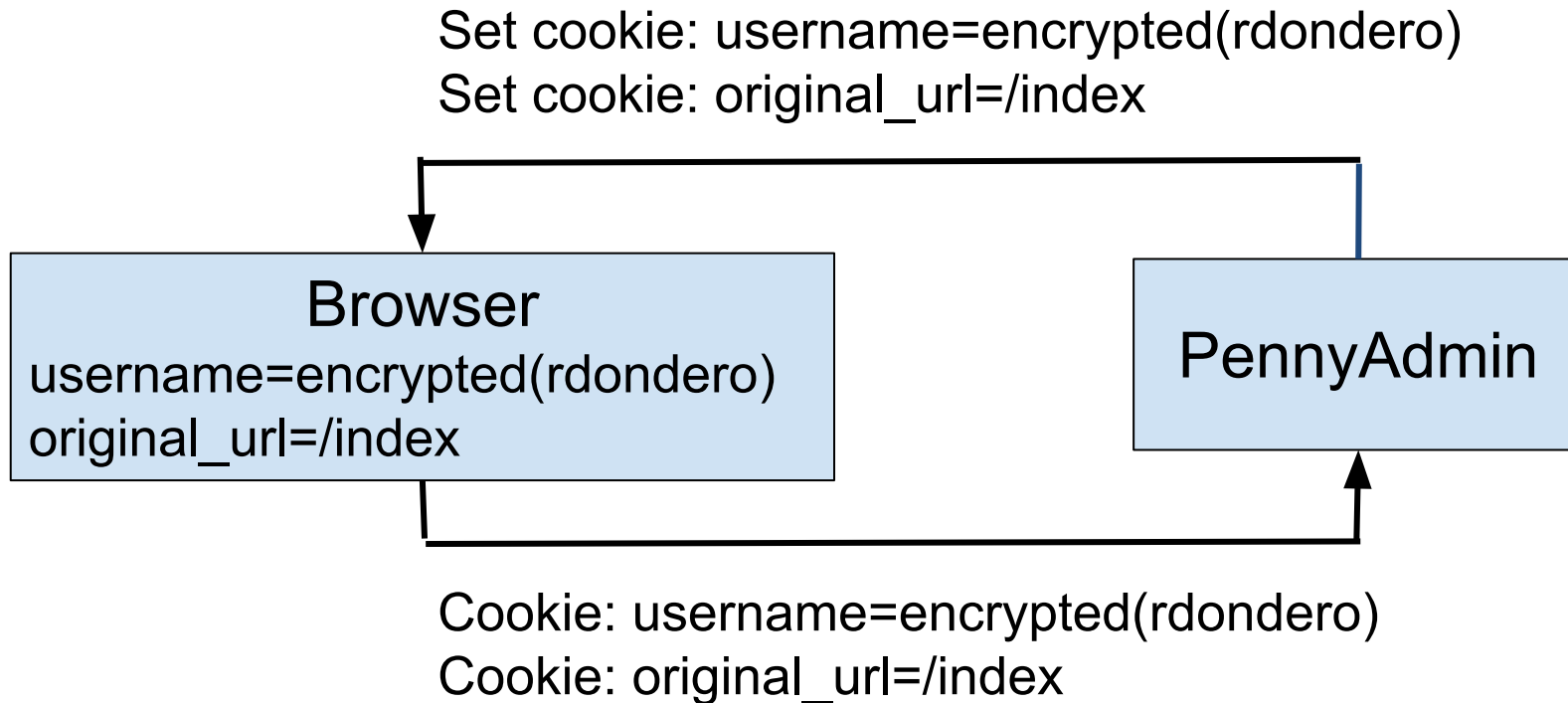
App
rejects
username,
redirects
to
login
page

Cookie Forgery Attacks

- **Solution 2:**
 - *Sessions*

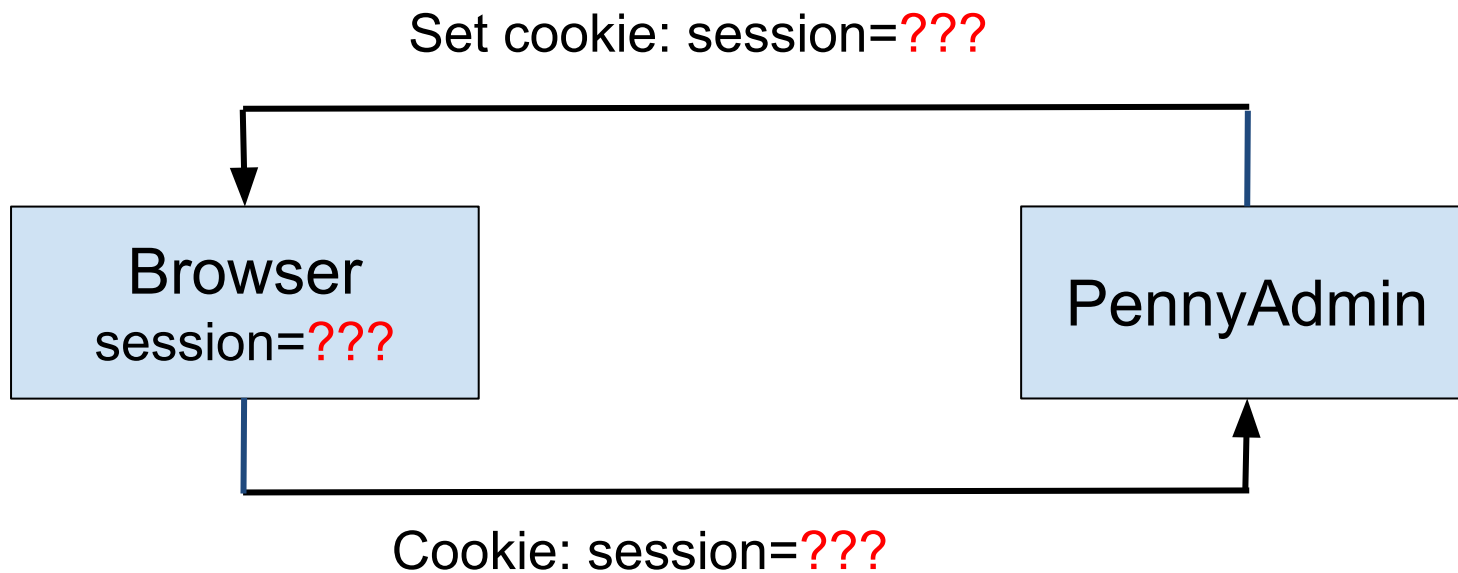
Cookie Forgery Attacks

Without sessions:



Cookie Forgery Attacks

With sessions:



eyJvcmlnaW5hbF91cmwiOiJodHRwOi8vbG9jYWxob3N0OjU1NTU1LyIsInVzZ
XJuYW1lIjoicmRvbmlcm8ifQ.YsDrnQ.fc45qAmc0Vk6pAr32bQnogTtx1c

Using my secret key, decrypts to:

original_url=/index; username=rdontero;

Cookie Forgery Attacks

- See **PennyAdmin05bSession** app
 - runserver.py
 - penny.sql, penny.sqlite
 - database.py
 - header.html, footer.html
 - index.html, show.html,
 - add.html, delete.html, reportresults.html
 - login.html, signup.html, loggedout.html
 - top.py, **penny.py**, **auth.py**

Cookie Forgery Attacks

- See **PennyAdmin05bSession** app
 - Example:
 - Attacker runs **cookieforgeryattack.py**
 - Sends forged session cookie to PennyAdmin app

Cookie Forgery Attacks

- See **PennyAdmin05bSession** app
 - Example (cont.):

```
$ python cookieforgeryattack.py localhost 55555
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.3 Python/3.12.3
Date: Sat, 31 Aug 2024 19:03:41 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 199
Location: /login
Vary: Cookie
Set-Cookie:
session=eyJvcmlnaW5hbF91cmwiOiJodHRwOi8vbG9jYWxob3N0L3Nob3cif
Q.ZtNpDQ.24_ouOA3YNt2oeAz9gEiz_sGZf0; HttpOnly; Path=/
Connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL:
<a href="/login">/login</a>. If not, click the link.
```

App
rejects
username,
redirects
to
login
page

Cookie Forgery Attacks

- Q: Project concern?
- A: **Yes!!!**
 - Iff your project app stores, in cookies, data that must not be forged

Agenda

- Cookie forgery attacks
- **Cookie privacy attacks**
- Cross-site request forgery (CSRF) attacks

Cookie Privacy Attacks

- **Observation 1:** Some apps store private data in sessions
 - Examples: Apps that use CAS, EntraID, Google, Auth0

Cookie Privacy Attacks

- **Problem 1:** Flask (client-side) session encryption thwarts forgery/tampering, but not viewing!!!
 - See <https://blog.miguelgrinberg.com/post/how-secure-is-the-flask-user-session>

Cookie Privacy Attacks

Example: In **PennyAdmin05bSession** app
The value of the session cookie:

```
eyJvcmlnaW5hbF91cmwiOiJodHRwOi8vbG9jYWxob3N0OjU1NTU1LyIsInVzZXJuYW1lIjoicmRvbmlcm8ifQ.Ys  
DrnQ.fc45qAmc0Vk6pAr32bQnogTtx1c
```

Name/value pairs (base-64 encoded)

Expiration date (base-64 encoded)

Signature (generated using secret key)

Cookie Privacy Attacks

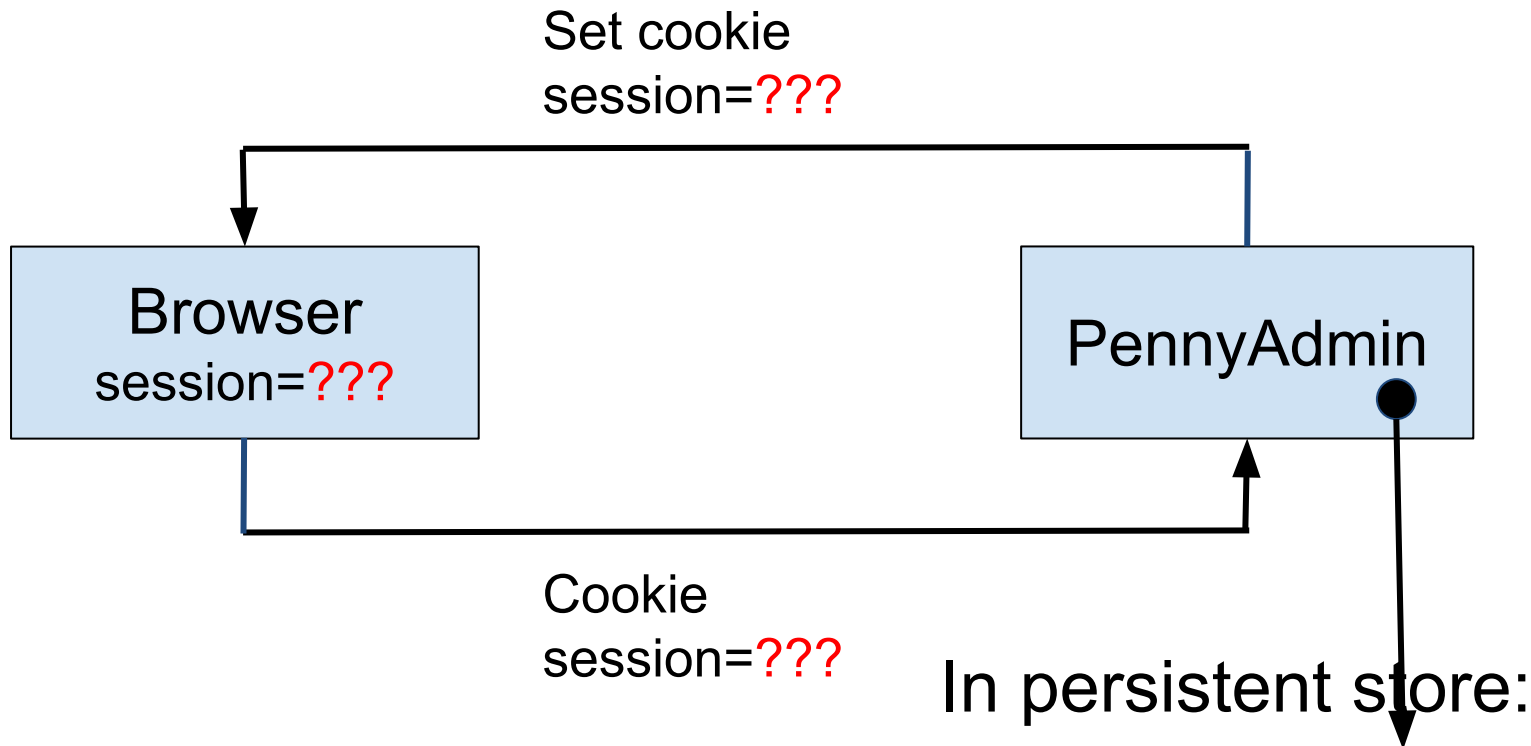
```
$ python
Python 3.10.5 (main, Jun 11 2022, 16:53:24) [GCC
9.4.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> import base64
>>>
base64.urlsafe_b64decode('eyJvcmlnaW5hbF91cmwiOiJod
HRwOi8vbG9jYWxob3N0OjU1NTU1LyIsInVzZXJuYW11IjoicmRv
bmRlcm8ifQ==')
b'{"original_url":"http://localhost:55555/", "userna
me":"rdondero"}'
>>> quit()
$
```

Cookie Privacy Attacks

- **Problem 2:** You really shouldn't store private data in (client-side) sessions anyway!
- **Solution:** Flask *server-side sessions*
 - Via the *flask_session* module

Cookie Privacy Attacks

Name=value pairs stored on server-side



ec7a9d68-5e2e-4f8d-
8d19-770e9aa848ac

SESSION_ID	SESSION_VALUE
???	original_url=http://localhost:55555 username=rdonero
...	...

Cookie Privacy Attacks

- See **PennyAdmin6SSSession** app:
 - runserver.py
 - penny.sql, penny.sqlite
 - book.py, database.py
 - header.html, footer.html
 - index.html, show.html,
 - add.html, delete.html, reportresults.html
 - login.html, loggedout.html
 - top.py, auth.py, **penny.py**

Cookie Privacy Attacks

- **Problem:**
 - Must maintain the persistent store
 - Routinely must delete old sessions from the persistent store
- **Solution:**
 - Routinely execute this command:
 - `flask --app penny.py session_cleanup`

Cookie Privacy Attacks

- Q: Project concern?
- A: **Yes!!!**
 - Iff your project app stores, in cookies, data that must be kept private

Agenda

- Cookie forgery attacks
- Cookie privacy attacks
- **Cross-site request forgery (CSRF) attacks**

CSRF Attacks

- *Cross-Site Request Forgery (CSRF)*

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

– <https://owasp.org/www-community/attacks/csrf>

CSRF Attacks

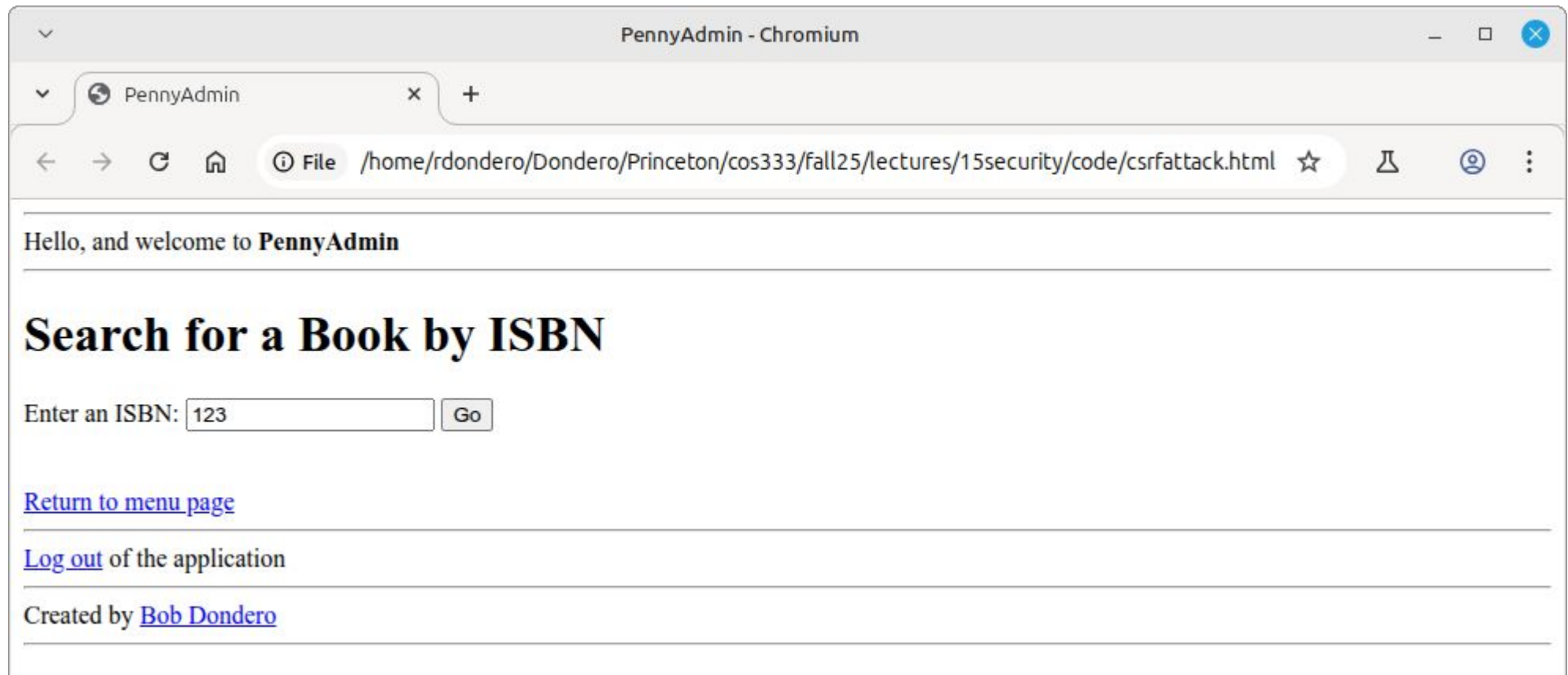
- **Problem:**
 - PennyAdmin is vulnerable to CSRF attacks

CSRF Attacks

- **Recall PennyAdmin06SSSession:**
 - Example:
 - Browser user visits app and logs in; browser session is authenticated/authorized
 - Attacker tricks user into visiting **csrfattack.html**
 - User submits form on csrfattack.html

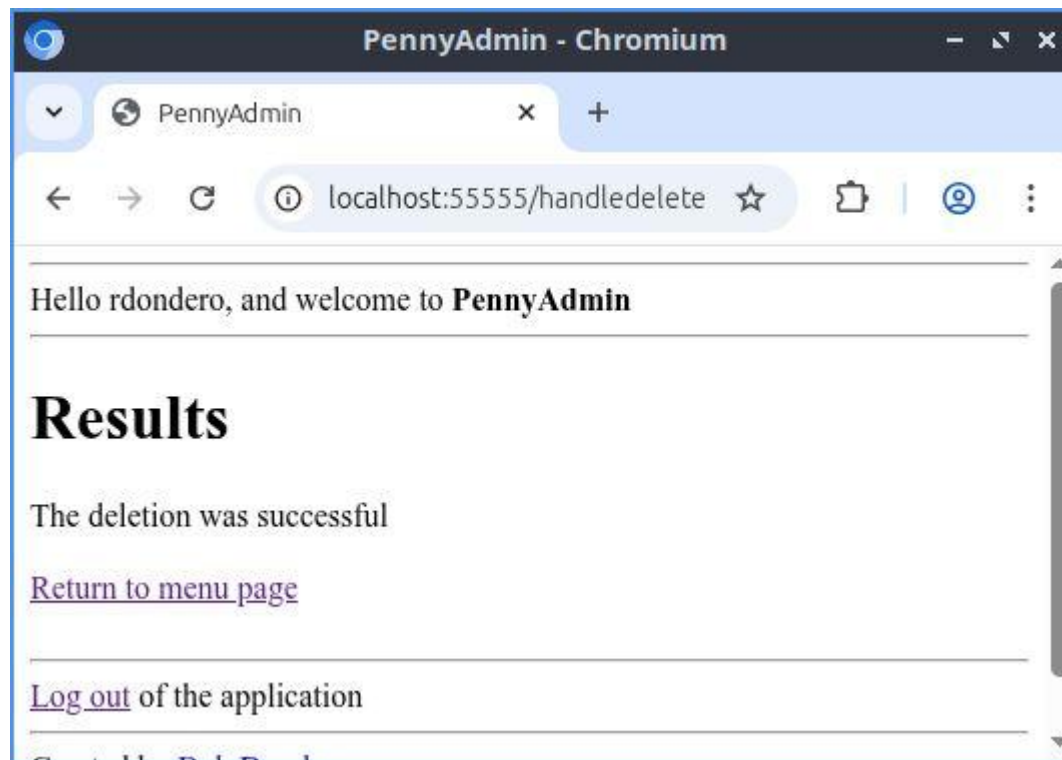
CSRF Attacks

- **Recall PennyAdmin06SSSession:**
 - Example:



CSRF Attacks

- **Recall PennyAdmin06SSSession:**
 - Example (cont.):



User
unintentionally
deletes
a
book!

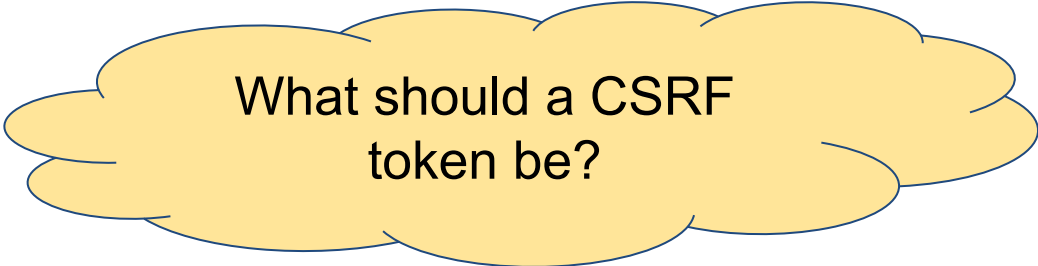
CSRF Attacks

- **Solution (in general):**
 - **PennyAuth app** must make sure that any POST request that it receives was sent from a page that **PennyAuth app** created

CSRF Attacks

- **Solution 1:**

- Use *CSRF tokens*
 - App creates a token
 - App places token in session
 - App requires any (POST) HTTP request to contain that token
 - Upon receipt of request, app makes sure that token in request equals token in session



What should a CSRF token be?

CSRF Attacks

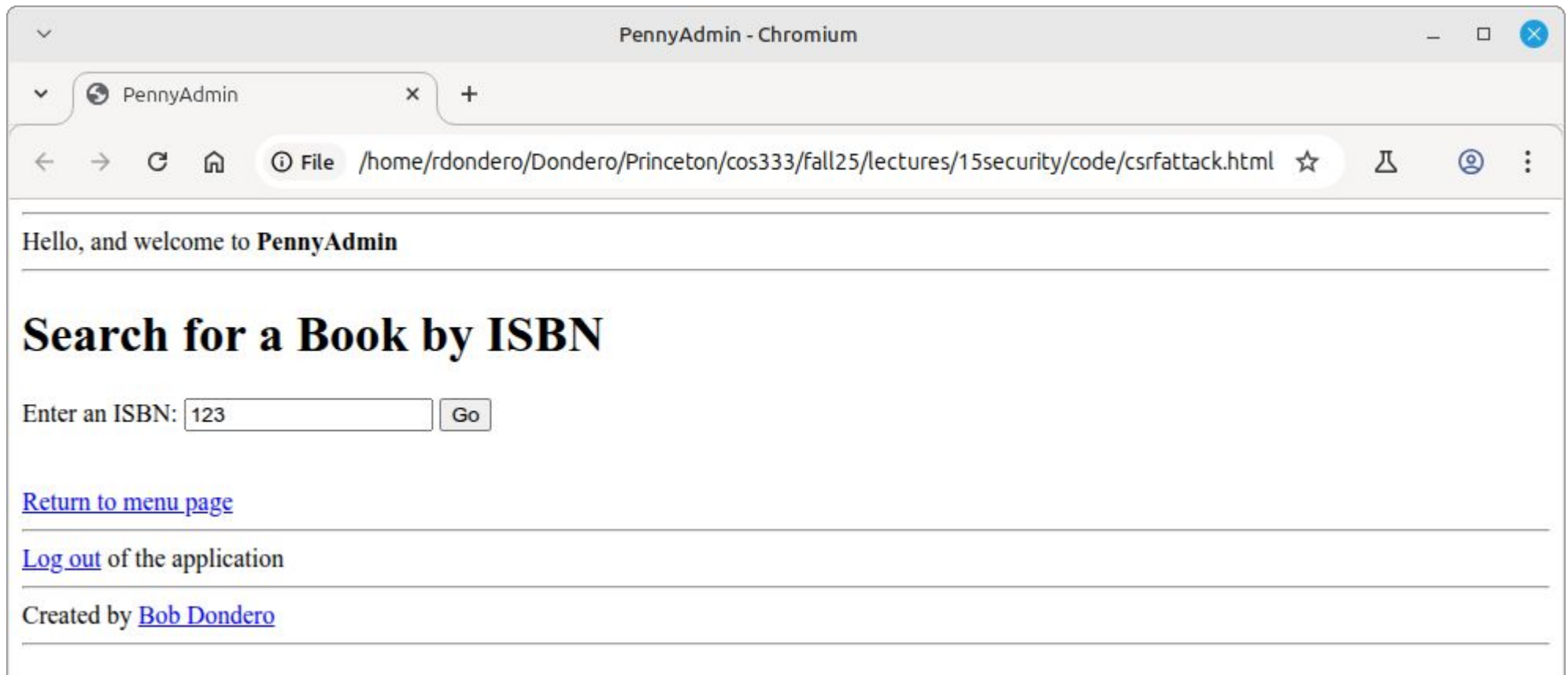
- **See PennyAdmin07aCsrfToken**
 - runserver.py
 - penny.sql, penny.sqlite
 - database.py
 - header.html, footer.html
 - index.html, show.html,
 - **add.html, delete.html, reportresults.html**
 - **login.html, signup.html, loggedout.html**
 - top.py, **penny.py, auth.py**

CSRF Attacks

- **See PennyAdmin07aCsrfToken**
 - Example:
 - Browser user visits app and logs in; browser session is authenticated/authorized
 - Attacker tricks user into visiting **csrfattack.html**
 - User submits form on csrfattack.html

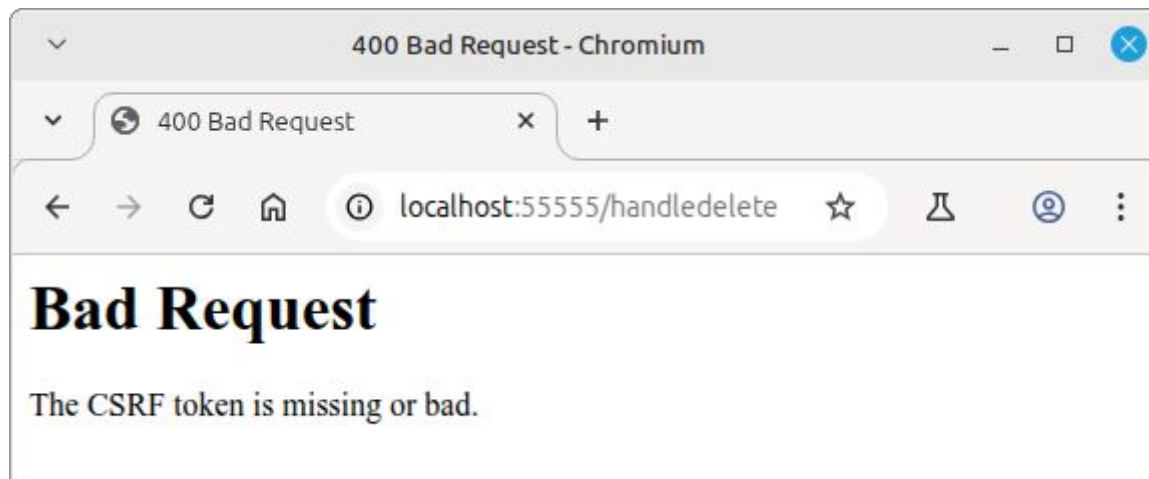
CSRF Attacks

- **See PennyAdmin07aCsrfToken**
 - Example:



CSRF Attacks

- **See PennyAdmin07aCsrfToken**
 - Example (cont.):



App
rejects
request

CSRF Attacks

- **Solution 2**

- Use CSRF tokens via
`flask_wtf.csrf.CSRFProtect`

- **flask_wtf.csrf.CSRFProtect**

- An integration of Flask and WTForms

- **WTForms**

- “... a flexible forms validation and rendering library for Python web development.”
 - <https://wtforms.readthedocs.io/en/3.2.x/>

CSRF Attacks

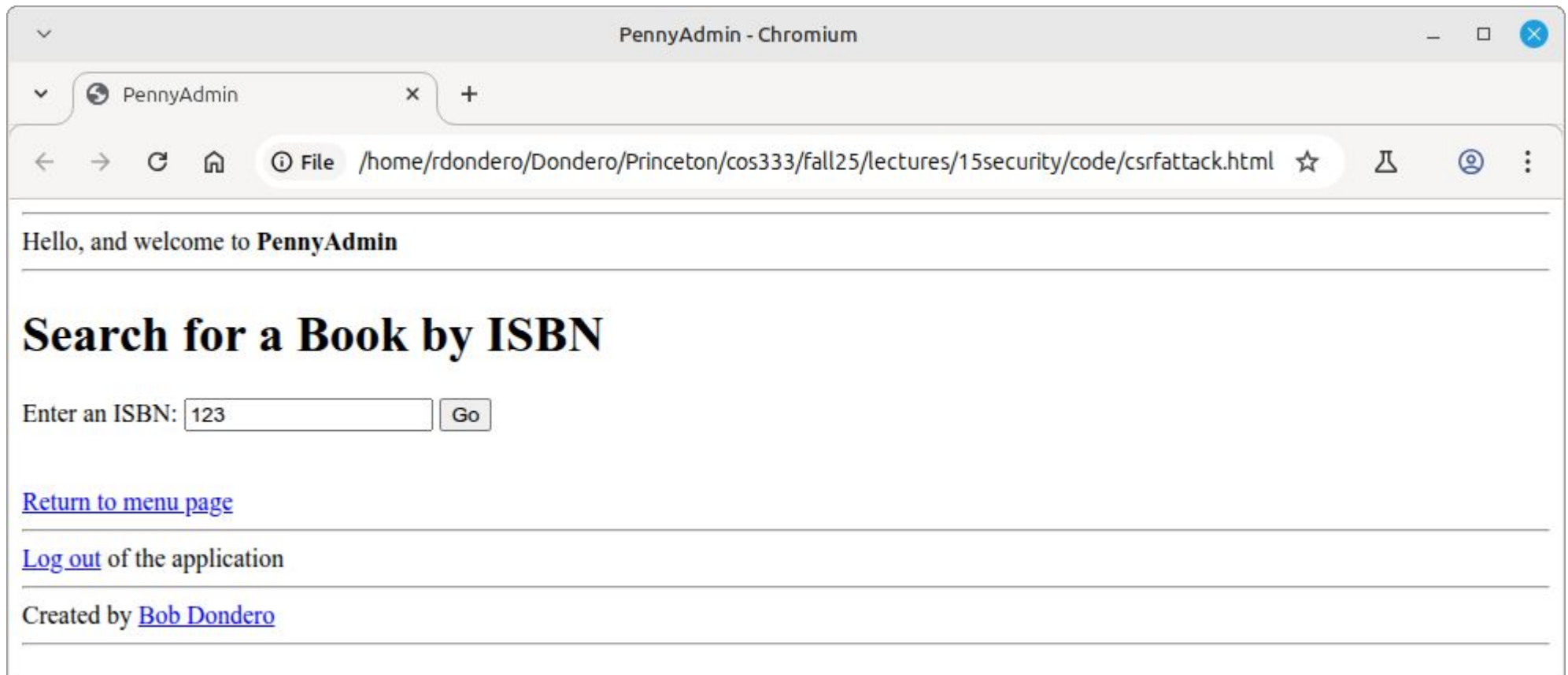
- **See PennyAdmin07bCsrfToken**
 - runserver.py
 - penny.sql, penny.sqlite
 - database.py
 - header.html, footer.html
 - index.html, show.html,
 - **add.html, delete.html, reportresults.html**
 - **login.html, signup.html, loggedout.html**
 - top.py, **penny.py, auth.py**

CSRF Attacks

- **See PennyAdmin07bCsrfToken**
 - Example:
 - Browser user visits app and logs in; browser session is authenticated/authorized
 - Attacker tricks user into visiting **csrfattack.html**
 - User submits form on csrfattack.html

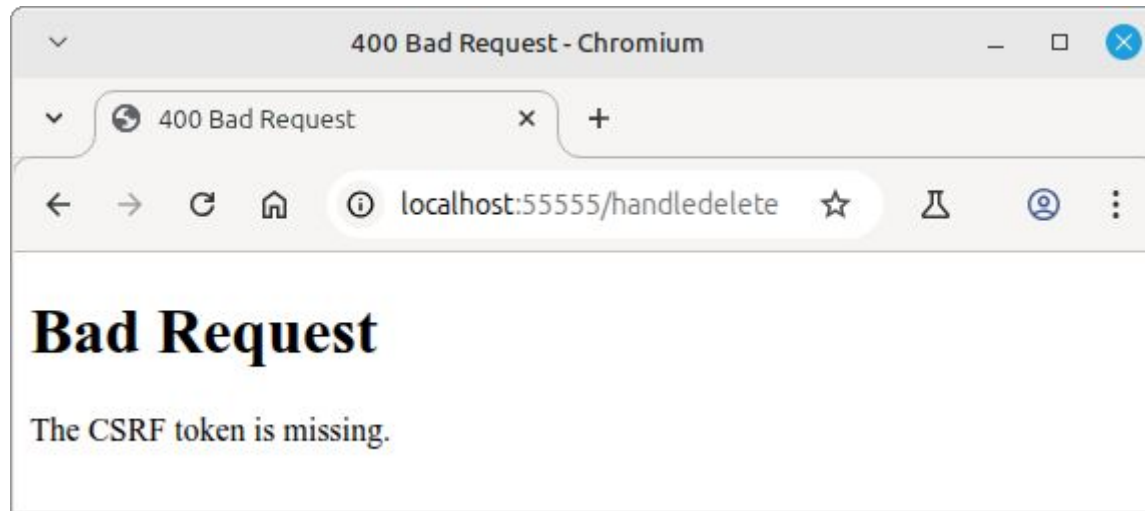
CSRF Attacks

- **See PennyAdmin07bCsrfToken**
 - Example:



CSRF Attacks

- **See PennyAdmin07bCsrfToken**
 - Example:



App
rejects
request

CSRF Attacks

- Note:
 - Really should protect **all** POST requests with CSRF tokens
 - POST requests via forms (shown)
 - POST requests via AJAX (not shown)

CSRF Attacks

- Q: Project concern?
- A: Yes

Lecture Summary

- In this lecture we covered:
 - Cookie forgery attacks
 - Cookie privacy attacks
 - Cross-site request forgery (CSRF) attacks