Security Issues in Web Programming (Part 3)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

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

Quick Review

- Recall **PennyAdmin06Auth** app
  - Username cookie exists => user is authenticated

Agenda

- **Cookie forgery attacks**
- Cross-site request forgery (CSRF) attacks
- Data storage attacks

Cookie Forgery Attacks

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

Cookie Forgery Attacks

- Recall **PennyAdmin06Auth** app
  - Example:
    - Attacker runs **cookieforgeryattack.py**
      - Sends forged username cookie to PennyAdmin app

Cookie Forgery Attacks

- Recall **PennyAdmin06Auth** 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 **PennyAdmin06Auth** 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()
'[redacted]'
>>> 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 **PennyAdmin07Encrypt** 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 **PennyAdmin07Encrypt** app
  - Example:
    - Attacker runs **cookieforgeryattack.py**
      - Sends forged username cookie to PennyAdmin app

Cookie Forgery Attacks

- See **PennyAdmin07Encrypt** 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.
$
```

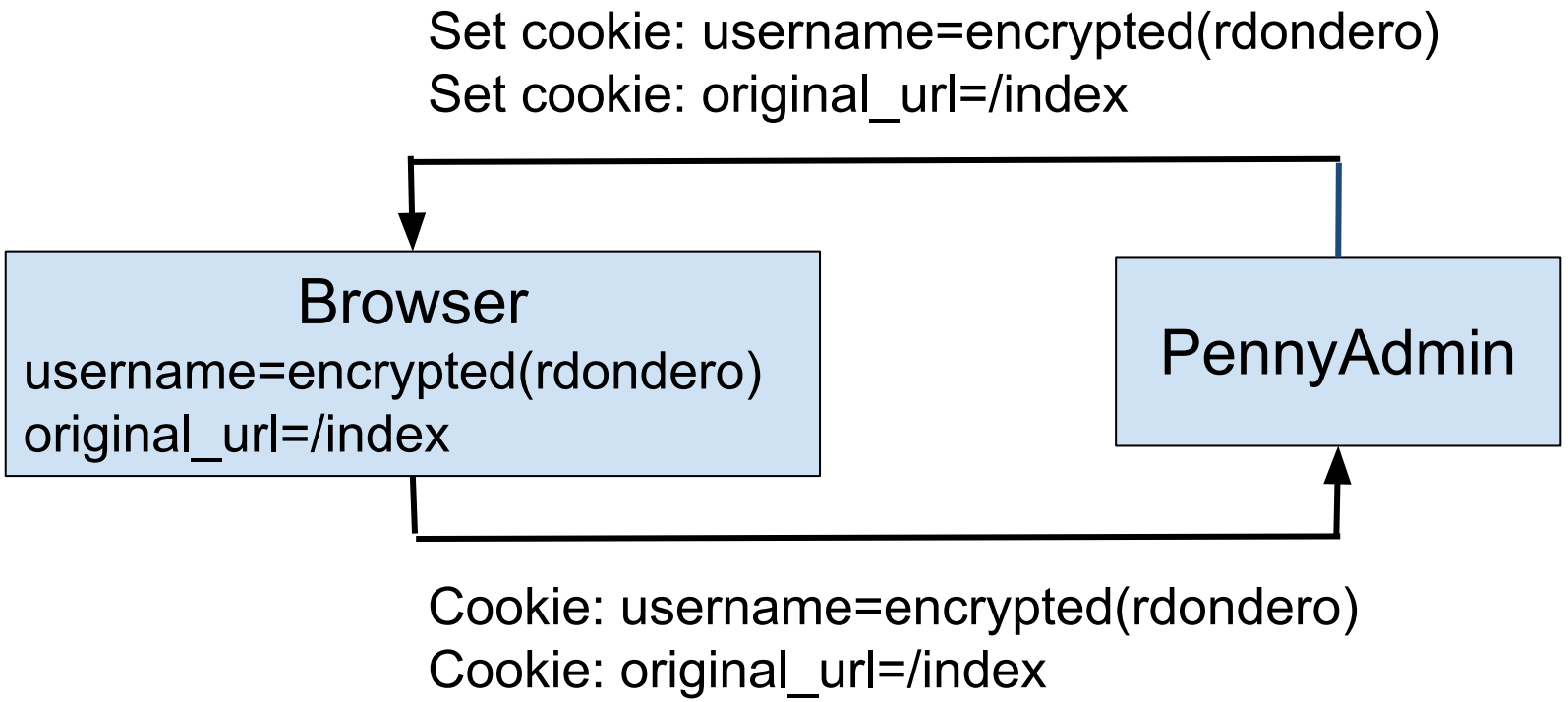
App rejects username, redirects to login page

Cookie Forgery Attacks

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

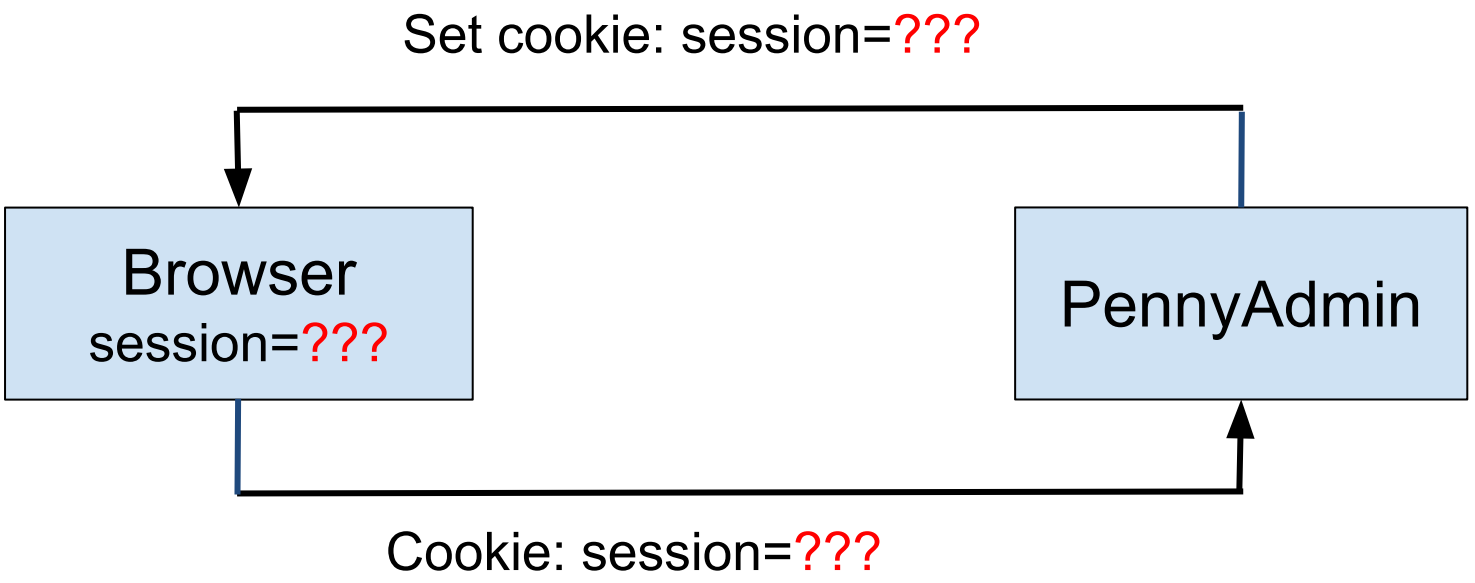
Cookie Forgery Attacks

Without sessions:



Cookie Forgery Attacks

With sessions:



eyJvcmlnaW5hbF91cmwiOiJodHRwOi8vbG9jYWxob3N0OjU1NTU1LyIsInVzZXJuYW1lIjoicmRvbmRlcm8ifQ.YsDrnQ.fc45qAmc0Vk6pAr32bQnogTtx1c

Using my secret key, decrypts to:

original_url=/index; username=rdondero;

Cookie Forgery Attacks

- See **PennyAdmin08Session** 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 **PennyAdmin08Session** app
  - Example:
    - Attacker runs **cookieforgeryattack.py**
      - Sends forged session cookie to PennyAdmin app

Cookie Forgery Attacks

- See **PennyAdmin08Session** 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
- **Cross-site request forgery (CSRF) attacks**
- Data storage 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

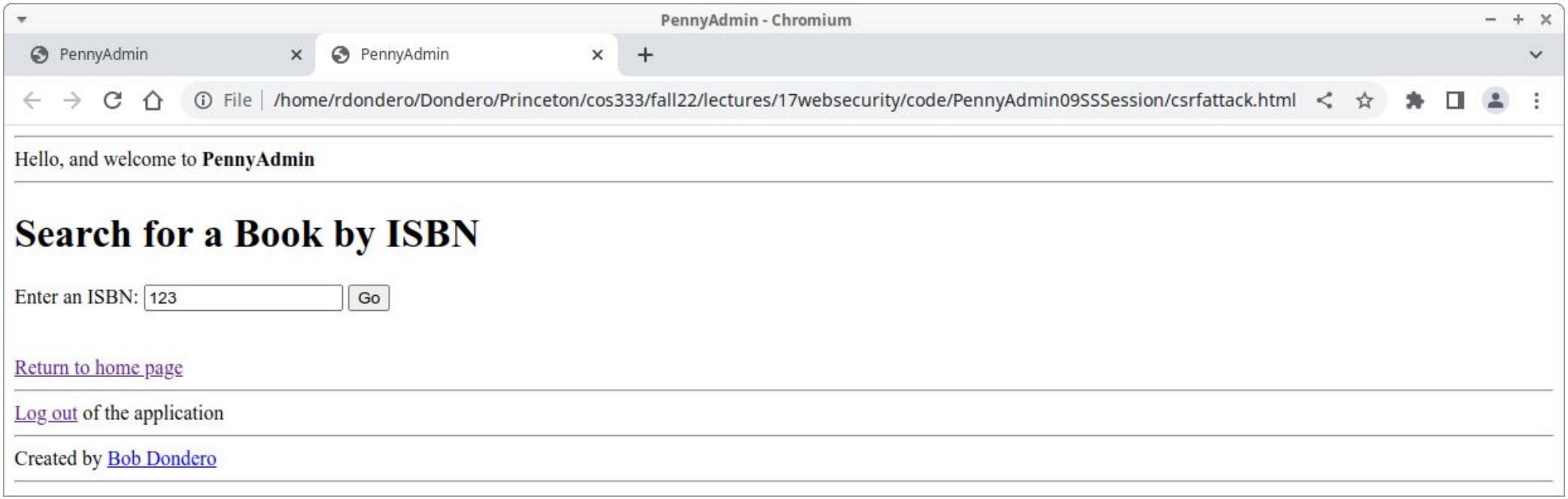
- **Problem**:
  - PennyAdmin is vulnerable to CSRF attacks

CSRF Attacks

- **Recall PennyAdmin08Session**:
  - 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

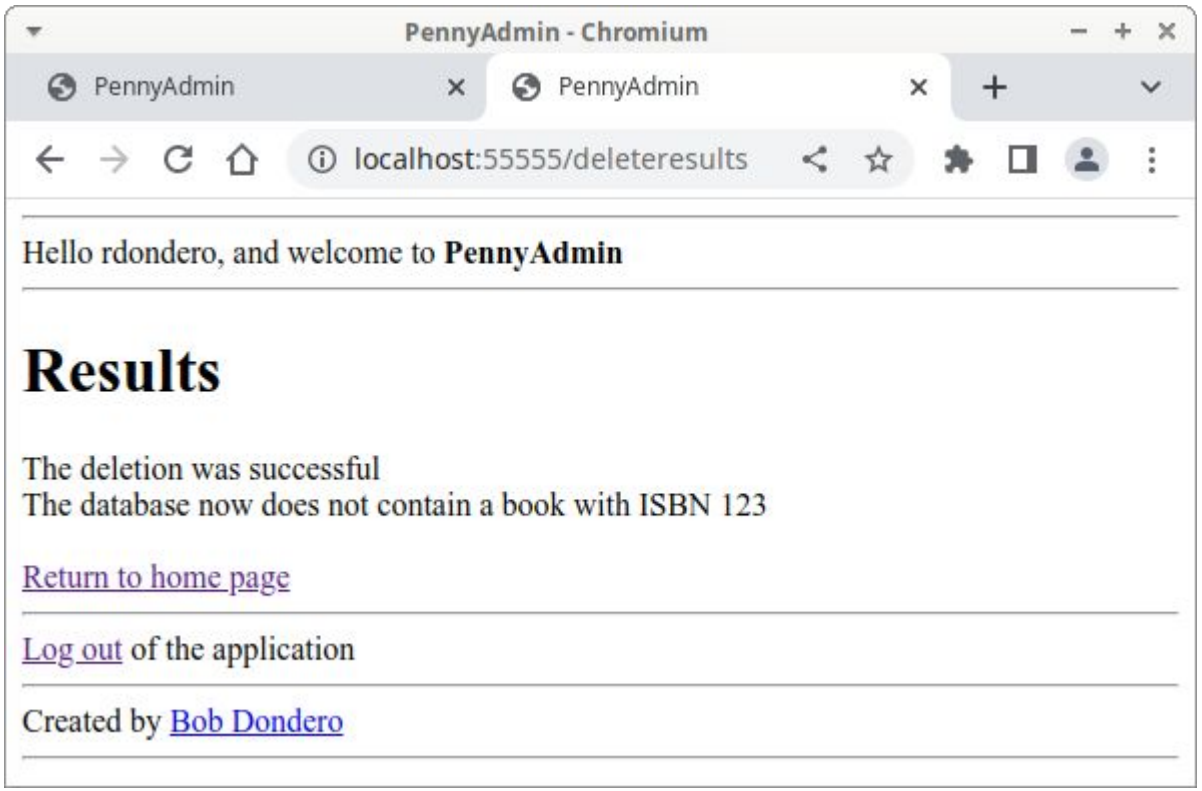
CSRF Attacks

- **Recall PennyAdmin08Session**:
  - Example:



CSRF Attacks

- **Recall PennyAdmin08Session**:
  - Example (cont.):



User unwittingly 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

CSRF Attacks

- **See PennyAdmin09CsrfToken**
  - 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

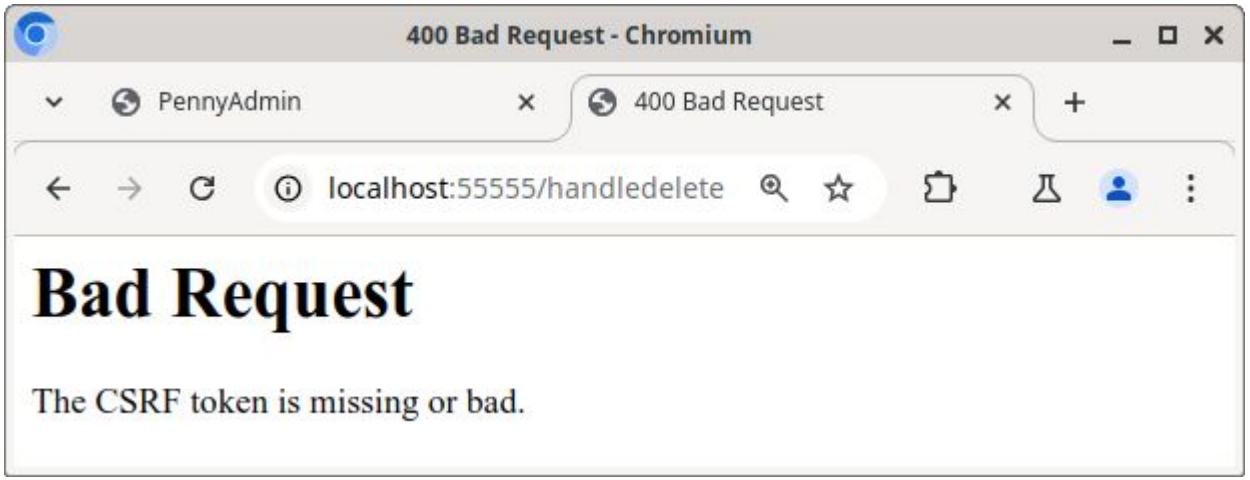
- **See PennyAdmin09CsrfToken**
  - Example:
    - Browser user visits app and logs in; browser session is authenticated/authorized
    - Attacker tricks user into visiting **csrfattack.html**
    - User submits form on csrfattack.html

CSRF Attacks

- **See PennyAdmin09CsrfToken**
  - Example:

CSRF Attacks

- **See PennyAdmin09CsrfToken**
  - Example (cont.):



App rejects request

CSRF Attacks

- **Solution 2**
  - Use CSRF tokens via ***flask_wtf.csrf.CSRFProtect***

CSRF Attacks

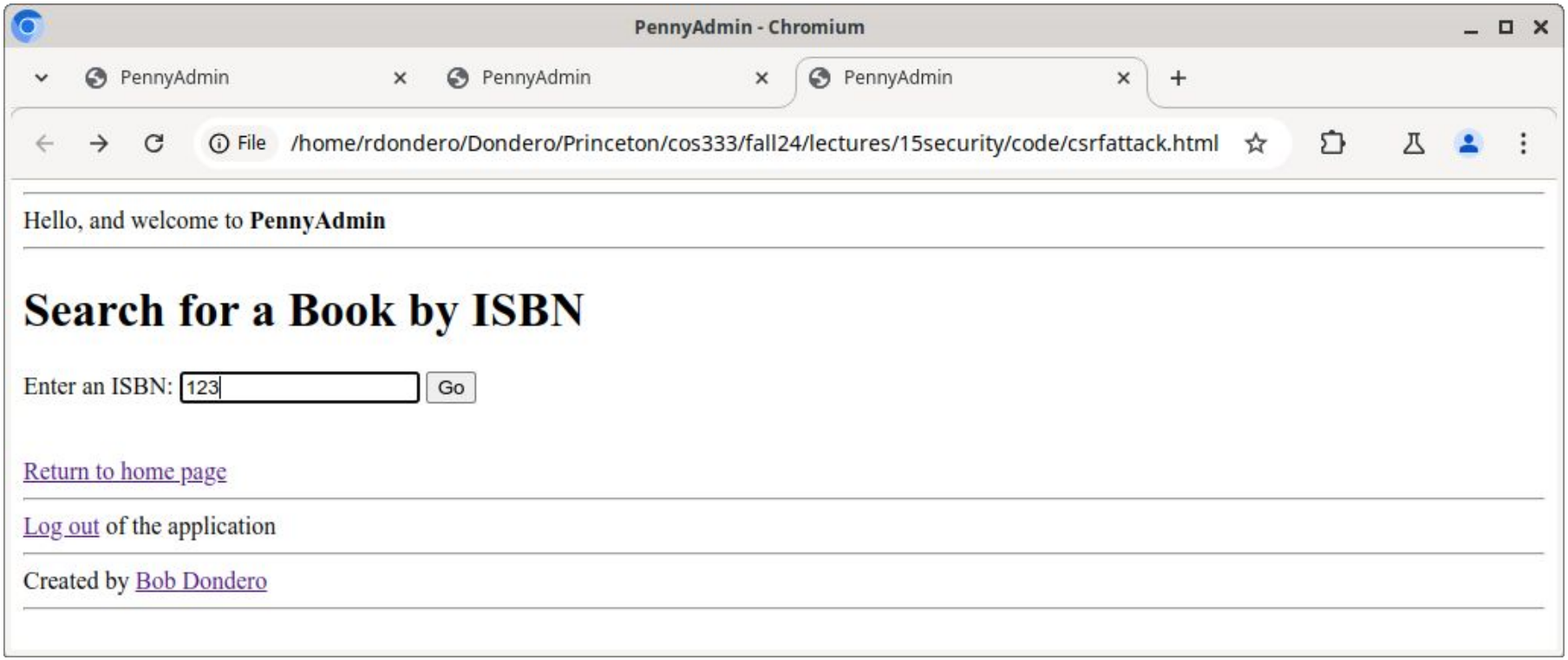
- **See PennyAdmin10CsrfToken**
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - header.html, footer.html
  - index.html, show.html,
  - **add.html**, **delete.html**, reportresults.html
  - **login.html**, **signup.html**, loggedout.html
  - **top.py**, **penny.py**, **auth.py**

CSRF Attacks

- **See PennyAdmin10CsrfToken**
  - 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

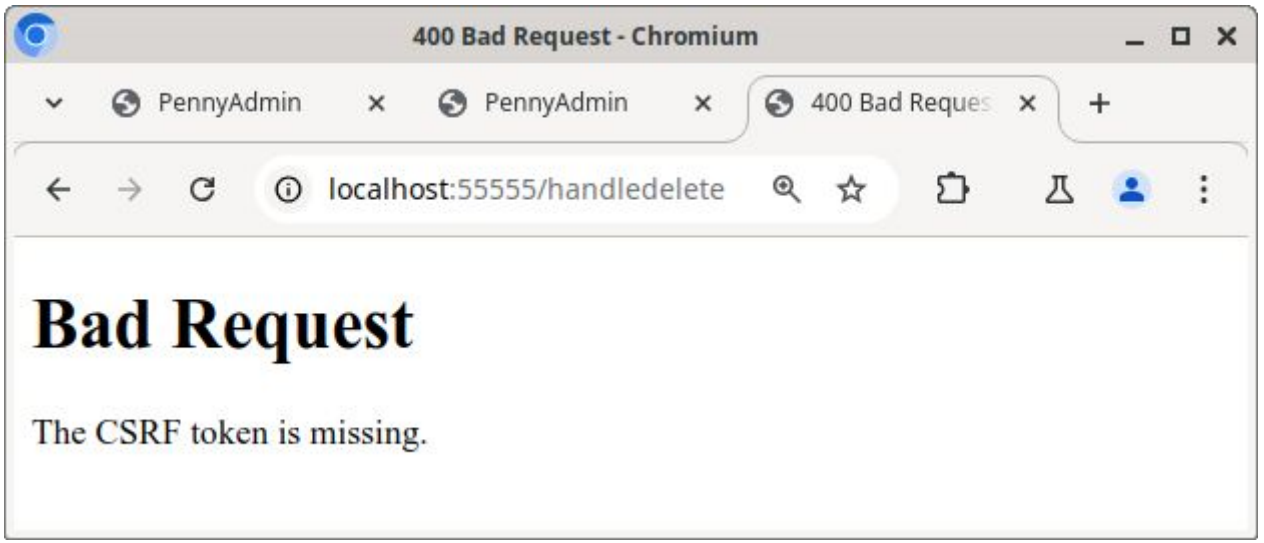
CSRF Attacks

- **See PennyAdmin10CsrfToken**
  - Example:



CSRF Attacks

- **See PennyAdmin10CsrfToken**
  - 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

Agenda

- Cookie forgery attacks
- Cross-site request forgery (CSRF) attacks
- **Data storage attacks**

Data Storage Attacks

- **Problem:**
  - PennyAdmin app stores passwords in DB
  - If attacker gains access to DB
  - … Then attacker learns passwords

Data Storage Attacks

- **Insight:**
  - PennyAdmin doesn't really need to store passwords
  - It's sufficient for PennyAdmin to know if a given password is correct

Data Storage Attacks

- **Solution:**
  - Store ***password hash codes*** instead of passwords
    - `hash_code = hash(password)`

Data Storage Attacks

- Which hash function?
  - ***md5***?
    - `hash_code = md5(password)`
    - No! See https://en.wikipedia.org/wiki/MD5
  - ***sha256***?
    - `hash_code = sha256(password)`
    - Yes! See https://en.wikipedia.org/wiki/SHA-2

Data Storage Attacks

- See **PennyAdmin11Hash** 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**

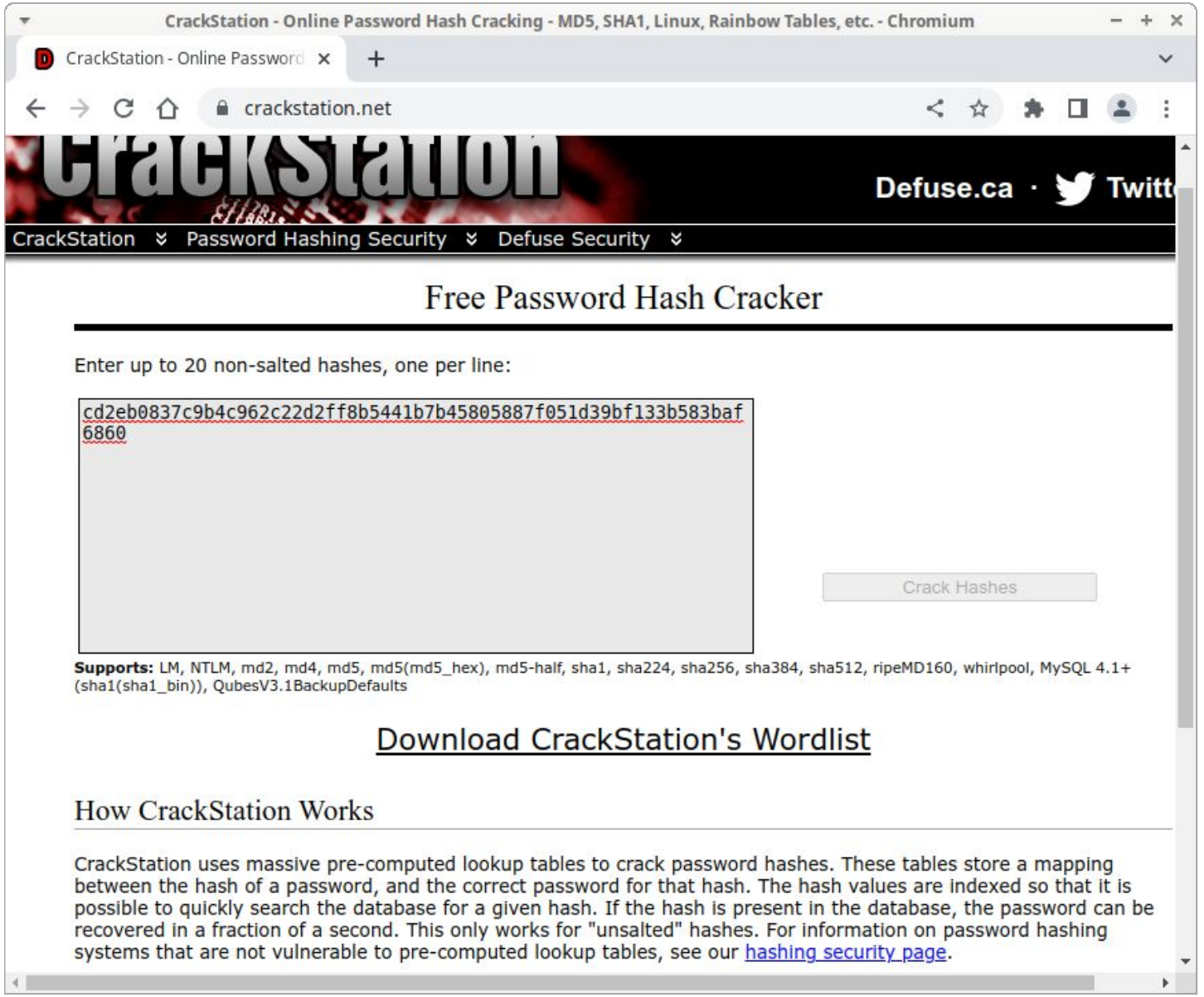
Data Storage Attacks

- **Problem:**
  - PennyAdmin app stores password hash codes in DB
  - If attacker gains access to DB, then…
    - Attacker learns password hash codes
  - If a password is common, then…
    - Attacker might find password hash code in a ***rainbow table*** (huge malevolent list of hash codes), and thereby learn the password

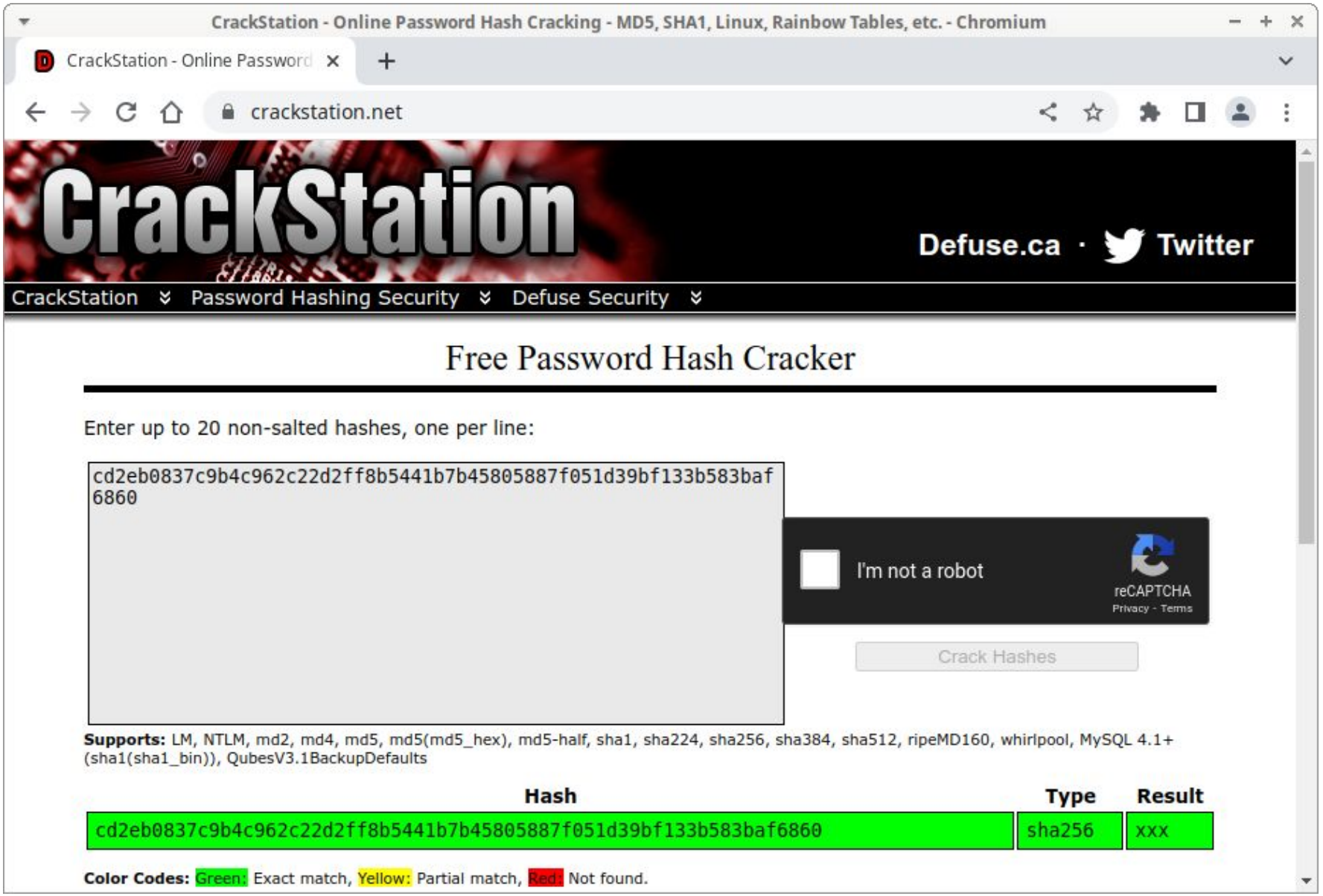
Data Storage Attacks

- **Example:**
  - Password:
    - xxx
  - sha256 hash code of that password:
    - cd2eb0837c9b4c962c22d2ff8b5441b7b45805887f051d39bf133b583baf6860
  - See https://crackstation.net/
    - Can derive xxx

Data Storage Attacks



Data Storage Attacks



Data Storage Attacks

- **Solution:**
  - Store hash codes of ***salted*** passwords
    - `hash_code = sha256('!@#$%^' + password)`
    - hash codes of salted passwords will not be found in a rainbow table

Data Storage Attacks

- **Problem:**
  - If an attacker learns the app's salt string, then the attacker (with lots of effort) might generate a rainbow table that contains hash codes for common salted passwords
    - … And so might discover the app's (common) passwords
- **Solution:**
  - Use a different salt string for each password

Data Storage Attacks

Salting and sha256 hashing in Python

```
$ python
>>> import werkzeug.security
>>> h = werkzeug.security.generate_password_hash('xxx', 'pbkdf2')
>>> h
'pbkdf2:sha256:600000$G8hNoAKf6ttD5iBa$262b04f2f287889ddffd77b0a735
b543491954d917d20bb36ae6ce2bd0ee5fde'
>>> werkzeug.security.check_password_hash(h, 'xxx')
True
>>> werkzeug.security.check_password_hash(h, 'yyy')
False
>>> quit()
$
```

Data Storage Attacks

Salting and sha256 hashing in Python

algorithm salt hashcode

```
pbkdf2:sha256:600000$G8hNoAKf6ttD5iBa$262b04
f2f287889ddffd77b0a735b543491954d917d20bb36a
e6ce2bd0ee5fde
```

Data Storage Attacks

- See **PennyAdmin12SaltHash** 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**

Data Storage Attacks

- Q: Project concern?

- A: **Yes**
  - If your app stores passwords

Summary

- We have covered:
  - Cookie forgery attacks
  - Cross-site request forgery (CSRF) attacks
  - Data storage attacks