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

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

cookieforgeryattack.py (Page 1 of 1)

```python
#!/usr/bin/env python

#----------------------------------------------------------------------
# cookieforgeryattack.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import sys
import socket

def main():

    if len(sys.argv) != 3:
        print('Usage: python %s host port' % sys.argv[0])
        sys.exit(1)

    try:
        host = sys.argv[1]
        port = int(sys.argv[2])

        with socket.socket() as sock:
            sock.connect((host, port))

            out_flo = sock.makefile(mode='w', encoding='iso-8859-1')
            out_flo.write('GET ' + '/show' + ' HTTP/1.1\r\n')
            out_flo.write('Host: ' + host + '\r\n')
            out_flo.write('Cookie: username=rdondero\r\n')
            out_flo.write('\r\n')
            out_flo.flush()

            in_flo = sock.makefile(mode='r', encoding='iso-8859-1')
            for line in in_flo:
                print(line, end='')

    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

This page is intentionally blank.

Cookie Forgery Attacks

- Recall **<u>PennyAdmin06Auth</u>** 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 **<u>PennyAdmin06Auth</u>** 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**
  -

PennyAdmin07Encrypt/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import os
import cryptocode
import flask
import dotenv
import database

from top import app

#----------------------------------------------------------------------

dotenv.load_dotenv()
secret_key = os.environ['APP_SECRET_KEY']

#----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

#----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''
    html = flask.render_template('login.html', msg=msg)
    response = flask.make_response(html)
    return response

#----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.request.cookies.get('original_url', '/index')
    response = flask.redirect(original_url)
    encrypted_username = cryptocode.encrypt(username, secret_key)
    response.set_cookie('username', encrypted_username)
    return response

#----------------------------------------------------------------------

```

PennyAdmin07Encrypt/auth.py (Page 2 of 2)

```
@app.route('/logout', methods=['GET'])
def logout():

    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)

    # Delete cookies in the browser by setting them to expire at
    # a time that is in the past.
    response.set_cookie('username', '', expires=0)
    response.set_cookie('original_url', '', expires=0)

    return response

#----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''
    html_code = flask.render_template('signup.html',
        error_msg=error_msg)
    response = flask.make_response(html_code)
    return response

#----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))
    successful = database.add_user(username, password)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))
    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))

#----------------------------------------------------------------------

def authenticate():

    encrypted_username = flask.request.cookies.get('username')

    if encrypted_username is None:
        response = flask.redirect(flask.url_for('login'))
        response.set_cookie('original_url', flask.request.url)
        flask.abort(response)

    username = cryptocode.decrypt(encrypted_username, secret_key)
    if not username:
        response = flask.redirect(flask.url_for('login'))
        response.set_cookie('original_url', flask.request.url)
        flask.abort(response)

    return username
```

Cookie Forgery Attacks

- See **PennyAdmin07Encrypt** app
  - Example:
    - Attacker runs **cookieforgeryattack.py**
      - Sends forged username cookie to PennyAdmin app

cookieforgeryattack.py (Page 1 of 1)

```python
#!/usr/bin/env python

#----------------------------------------------------------------------
# cookieforgeryattack.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import sys
import socket

def main():

    if len(sys.argv) != 3:
        print('Usage: python %s host port' % sys.argv[0])
        sys.exit(1)

    try:
        host = sys.argv[1]
        port = int(sys.argv[2])

        with socket.socket() as sock:
            sock.connect((host, port))

            out_flo = sock.makefile(mode='w', encoding='iso-8859-1')
            out_flo.write('GET ' + '/show' + ' HTTP/1.1\r\n')
            out_flo.write('Host: ' + host + '\r\n')
            out_flo.write('Cookie: username=rdondero\r\n')
            out_flo.write('\r\n')
            out_flo.flush()

            in_flo = sock.makefile(mode='r', encoding='iso-8859-1')
            for line in in_flo:
                print(line, end='')

    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

This page is intentionally blank.

Cookie Forgery Attacks

- See **<u>PennyAdmin07Encrypt</u>** 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.
$
```

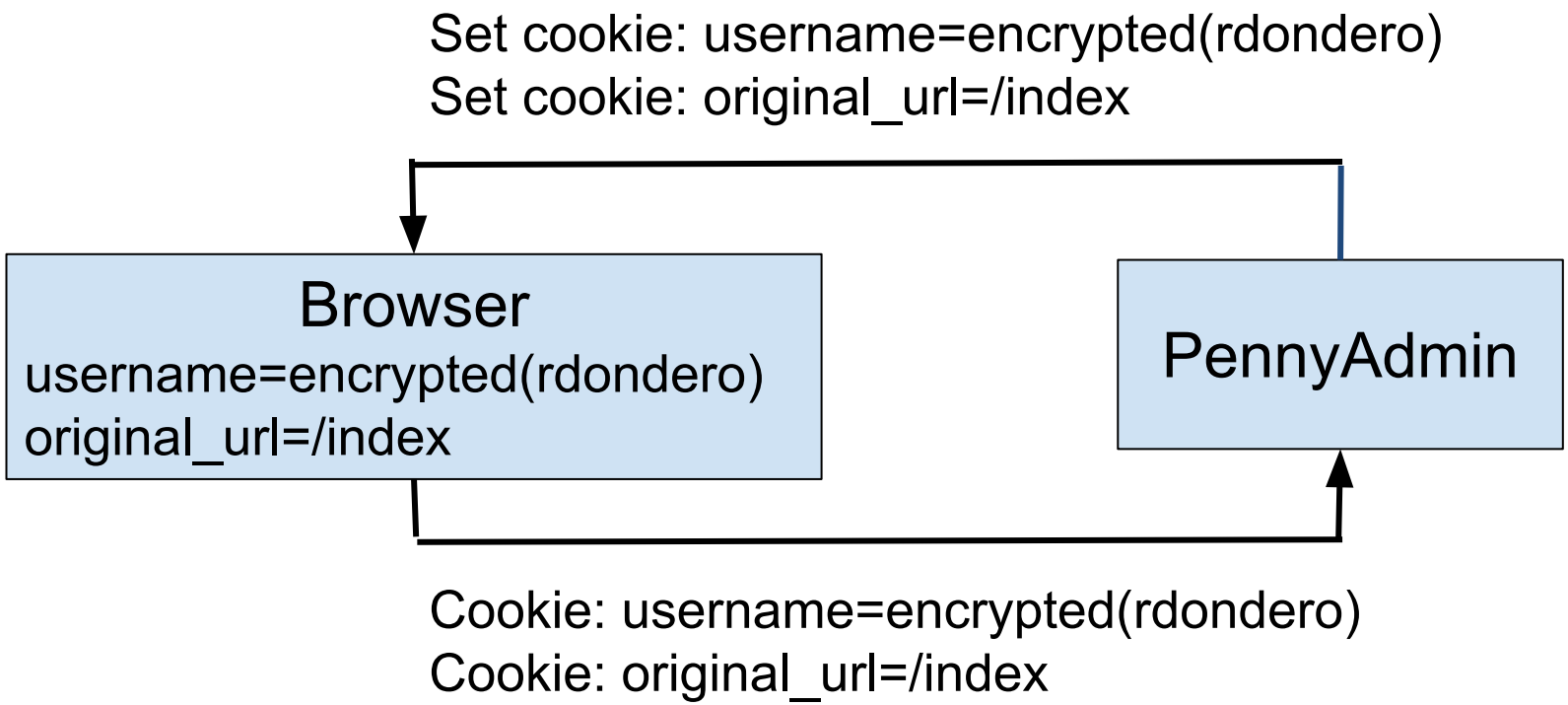
App rejects username, redirects to login page

Cookie Forgery Attacks

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

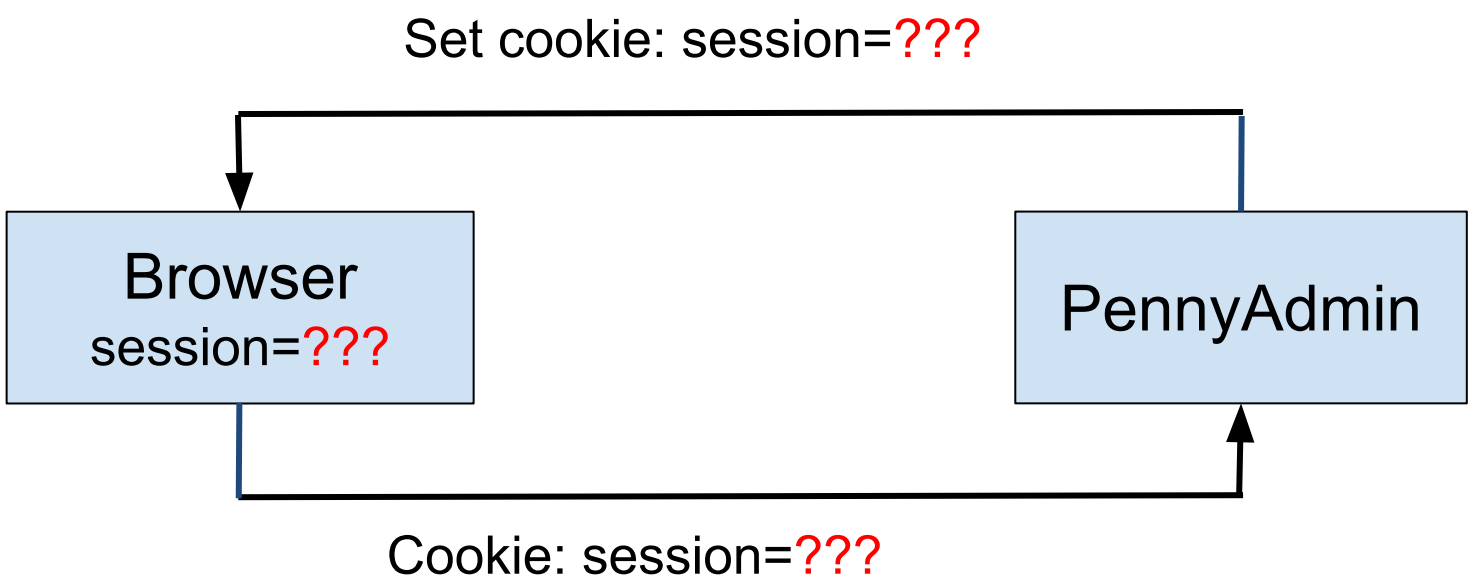
Cookie Forgery Attacks

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**

PennyAdmin08Session/top.py (Page 1 of 1)

```
#!/usr/bin/env python

#----------------------------------------------------------------------
# top.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import os
import dotenv
import flask

app = flask.Flask('penny', template_folder='.')

dotenv.load_dotenv()
app.secret_key = os.environ['APP_SECRET_KEY']
```

blank (Page 1 of 1)

This page is intentionally blank.

PennyAdmin08Session/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import os
import flask
import dotenv
import database

from top import app

#----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

#----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''

    html = flask.render_template('login.html', msg=msg)
    response = flask.make_response(html)
    return response

#----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
    return response

#----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
```

PennyAdmin08Session/auth.py (Page 2 of 2)

```
    return response

#----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''

    html_code = flask.render_template('signup.html',
        error_msg=error_msg)
    response = flask.make_response(html_code)
    return response

#----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))
    successful = database.add_user(username, password)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))

    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))

#----------------------------------------------------------------------

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
    return username
```

Cookie Forgery Attacks

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

cookieforgeryattack.py (Page 1 of 1)

```python
#!/usr/bin/env python

#----------------------------------------------------------------------
# cookieforgeryattack.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import sys
import socket

def main():

    if len(sys.argv) != 3:
        print('Usage: python %s host port' % sys.argv[0])
        sys.exit(1)

    try:
        host = sys.argv[1]
        port = int(sys.argv[2])

        with socket.socket() as sock:
            sock.connect((host, port))

            out_flo = sock.makefile(mode='w', encoding='iso-8859-1')
            out_flo.write('GET ' + '/show' + ' HTTP/1.1\r\n')
            out_flo.write('Host: ' + host + '\r\n')
            out_flo.write('Cookie: username=rdondero\r\n')
            out_flo.write('\r\n')
            out_flo.flush()

            in_flo = sock.makefile(mode='r', encoding='iso-8859-1')
            for line in in_flo:
                print(line, end='')

    except Exception as ex:
        print(ex, file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

blank (Page 1 of 1)

This page is intentionally blank.

Cookie Forgery Attacks

- See **<u>PennyAdmin08Session</u>** 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

csrfattack.html (Page 1 of 1)

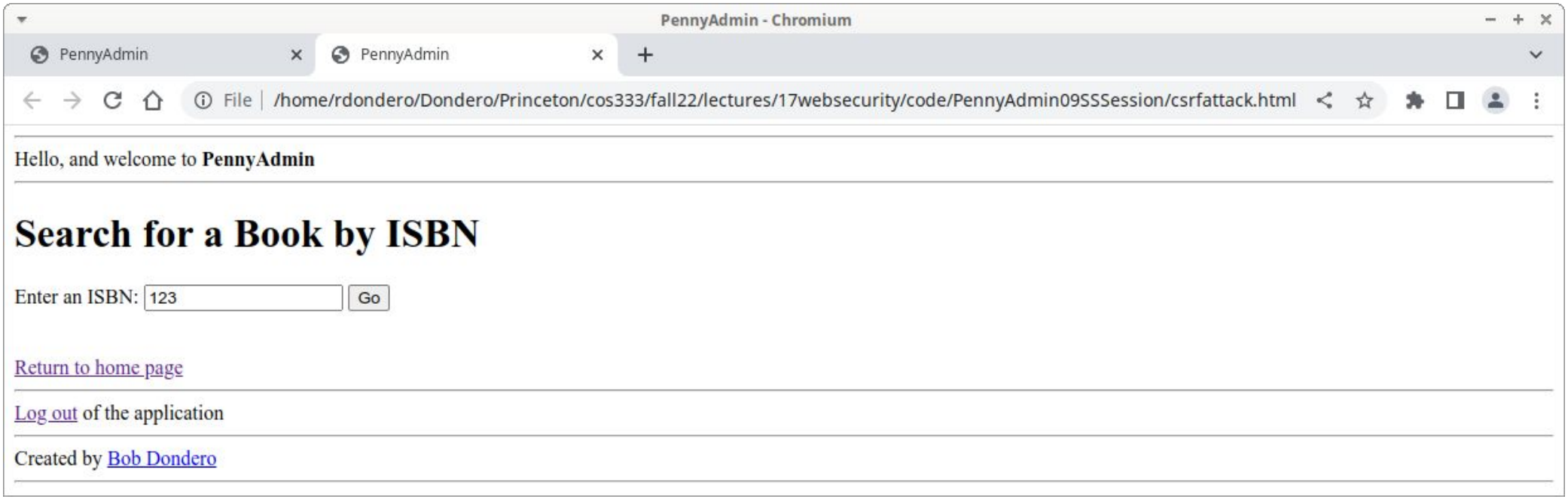
```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:     <hr>Hello, and welcome to <strong>PennyAdmin</strong><hr>
 8:       <h1>Search for a Book by ISBN</h1>
 9:       <form action="http://localhost:55555/handledelete" method="post">
10:          Enter an ISBN:
11:          <input type="text" name="isbn" autofocus>
12:          <input type="submit" value="Go">
13:       </form>
14:       <br>
15:       <br>
16:       <a href="http://localhost:55555/index">Return to home page</a>
17:       <br>
18:       <hr>
19:       <a href="http://localhost:55555/logout">Log out</a>
20:       of the application</br>
21:       <hr>
22:       Created by <a href="https://www.cs.princeton.edu/~rdondero">
23:       Bob Dondero</a>
24:       <hr>
25:    </body>
26: </html>
```

PennyAdmin09CsrfToken/add.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Add a Book</h1>
 9:       <form action="/handleadd" method="post">
10:          <input type="hidden" name="csrf_token" value="{{csrf_token}}">
11:
12:          Enter an ISBN:
13:          <input type="text" name="isbn" autofocus
14:             required pattern=".*\S.*"
15:             title="At least one non-white-space char">
16:          <br>
17:
18:          Enter an author:
19:          <input type="text" name="author"
20:             required pattern=".*\S.*"
21:             title="At least one non-white-space char">
22:          <br>
23:
24:          Enter a title:
25:          <input type="text" name="title"
26:             required pattern=".*\S.*"
27:             title="At least one non-white-space char">
28:          <br>
29:
30:          <input type="submit" value="Go">
31:       </form>
32:       <br>
33:       <a href="/index">Return to home page</a>
34:       <br>
35:       {% include 'footer.html' %}
36:    </body>
37: </html>
```

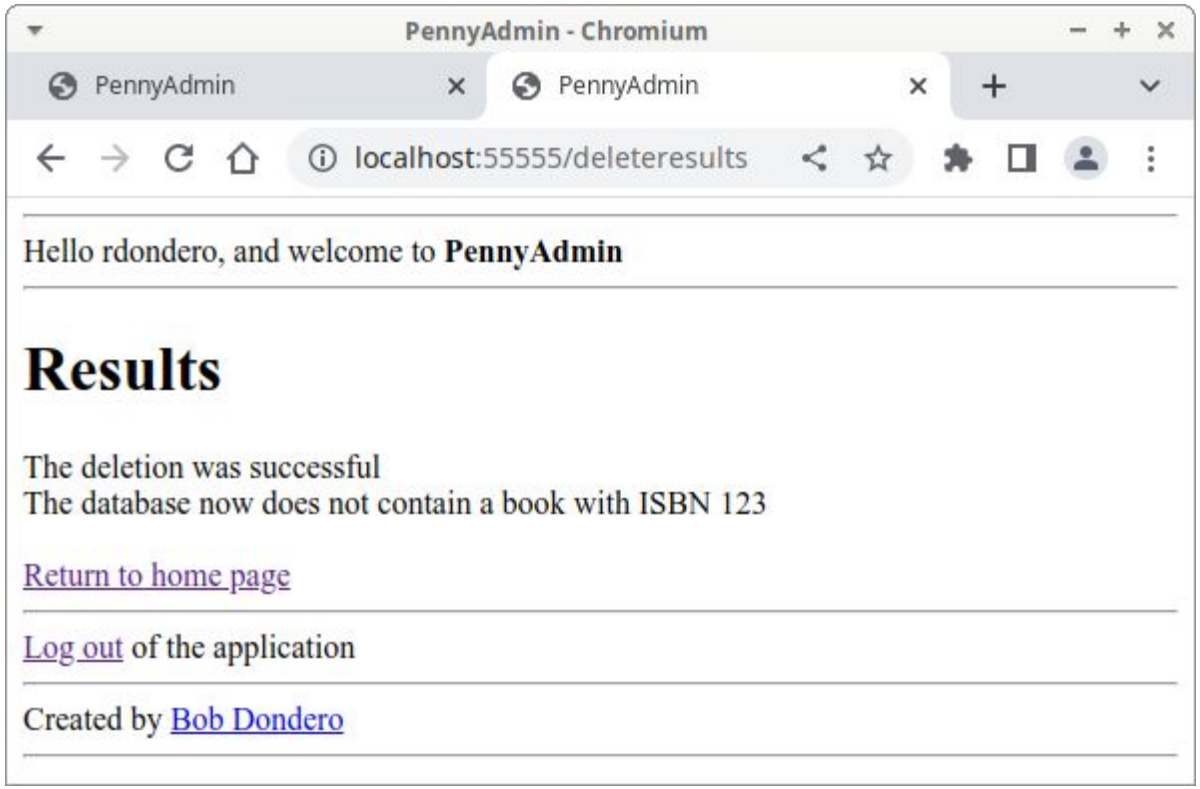
CSRF Attacks

- **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**

csrfattack.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:     <hr>Hello, and welcome to <strong>PennyAdmin</strong><hr>
 8:       <h1>Search for a Book by ISBN</h1>
 9:       <form action="http://localhost:55555/handledelete" method="post">
10:          Enter an ISBN:
11:          <input type="text" name="isbn" autofocus>
12:          <input type="submit" value="Go">
13:       </form>
14:       <br>
15:       <br>
16:       <a href="http://localhost:55555/index">Return to home page</a>
17:       <br>
18:       <hr>
19:       <a href="http://localhost:55555/logout">Log out</a>
20:       of the application</br>
21:       <hr>
22:       Created by <a href="https://www.cs.princeton.edu/~rdondero">
23:       Bob Dondero</a>
24:       <hr>
25:    </body>
26: </html>
```

PennyAdmin09CsrfToken/add.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Add a Book</h1>
 9:       <form action="/handleadd" method="post">
10:          <input type="hidden" name="csrf_token" value="{{csrf_token}}">
11:
12:          Enter an ISBN:
13:          <input type="text" name="isbn" autofocus
14:             required pattern=".*\S.*"
15:             title="At least one non-white-space char">
16:          <br>
17:
18:          Enter an author:
19:          <input type="text" name="author"
20:             required pattern=".*\S.*"
21:             title="At least one non-white-space char">
22:          <br>
23:
24:          Enter a title:
25:          <input type="text" name="title"
26:             required pattern=".*\S.*"
27:             title="At least one non-white-space char">
28:          <br>
29:
30:          <input type="submit" value="Go">
31:       </form>
32:       <br>
33:       <a href="/index">Return to home page</a>
34:       <br>
35:       {% include 'footer.html' %}
36:    </body>
37: </html>
```

PennyAdmin09CsrfToken/delete.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>PennyAdmin</title>
   </head>
   <body>
      {% include 'header.html' %}
      <h1>Delete a Book</h1>
      <form action="/handledelete" method="post">
         <input type="hidden" name="csrf_token" value="{{csrf_token}}">

         Enter an ISBN:
         <input type="text" name="isbn" autofocus
            required pattern=".*\S.*"
            title="At least one non-white-space char">
         <input type="submit" value="Go">
      </form>
      <br>
      <br>
      <a href="/index">Return to home page</a>
      <br>
      {% include 'footer.html' %}
   </body>
</html>
```

PennyAdmin09CsrfToken/login.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h1>Login to Penny</h1>
      <!-- Use post instead of get for security. -->
      <form action="/handlelogin" method="post">
         <input type="hidden" name="csrf_token" value="{{csrf_token}}">

         <table>
            <tbody>
               <tr>
                  <td style="text-align:right">User name:</td>
                  <td><input type="text" name="username" autofocus
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  </td>
               </tr>
               <tr>
                  <td style="text-align:right">Password:</td>
                  <td><input type="password" name="password"
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  <td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" value="Go"></td>
               </tr>
            </tbody>
         </table>
      </form>
      <br>
      Don't have an account? <a href="/signup">Sign up</a>
      <br>
      <br>
      <strong>{{msg}}</strong>
   </body>
</html>
```

PennyAdmin09CsrfToken/signup.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h1>Penny New User Signup</h1>
      <!-- Use post instead of get for security. -->
      <form action="/handlesignup" method="post">
         <input type="hidden" name="csrf_token" value="{{csrf_token}}">

         <table>
            <tbody>
               <tr>
                  <td style="text-align:right">User name:</td>
                  <td><input type="text" name="username" autofocus
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  </td>
               </tr>
               <tr>
                  <td style="text-align:right">Password:</td>
                  <td><input type="password" name="password"
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  <td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" value="Go"></td>
               </tr>
            </tbody>
         </table>
      </form>
      <br>
      <strong>{{error_msg}}</strong>
   </body>
</html>
```

PennyAdmin09CsrfToken/penny.py (Page 1 of 3)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# penny.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import flask
import database
import auth

from top import app

#-----------------------------------------------------------------------

@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def index():

    username = auth.authenticate()
    is_authorized = database.is_authorized(username)

    html_code = flask.render_template('index.html', username=username,
        is_authorized=is_authorized)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/show', methods=['GET'])
def show():

    username = auth.authenticate()

    books = database.get_books()
    html_code = flask.render_template('show.html',
        username=username, books=books)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

def report_results(username, message1, message2):

    html_code = flask.render_template('reportresults.html',
        username=username, message1=message1, message2=message2)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/add', methods=['GET'])
def add():

    username = auth.authenticate()
    if not database.is_authorized(username):
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    csrf_token = os.urandom(12).hex()
    flask.session['csrf_token'] = csrf_token

    html_code = flask.render_template('add.html',
```

PennyAdmin09CsrfToken/penny.py (Page 2 of 3)

```
 66:         username=username, csrf_token=csrf_token)
 67:
 68:     response = flask.make_response(html_code)
 69:     return response
 70:
 71: #----------------------------------------------------------------------
 72:
 73: @app.route('/handleadd', methods=['POST'])
 74: def handle_add():
 75:
 76:     username = auth.authenticate()
 77:     if not database.is_authorized(username):
 78:         html_code = 'You are not authorized to add books.'
 79:         response = flask.make_response(html_code)
 80:         return response
 81:
 82:     csrf_token_from_session = flask.session.get('csrf_token')
 83:     csrf_token_from_request = flask.request.form.get('csrf_token')
 84:
 85:     if ((csrf_token_from_session is None)
 86:         or (csrf_token_from_request is None)
 87:         or (csrf_token_from_request != csrf_token_from_session)):
 88:         html_code = '<title>400 Bad Request</title>'
 89:         html_code += '<h1>Bad Request</h1>'
 90:         html_code += '<p>The CSRF token is missing or bad.</p>'
 91:         response = flask.make_response(html_code)
 92:         return response
 93:
 94:     isbn = flask.request.form.get('isbn')
 95:     if (isbn is None) or (isbn.strip() == ''):
 96:         return report_results(username, 'Missing ISBN', '')
 97:
 98:     author = flask.request.form.get('author')
 99:     if (author is None) or (author.strip() == ''):
100:         return report_results(username, 'Missing author', '')
101:
102:     title = flask.request.form.get('title')
103:     if (title is None) or (title.strip() == ''):
104:         return report_results(username, 'Missing title', '')
105:
106:     isbn = isbn.strip()
107:     author = author.strip()
108:     title = title.strip()
109:
110:     successful = database.add_book(isbn, author, title)
111:     if successful:
112:         message1 = 'The addition was successful'
113:         message2 = 'The database now contains a book with isbn ' + isbn
114:         message2 += ' author ' + author + ' and title ' + title
115:     else:
116:         message1 = 'The addition was unsuccessful'
117:         message2 = 'A book with ISBN ' + isbn + ' already exists'
118:
119:     return report_results(username, message1, message2)
120:
121: #----------------------------------------------------------------------
122:
123: @app.route('/delete', methods=['GET'])
124: def delete():
125:
126:     username = auth.authenticate()
127:     if not database.is_authorized(username):
128:         html_code = 'You are not authorized to delete books.'
129:         response = flask.make_response(html_code)
130:         return response
```

PennyAdmin09CsrfToken/penny.py (Page 3 of 3)

```
131:
132:     csrf_token = os.urandom(12).hex()
133:     flask.session['csrf_token'] = csrf_token
134:
135:     html_code = flask.render_template('delete.html',
136:         username=username, csrf_token=csrf_token)
137:
138:     response = flask.make_response(html_code)
139:     return response
140:
141: #----------------------------------------------------------------------
142:
143: @app.route('/handledelete', methods=['POST'])
144: def handle_delete():
145:
146:     username = auth.authenticate()
147:     if not database.is_authorized(username):
148:         html_code = 'You are not authorized to delete books.'
149:         response = flask.make_response(html_code)
150:         return response
151:
152:     csrf_token_from_session = flask.session.get('csrf_token')
153:     csrf_token_from_request = flask.request.form.get('csrf_token')
154:
155:     if ((csrf_token_from_session is None)
156:         or (csrf_token_from_request is None)
157:         or (csrf_token_from_request != csrf_token_from_session)):
158:         html_code = '<title>400 Bad Request</title>'
159:         html_code += '<h1>Bad Request</h1>'
160:         html_code += '<p>The CSRF token is missing or bad.</p>'
161:         response = flask.make_response(html_code)
162:         return response
163:
164:     isbn = flask.request.form.get('isbn')
165:     if (isbn is None) or (isbn.strip() == ''):
166:         return report_results(username, 'Missing ISBN', '')
167:
168:     isbn = isbn.strip()
169:
170:     database.delete_book(isbn)
171:
172:     message1 = 'The deletion was successful'
173:     message2 = 'The database now does not contain a book with ISBN '
174:     message2 += isbn
175:
176:     return report_results(username, message1, message2)
```

PennyAdmin09CsrfToken/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import flask
import database
from top import app
#-----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

#-----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''

    csrf_token = os.urandom(12).hex()
    flask.session['csrf_token'] = csrf_token

    html = flask.render_template('login.html',
        msg=msg, csrf_token=csrf_token)

    response = flask.make_response(html)
    return response

#-----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    csrf_token_from_session = flask.session.get('csrf_token')
    csrf_token_from_request = flask.request.form.get('csrf_token')

    if ((csrf_token_from_session is None)
        or (csrf_token_from_request is None)
        or (csrf_token_from_request != csrf_token_from_session)):
        html_code = '<title>400 Bad Request</title>'
        html_code += '<h1>Bad Request</h1>'
        html_code += '<p>The CSRF token is missing or bad.</p>'
        response = flask.make_response(html_code)
        return response

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
```

PennyAdmin09CsrfToken/auth.py (Page 2 of 2)

```
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
    return response

#-----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''

    csrf_token = os.urandom(12).hex()
    flask.session['csrf_token'] = csrf_token

    html_code = flask.render_template('signup.html',
        error_msg=error_msg, csrf_token=csrf_token)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))
    successful = database.add_user(username, password)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))

    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))

#-----------------------------------------------------------------------

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
    return username
```

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

csrfattack.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:     <hr>Hello, and welcome to <strong>PennyAdmin</strong><hr>
 8:       <h1>Search for a Book by ISBN</h1>
 9:       <form action="http://localhost:55555/handledelete" method="post">
10:          Enter an ISBN:
11:          <input type="text" name="isbn" autofocus>
12:          <input type="submit" value="Go">
13:       </form>
14:       <br>
15:       <br>
16:       <a href="http://localhost:55555/index">Return to home page</a>
17:       <br>
18:       <hr>
19:       <a href="http://localhost:55555/logout">Log out</a>
20:       of the application</br>
21:       <hr>
22:       Created by <a href="https://www.cs.princeton.edu/~rdondero">
23:       Bob Dondero</a>
24:       <hr>
25:    </body>
26: </html>
```

PennyAdmin09CsrfToken/add.html (Page 1 of 1)

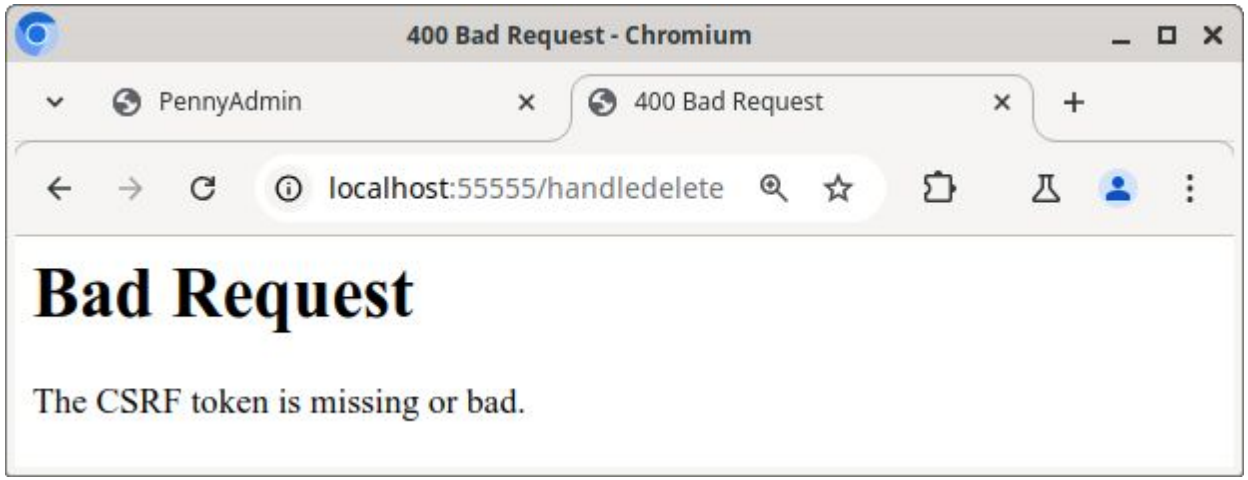
```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Add a Book</h1>
 9:       <form action="/handleadd" method="post">
10:          <input type="hidden" name="csrf_token" value="{{csrf_token}}">
11:
12:          Enter an ISBN:
13:          <input type="text" name="isbn" autofocus
14:             required pattern=".*\S.*"
15:             title="At least one non-white-space char">
16:          <br>
17:
18:          Enter an author:
19:          <input type="text" name="author"
20:             required pattern=".*\S.*"
21:             title="At least one non-white-space char">
22:          <br>
23:
24:          Enter a title:
25:          <input type="text" name="title"
26:             required pattern=".*\S.*"
27:             title="At least one non-white-space char">
28:          <br>
29:
30:          <input type="submit" value="Go">
31:       </form>
32:       <br>
33:       <a href="/index">Return to home page</a>
34:       <br>
35:       {% include 'footer.html' %}
36:    </body>
37: </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**

PennyAdmin10CsrfToken/add.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Add a Book</h1>
 9:       <form action="/handleadd" method="post">
10:          <input type="hidden" name="csrf_token"
11:             value="{{csrf_token()}}">
12:
13:          Enter an ISBN:
14:          <input type="text" name="isbn" autofocus
15:             required pattern=".*\S.*"
16:             title="At least one non-white-space char">
17:          <br>
18:
19:          Enter an author:
20:          <input type="text" name="author"
21:             required pattern=".*\S.*"
22:             title="At least one non-white-space char">
23:          <br>
24:
25:          Enter a title:
26:          <input type="text" name="title"
27:             required pattern=".*\S.*"
28:             title="At least one non-white-space char">
29:          <br>
30:
31:          <input type="submit" value="Go">
32:       </form>
33:       <br>
34:       <a href="/index">Return to home page</a>
35:       <br>
36:       {% include 'footer.html' %}
37:    </body>
38: </html>
```

PennyAdmin10CsrfToken/delete.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Delete a Book</h1>
 9:       <form action="/handledelete" method="post">
10:          <input type="hidden" name="csrf_token"
11:             value="{{csrf_token()}}">
12:
13:          Enter an ISBN:
14:          <input type="text" name="isbn" autofocus
15:             required pattern=".*\S.*"
16:             title="At least one non-white-space char">
17:          <input type="submit" value="Go">
18:       </form>
19:       <br>
20:       <br>
21:       <a href="/index">Return to home page</a>
22:       <br>
23:       {% include 'footer.html' %}
24:    </body>
25: </html>
```

PennyAdmin10CsrfToken/login.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h1>Login to Penny</h1>
      <!-- Use post instead of get for security. -->
      <form action="/handlelogin" method="post">
         <input type="hidden" name="csrf_token"
            value="{{csrf_token()}}">

         <table>
            <tbody>
               <tr>
                  <td style="text-align:right">User name:</td>
                  <td><input type="text" name="username" autofocus
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  </td>
               </tr>
               <tr>
                  <td style="text-align:right">Password:</td>
                  <td><input type="password" name="password"
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  <td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" value="Go"></td>
               </tr>
            </tbody>
         </table>
      </form>
      <br>
      Don’t have an account? <a href="/signup">Sign up</a>
      <br>
      <br>
      <strong>{{msg}}</strong>
   </body>
</html>
```

PennyAdmin10CsrfToken/signup.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h1>Penny New User Signup</h1>
      <!-- Use post instead of get for security. -->
      <form action="/handlesignup" method="post">
         <input type="hidden" name="csrf_token"
            value="{{csrf_token()}}">

         <table>
            <tbody>
               <tr>
                  <td style="text-align:right">User name:</td>
                  <td><input type="text" name="username" autofocus
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  </td>
               </tr>
               <tr>
                  <td style="text-align:right">Password:</td>
                  <td><input type="password" name="password"
                     required pattern=".*\S.*"
                     title="At least one non-white-space char">
                  <td>
               </tr>
               <tr>
                  <td></td>
                  <td><input type="submit" value="Go"></td>
               </tr>
            </tbody>
         </table>
      </form>
      <br>
      <strong>{{error_msg}}</strong>
   </body>
</html>
```

PennyAdmin10CsrfToken/top.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# top.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import flask
import flask_wtf.csrf
import dotenv

app = flask.Flask('penny', template_folder='.')

dotenv.load_dotenv()
app.secret_key = os.environ['APP_SECRET_KEY']

flask_wtf.csrf.CSRFProtect(app)
```

PennyAdmin10CsrfToken/penny.py (Page 1 of 3)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# penny.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import flask
import database
import auth

from top import app

#-----------------------------------------------------------------------

@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def index():

    username = auth.authenticate()
    is_authorized = auth.is_authorized(username)

    html_code = flask.render_template('index.html', username=username,
        is_authorized=is_authorized)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/show', methods=['GET'])
def show():

    username = auth.authenticate()

    books = database.get_books()
    html_code = flask.render_template('show.html',
        username=username, books=books)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

def report_results(username, message1, message2):

    html_code = flask.render_template('reportresults.html',
        username=username, message1=message1, message2=message2)
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/add', methods=['GET'])
def add():

    username = auth.authenticate()
    if not auth.is_authorized(username):
        html_code = 'You are not authorized to add books.'
        response = flask.make_response(html_code)
        return response

    html_code = flask.render_template('add.html', username=username)

    response = flask.make_response(html_code)
    return response

```

PennyAdmin10CsrfToken/penny.py (Page 2 of 3)

```
 66: #----------------------------------------------------------------------
 67:
 68: @app.route('/handleadd', methods=['POST'])
 69: def handle_add():
 70:
 71:     username = auth.authenticate()
 72:     if not auth.is_authorized(username):
 73:         html_code = 'You are not authorized to add books.'
 74:         response = flask.make_response(html_code)
 75:         return response
 76:
 77:     isbn = flask.request.form.get('isbn')
 78:     if (isbn is None) or (isbn.strip() == ''):
 79:         return report_results(username, 'Missing ISBN', '')
 80:
 81:     author = flask.request.form.get('author')
 82:     if (author is None) or (author.strip() == ''):
 83:         return report_results(username, 'Missing author', '')
 84:
 85:     title = flask.request.form.get('title')
 86:     if (title is None) or (title.strip() == ''):
 87:         return report_results(username, 'Missing title', '')
 88:
 89:     isbn = isbn.strip()
 90:     author = author.strip()
 91:     title = title.strip()
 92:
 93:     successful = database.add_book(isbn, author, title)
 94:     if successful:
 95:         message1 = 'The addition was successful'
 96:         message2 = 'The database now contains a book with isbn ' + isbn
 97:         message2 += ' author ' + author + ' and title ' + title
 98:     else:
 99:         message1 = 'The addition was unsuccessful'
100:         message2 = 'A book with ISBN ' + isbn + ' already exists'
101:
102:     return report_results(username, message1, message2)
103:
104: #----------------------------------------------------------------------
105:
106: @app.route('/delete', methods=['GET'])
107: def delete():
108:
109:     username = auth.authenticate()
110:     if not auth.is_authorized(username):
111:         html_code = 'You are not authorized to delete books.'
112:         response = flask.make_response(html_code)
113:         return response
114:
115:     html_code = flask.render_template('delete.html', username=username)
116:
117:     response = flask.make_response(html_code)
118:     return response
119:
120: #----------------------------------------------------------------------
121:
122: @app.route('/handledelete', methods=['POST'])
123: def handle_delete():
124:
125:     username = auth.authenticate()
126:     if not auth.is_authorized(username):
127:         html_code = 'You are not authorized to delete books.'
128:         response = flask.make_response(html_code)
129:         return response
130:
```

PennyAdmin10CsrfToken/penny.py (Page 3 of 3)

```
131:     isbn = flask.request.form.get('isbn')
132:     if (isbn is None) or (isbn.strip() == ''):
133:         return report_results(username, 'Missing ISBN', '')
134:
135:     isbn = isbn.strip()
136:
137:     database.delete_book(isbn)
138:
139:     message1 = 'The deletion was successful'
140:     message2 = 'The database now does not contain a book with ISBN '
141:     message2 += isbn
142:
143:     return report_results(username, message1, message2)
```

PennyAdmin10CsrfToken/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#----------------------------------------------------------------------

import flask
import database

from top import app

#----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_password = database.get_password(username)
    if stored_password is None:
        return False
    return password == stored_password

#----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''

    html = flask.render_template('login.html', msg=msg)

    response = flask.make_response(html)
    return response

#----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
    return response

#----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response
```

PennyAdmin10CsrfToken/auth.py (Page 2 of 2)

```

#----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''

    html_code = flask.render_template('signup.html',
        error_msg=error_msg)

    response = flask.make_response(html_code)
    return response

#----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))
    successful = database.add_user(username, password)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))

    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))
    # return flask.redirect(flask.url_for('login'))

#----------------------------------------------------------------------

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
    return username

#----------------------------------------------------------------------

def is_authorized(username):

    return database.is_authorized(username)
```

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

csrfattack.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:     <hr>Hello, and welcome to <strong>PennyAdmin</strong><hr>
 8:       <h1>Search for a Book by ISBN</h1>
 9:       <form action="http://localhost:55555/handledelete" method="post">
10:          Enter an ISBN:
11:          <input type="text" name="isbn" autofocus>
12:          <input type="submit" value="Go">
13:       </form>
14:       <br>
15:       <br>
16:       <a href="http://localhost:55555/index">Return to home page</a>
17:       <br>
18:       <hr>
19:       <a href="http://localhost:55555/logout">Log out</a>
20:       of the application</br>
21:       <hr>
22:       Created by <a href="https://www.cs.princeton.edu/~rdondero">
23:       Bob Dondero</a>
24:       <hr>
25:    </body>
26: </html>
```

PennyAdmin09CsrfToken/add.html (Page 1 of 1)

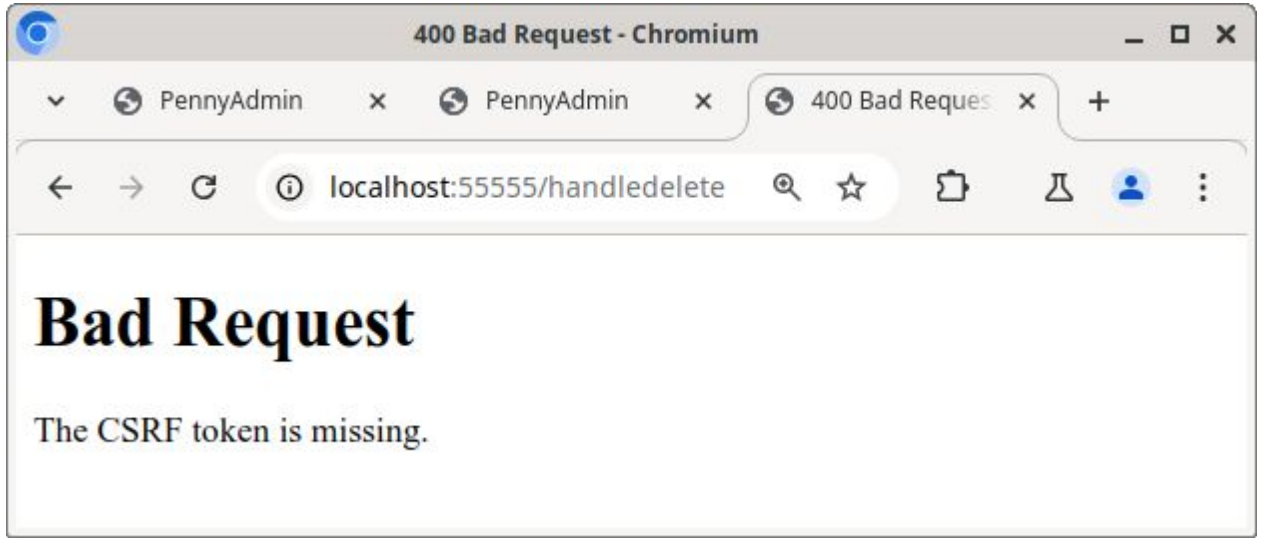
```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       {% include 'header.html' %}
 8:       <h1>Add a Book</h1>
 9:       <form action="/handleadd" method="post">
10:          <input type="hidden" name="csrf_token" value="{{csrf_token}}">
11:
12:          Enter an ISBN:
13:          <input type="text" name="isbn" autofocus
14:             required pattern=".*\S.*"
15:             title="At least one non-white-space char">
16:          <br>
17:
18:          Enter an author:
19:          <input type="text" name="author"
20:             required pattern=".*\S.*"
21:             title="At least one non-white-space char">
22:          <br>
23:
24:          Enter a title:
25:          <input type="text" name="title"
26:             required pattern=".*\S.*"
27:             title="At least one non-white-space char">
28:          <br>
29:
30:          <input type="submit" value="Go">
31:       </form>
32:       <br>
33:       <a href="/index">Return to home page</a>
34:       <br>
35:       {% include 'footer.html' %}
36:    </body>
37: </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**

PennyAdmin11Hash/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');

DROP TABLE IF EXISTS users;
CREATE TABLE users (username TEXT PRIMARY KEY, password TEXT);
INSERT INTO users (username, password) VALUES ('rdondero',
   'cd2eb0837c9b4c962c22d2ff8b5441b7b45805887f051d39bf133b583baf6860');
INSERT INTO users (username, password) VALUES ('bwk',
   'f2afd1cacb5441a5e65a7a460a5f9898b7b98b08aa6323a2e53c8b9a9686cd86');
INSERT INTO users (username, password) VALUES ('rs',
   '17f165d5a5ba695f27c023a83aa2b3463e23810e360b7517127e90161eebabda');

DROP TABLE IF EXISTS authorizedusers;
CREATE TABLE authorizedusers (username TEXT PRIMARY KEY);
INSERT INTO authorizedusers (username) VALUES ('rdondero');
INSERT INTO authorizedusers (username) VALUES ('bwk');
```

blank (Page 1 of 1)

This page is intentionally blank.

PennyAdmin11Hash/auth.py (Page 1 of 2)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import hashlib
import flask
import database

from top import app

#-----------------------------------------------------------------------

def _hash(password):

    hash_code = hashlib.sha256()
    hash_code.update(password.encode('ascii'))
    hash_code = hash_code.hexdigest()
    return hash_code

#-----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_hash_code = database.get_password(username)
    if stored_hash_code is None:
        return False
    hash_code = _hash(password)
    return hash_code == stored_hash_code

#-----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''

    html = flask.render_template('login.html', msg=msg)

    response = flask.make_response(html)
    return response

#-----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
```

PennyAdmin11Hash/auth.py (Page 2 of 2)

```python
    return response

#-----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''

    html_code = flask.render_template('signup.html',
        error_msg=error_msg)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))

    hash_code = _hash(password)
    successful = database.add_user(username, hash_code)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))

    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))

#-----------------------------------------------------------------------

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
    return username
```

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

CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. - Chromium

CrackStation - Online Passwor

crackstation.net

CrackStation

Defuse.ca · Twitt

CrackStation » Password Hashing Security » Defuse Security »

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
cd2eb0837c9b4c962c22d2ff8b5441b7b45805887f051d39bf133b583baf
6860
```

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Data Storage Attacks

CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. - Chromium

CrackStation - Online Passwor

crackstation.net

CrackStation

Defuse.ca · Twitter

CrackStation » Password Hashing Security » Defuse Security »

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
cd2eb0837c9b4c962c22d2ff8b5441b7b45805887f051d39bf133b583baf
6860
```

I'm not a robot

reCAPTCHA
Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
cd2eb0837c9b4c962c22d2ff8b5441b7b45805887f051d39bf133b583baf6860	sha256	xxx

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

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 (malevolent) list of hash codes

Data Storage Attacks

- Q: What should the salt string be?
- A: Any reasonably random string

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**

PennyAdmin12SaltHash/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;
CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');

DROP TABLE IF EXISTS users;
CREATE TABLE users (username TEXT PRIMARY KEY, password TEXT);
INSERT INTO users (username, password) VALUES ('rdondero',
   'pbkdf2:sha256:600000$UOVCfeB4bjAW4nYx$0641776b12a4054fe5fb72eb4f9bbf
73c4266099de5ec01f09c325ed56746c3a');
INSERT INTO users (username, password) VALUES ('bwk',
   'pbkdf2:sha256:600000$fQwzwUw8ZCPET43r$2390d394f64f239e5bc765f4163d49
d40b97601fae4d6ae0830c52296438e6ae');
INSERT INTO users (username, password) VALUES ('rs',
   'pbkdf2:sha256:600000$xbvn5wij1R8eJ5Mq$5bb39b39b45d65628091f4ba5fabda
99b30e9a60c447818db626f35f1dffdf9f');

DROP TABLE IF EXISTS authorizedusers;
CREATE TABLE authorizedusers (username TEXT PRIMARY KEY);
INSERT INTO authorizedusers (username) VALUES ('rdondero');
INSERT INTO authorizedusers (username) VALUES ('bwk');
```

blank (Page 1 of 1)

This page is intentionally blank.

PennyAdmin12SaltHash/auth.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import werkzeug.security
import flask
import database

from top import app

#-----------------------------------------------------------------------

def _valid_username_and_password(username, password):

    stored_hash_code = database.get_password(username)
    if stored_hash_code is None:
        return False
    return werkzeug.security.check_password_hash(
        stored_hash_code, password)

#-----------------------------------------------------------------------

@app.route('/login', methods=['GET'])
def login():

    msg = flask.request.args.get('msg')
    if msg is None:
        msg = ''

    html = flask.render_template('login.html', msg=msg)

    response = flask.make_response(html)
    return response

#-----------------------------------------------------------------------

@app.route('/handlelogin', methods=['POST'])
def handle_login():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    if not _valid_username_and_password(username, password):
        return flask.redirect(
            flask.url_for('login', msg='Wrong username or password'))
    original_url = flask.session.get('original_url', '/index')
    response = flask.redirect(original_url)
    flask.session['username'] = username
    return response

#-----------------------------------------------------------------------

@app.route('/logout', methods=['GET'])
def logout():

    flask.session.clear()
    html_code = flask.render_template('loggedout.html')
```

PennyAdmin12SaltHash/auth.py (Page 2 of 2)

```
    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/signup', methods=['GET'])
def signup():

    error_msg = flask.request.args.get('error_msg')
    if error_msg is None:
        error_msg = ''

    html_code = flask.render_template('signup.html',
        error_msg=error_msg)

    response = flask.make_response(html_code)
    return response

#-----------------------------------------------------------------------

@app.route('/handlesignup', methods=['POST'])
def handle_signup():

    username = flask.request.form.get('username')
    password = flask.request.form.get('password')
    if (username is None) or (username.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid username'))
    if (password is None) or (password.strip() == ''):
        return flask.redirect(
            flask.url_for('signup', error_msg='Invalid password'))

    hash_code = werkzeug.security.generate_password_hash(
        password, 'pbkdf2')
    successful = database.add_user(username, hash_code)
    if not successful:
        return flask.redirect(
            flask.url_for('signup', error_msg='Duplicate username'))

    return flask.redirect(
        flask.url_for('login', msg='You now are signed up.'))

#-----------------------------------------------------------------------

def authenticate():

    username = flask.session.get('username')
    if username is None:
        response = flask.redirect(flask.url_for('login'))
        flask.session['original_url'] = flask.request.url
        flask.abort(response)
    return username
```

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