Security Issues in Web Programming (Part 2)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

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

Agenda

- **Authentication & authorization**
- Cookie forgery attacks

Authentication & Authorization

- **Problem**:
  - PennyAdmin doesn't do **authentication** of its users
    - It doesn't know who its users are
  - PennyAdmin doesn't do **authorization** of its users
    - Only certain users – the Penny bookstore owners – should be allowed to change the PennyAdmin DB

Authentication & Authorization

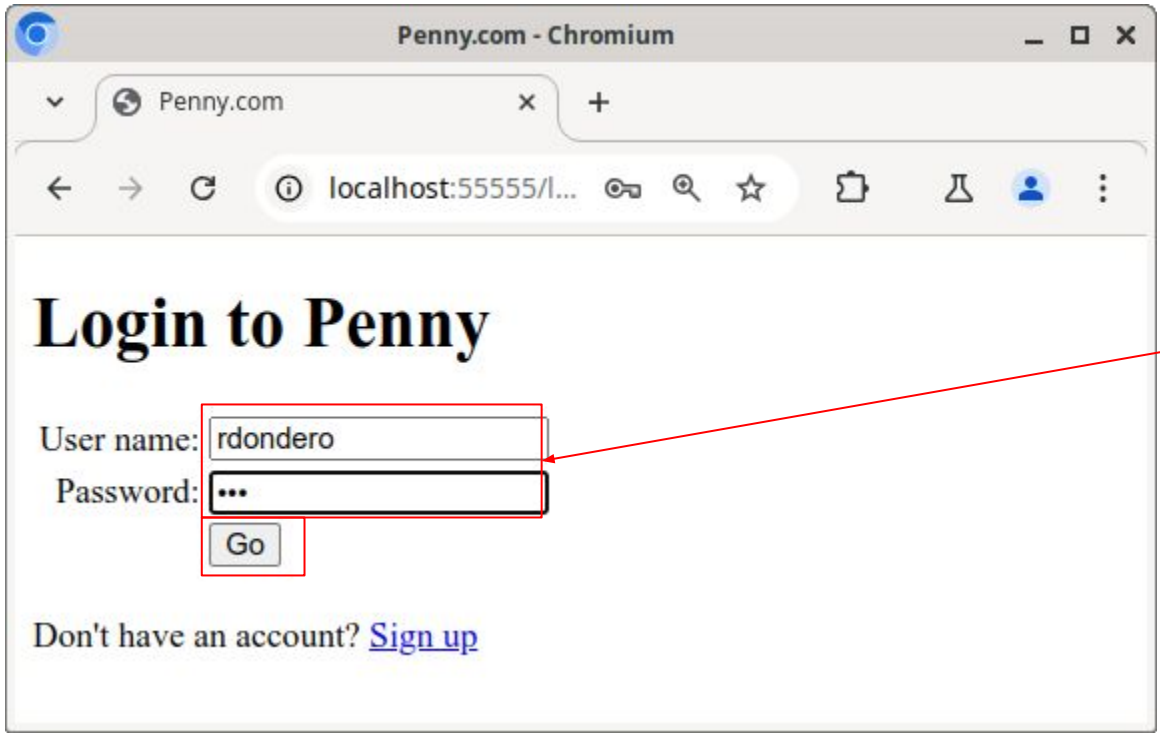
- ***Authenticate***
  - Make sure the user is *authentic*
  - Make sure the user is who he/she claims to be
- ***Authorize***
  - Having authenticated the user, make sure the user has *authority* to use the app in the way the user wants

Authentication & Authorization

- Suppose this is what we want…
  - **Show** books:
    - User must be **authenticated**
  - **Add** & **delete** books:
    - User must be **authenticated**
    - User must be **authorized**
      - Only rdondero and bwk have authority

Authentication & Authorization

- See **PennyAdmin06Auth** app

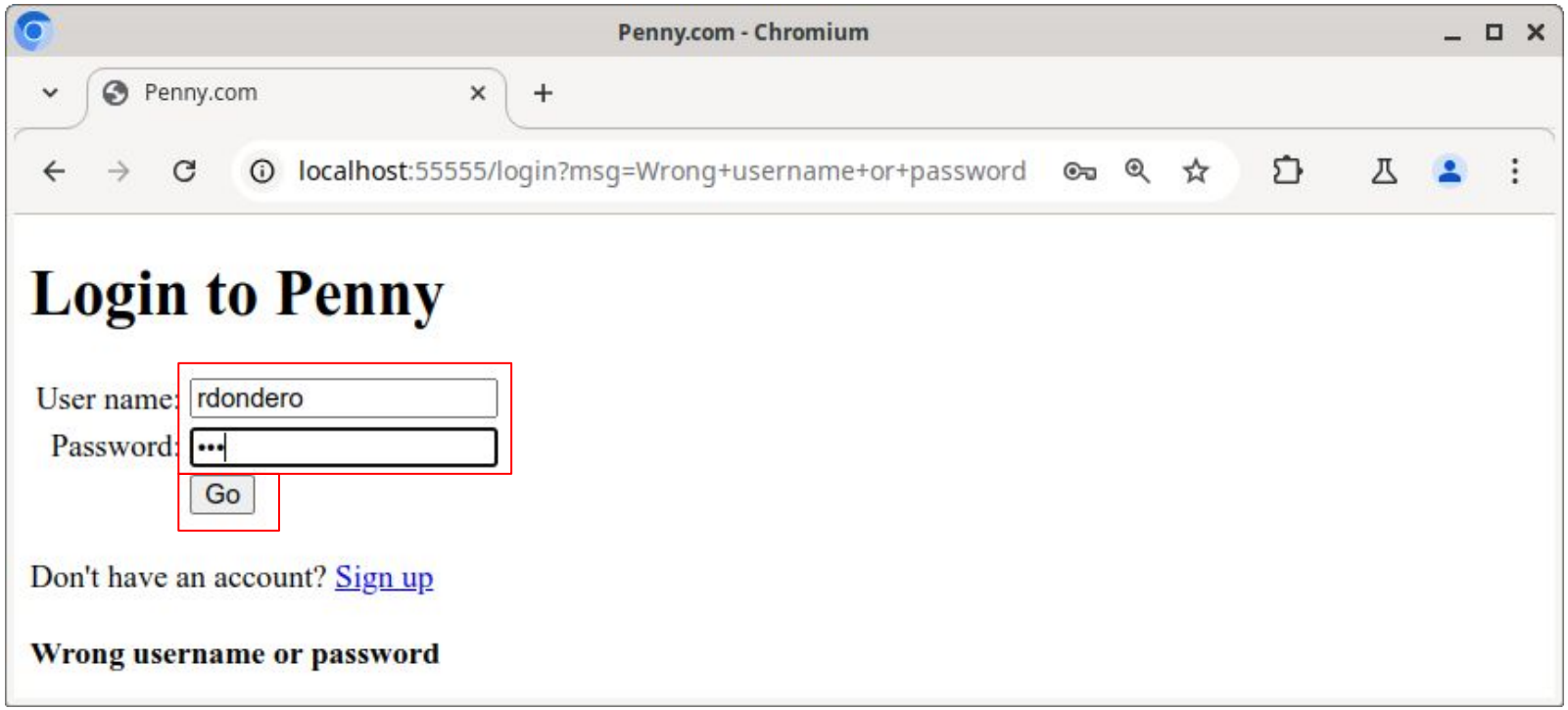


Login page

Wrong password

Authentication & Authorization

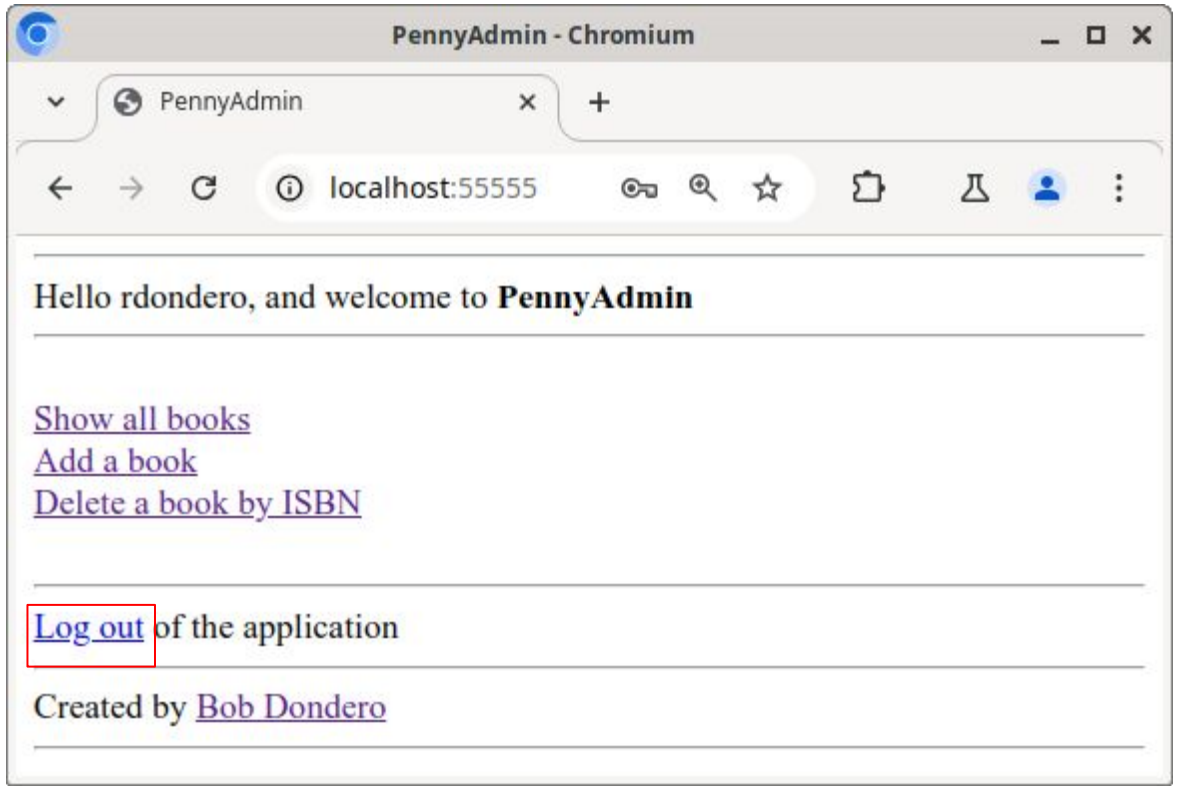
- See **<u>PennyAdmin06Auth</u>** app



Login page

Authentication & Authorization

- See **PennyAdmin06Auth** app



Index page

Username in header

Log out link in footer

Authentication & Authorization

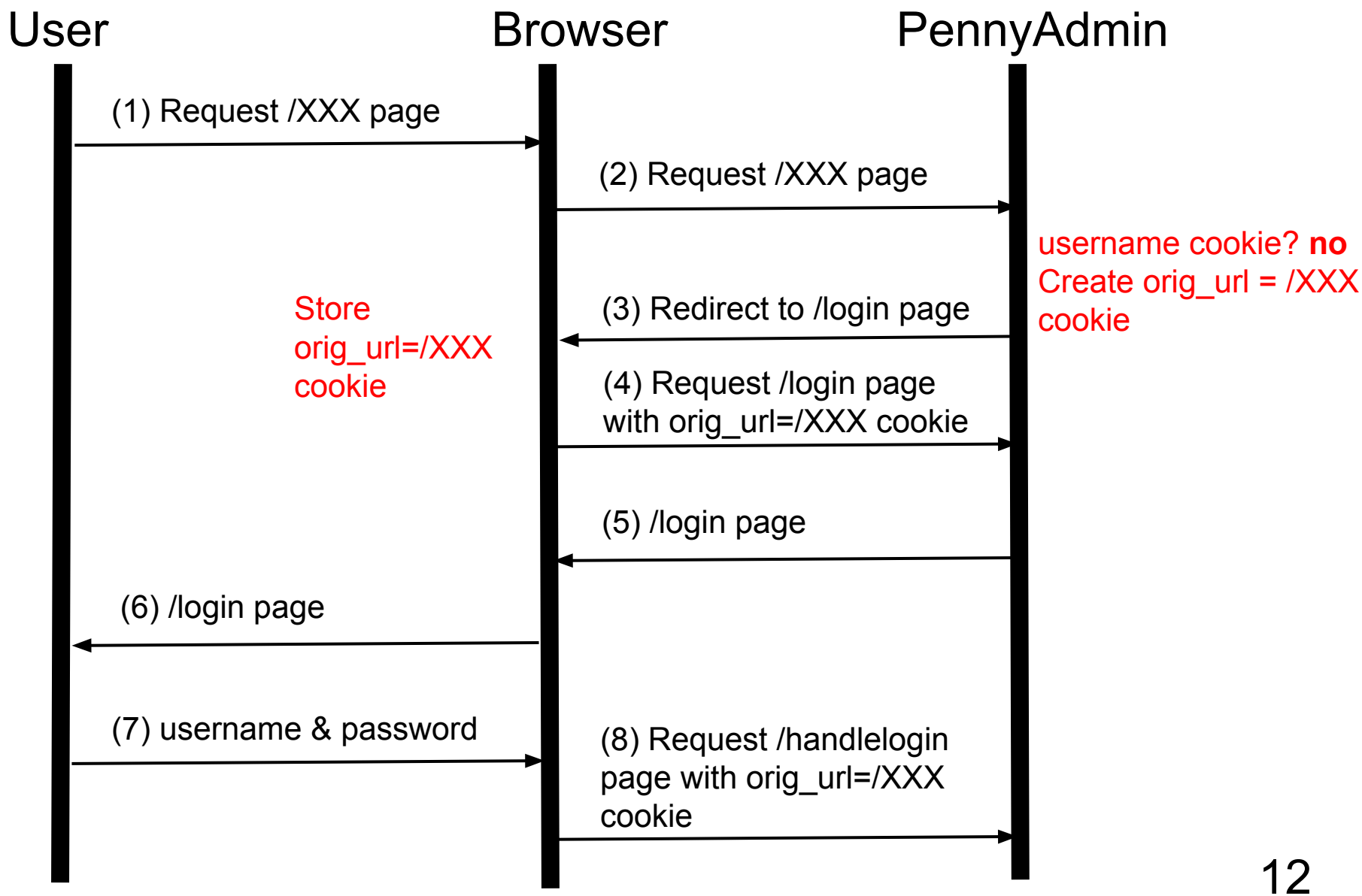
- See **<u>PennyAdmin06Auth</u>** app

Loggedout page

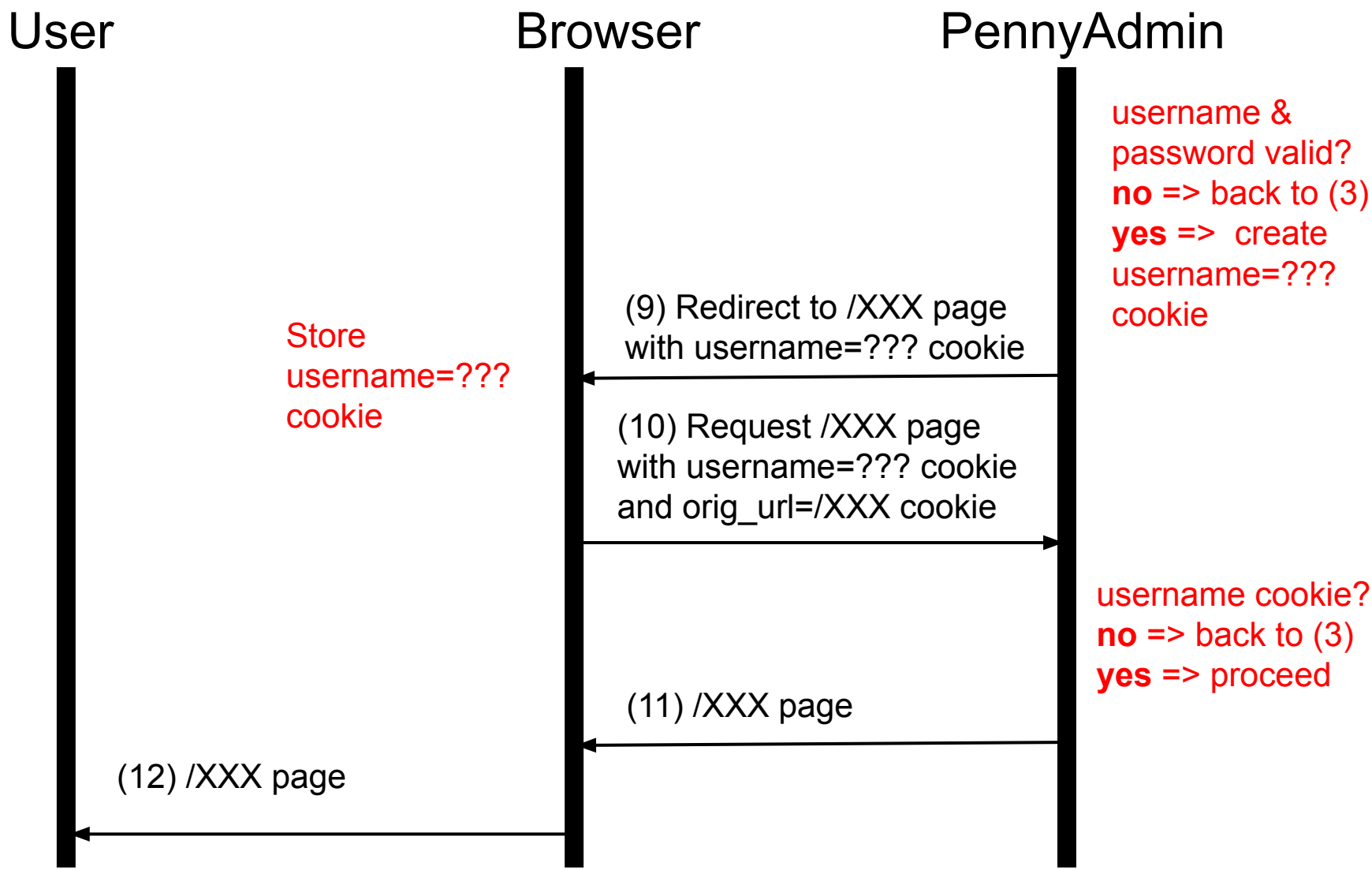
Authentication & Authorization

- See **<u>PennyAdmin06Auth</u>** app
  - The login flow…
  - (New user flow omitted)

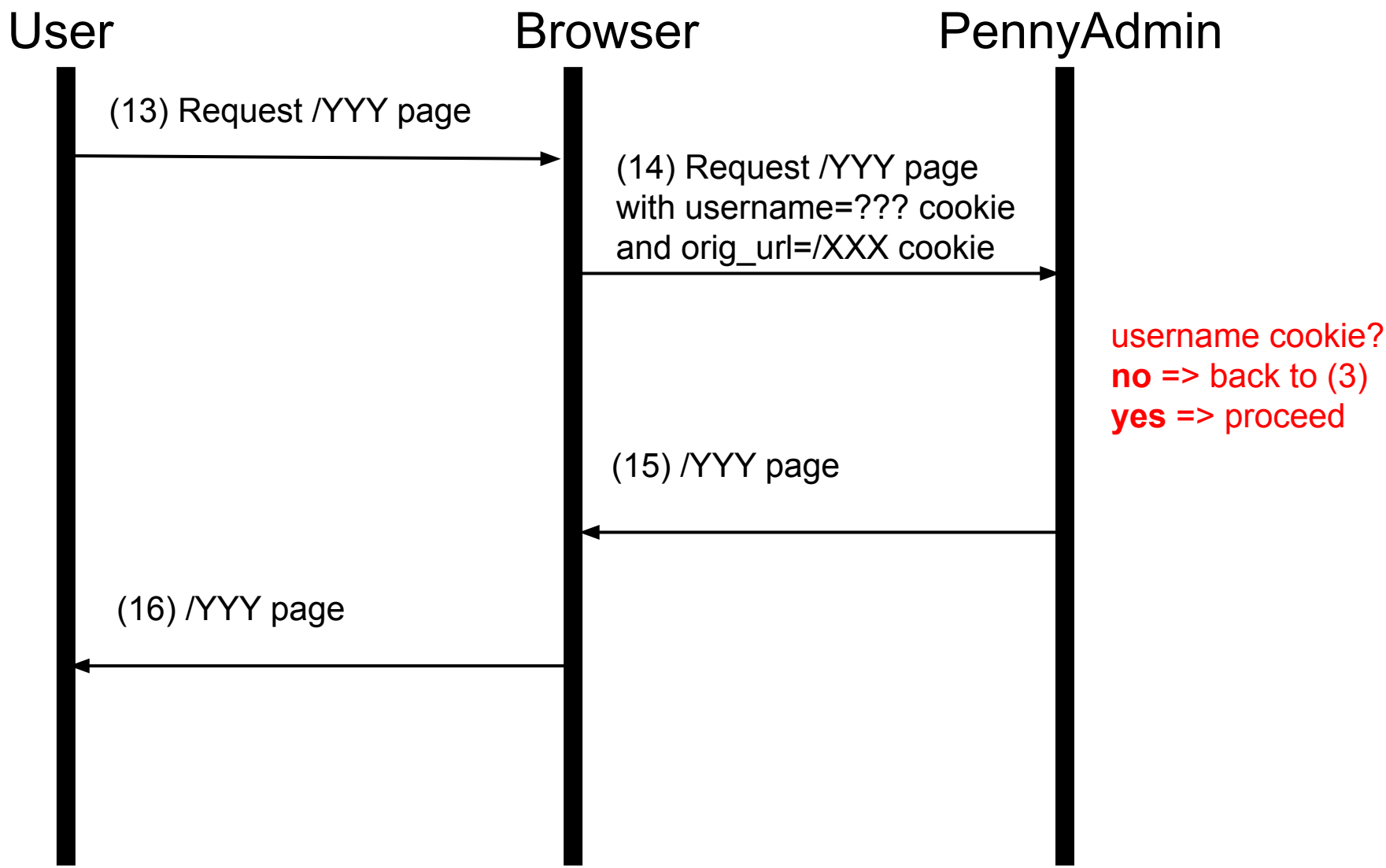
Authentication & Authorization



Authentication & Authorization



Authentication & Authorization



Authentication & Authorization

- See **PennyAdmin06Auth** 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**

PennyAdmin06Auth/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', 'xxx');
INSERT INTO users (username, password) VALUES ('bwk', 'yyy');
INSERT INTO users (username, password) VALUES ('rs', 'zzz');

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

PennyAdmin06Auth/database.py (Page 1 of 3)

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

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

import os
import sqlalchemy
import dotenv

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

dotenv.load_dotenv()
_database_url = os.getenv('DATABASE_URL', 'sqlite:///penny.sqlite')
_database_url = _database_url.replace('postgres://', 'postgresql://')

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

Base = sqlalchemy.orm.declarative_base()

class Book (Base):
    __tablename__ = 'books'
    isbn = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    author = sqlalchemy.Column(sqlalchemy.String)
    title = sqlalchemy.Column(sqlalchemy.String)

class User (Base):
    __tablename__ = 'users'
    username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)
    password = sqlalchemy.Column(sqlalchemy.String)

class AuthorizedUser (Base):
    __tablename__ = 'authorizedusers'
    username = sqlalchemy.Column(sqlalchemy.String, primary_key=True)

_engine = sqlalchemy.create_engine(_database_url)

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

def get_books():

    books = []

    with sqlalchemy.orm.Session(_engine) as session:
        query = session.query(Book)
        table = query.all()
        for row in table:
            book = {'isbn': row.isbn, 'author': row.author,
                'title': row.title}
            books.append(book)

    return books

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

def add_book(isbn, author, title):

    with sqlalchemy.orm.Session(_engine) as session:
        row = Book(isbn=isbn, author=author, title=title)
        session.add(row)
        try:
            session.commit()
            return True
        except sqlalchemy.exc.IntegrityError:
```

PennyAdmin06Auth/database.py (Page 2 of 3)

```
 66:             return False
 67:
 68: #----------------------------------------------------------------------
 69:
 70: def delete_book(isbn):
 71:
 72:     with sqlalchemy.orm.Session(_engine) as session:
 73:         session.query(Book).filter(Book.isbn==isbn).delete()
 74:         session.commit()
 75:
 76: #----------------------------------------------------------------------
 77:
 78: def get_password(username):
 79:
 80:     with sqlalchemy.orm.Session(_engine) as session:
 81:         query = session.query(User).filter(User.username==username)
 82:         try:
 83:             row = query.one()
 84:             return row.password
 85:         except sqlalchemy.exc.NoResultFound:
 86:             return None
 87:
 88: #----------------------------------------------------------------------
 89:
 90: def is_authorized(username):
 91:
 92:     with sqlalchemy.orm.Session(_engine) as session:
 93:         query = session.query(AuthorizedUser) \
 94:             .filter(AuthorizedUser.username==username)
 95:         try:
 96:             query.one()
 97:             return True
 98:         except sqlalchemy.exc.NoResultFound:
 99:             return False
100:
101: #----------------------------------------------------------------------
102:
103: def add_user(username, password):
104:
105:     with sqlalchemy.orm.Session(_engine) as session:
106:         row = User(username=username, password=password)
107:         session.add(row)
108:         try:
109:             session.commit()
110:             return True
111:         except sqlalchemy.exc.IntegrityError:
112:             return False
113:
114: #----------------------------------------------------------------------
115:
116: # For testing:
117:
118: def _write_books(books):
119:     for book in books:
120:         print('%s | %s | %s' % (book['isbn'], book['author'],
121:             book['title']))
122:
123: def _test():
124:     print('----------------------------------')
125:     print('Testing get_books()')
126:     print('----------------------------------')
127:     print()
128:     books = get_books()
129:     _write_books(books)
130:     print()
```

PennyAdmin06Auth/database.py (Page 3 of 3)

```
131:
132:     print('----------------------------------')
133:     print('Testing add_book()')
134:     print('----------------------------------')
135:     print()
136:     successful = add_book('456', 'Kernighan', 'New Book')
137:     if successful:
138:         print('Add was successful')
139:         print()
140:         books = get_books()
141:         _write_books(books)
142:         print()
143:     else:
144:         print('Add was unsuccessful')
145:         print()
146:         _write_books(books)
147:         print()
148:     successful = add_book('456', 'Kernighan', 'New Book')
149:     if successful:
150:         print('Add was successful')
151:         print()
152:         books = get_books()
153:         _write_books(books)
154:         print()
155:     else:
156:         print('Add was unsuccessful')
157:         print()
158:         _write_books(books)
159:         print()
160:
161:     print('----------------------------------')
162:     print('Testing delete_book()')
163:     print('----------------------------------')
164:     print()
165:     delete_book('456')
166:     books = get_books()
167:     _write_books(books)
168:     print()
169:     delete_book('456')
170:     books = get_books()
171:     _write_books(books)
172:     print()
173:
174:     print('----------------------------------')
175:     print('Testing get_password()')
176:     print('----------------------------------')
177:     print()
178:     password = get_password('rdondero')
179:     print(password)
180:     password = get_password('rdondero2')
181:     print(password)
182:     print()
183:
184: if __name__ == '__main__':
185:     _test()
```

PennyAdmin06Auth/header.html (Page 1 of 1)

```
1: <hr>Hello {{username}}, and welcome to <strong>PennyAdmin</strong><hr>
```

PennyAdmin06Auth/footer.html (Page 1 of 1)

```
1: <hr>
2: <a href="logout">Log out</a> of the application</br>
3: <hr>
4: Created by <a href="https://www.cs.princeton.edu/~rdondero">
5: Bob Dondero</a>
6: <hr>
```

PennyAdmin06Auth/index.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:       <br>
 9:       <a href="/show">Show all books</a><br>
10:       {% if is_authorized: %}
11:          <a href="/add">Add a book</a><br>
12:          <a href="/delete">Delete a book by ISBN</a><br>
13:       {% endif %}
14:       <br>
15:       {% include 'footer.html' %}
16:    </body>
17: </html>
```

PennyAdmin06Auth/login.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>Penny.com</title>
 5:    </head>
 6:    <body>
 7:       <h1>Login to Penny</h1>
 8:       <!-- Use post instead of get for security. -->
 9:       <form action="/handlelogin" method="post">
10:          <table>
11:             <tbody>
12:                <tr>
13:                   <td style="text-align:right">User name:</td>
14:                   <td><input type="text" name="username" autofocus
15:                      required pattern=".*\S.*"
16:                      title="At least one non-white-space char">
17:                   </td>
18:                </tr>
19:                <tr>
20:                   <td style="text-align:right">Password:</td>
21:                   <td><input type="password" name="password"
22:                      required pattern=".*\S.*"
23:                      title="At least one non-white-space char">
24:                   <td>
25:                </tr>
26:                <tr>
27:                   <td></td>
28:                   <td><input type="submit" value="Go"></td>
29:                </tr>
30:             </tbody>
31:          </table>
32:       </form>
33:       <br>
34:       Don't have an account? <a href="/signup">Sign up</a>
35:       <br>
36:       <br>
37:       <strong>{{msg}}</strong>
38:    </body>
39: </html>
```

PennyAdmin06Auth/signup.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>Penny.com</title>
 5:    </head>
 6:    <body>
 7:       <h1>Penny New User Signup</h1>
 8:       <!-- Use post instead of get for security. -->
 9:       <form action="/handlesignup" method="post">
10:          <table>
11:             <tbody>
12:                <tr>
13:                   <td style="text-align:right">User name:</td>
14:                   <td><input type="text" name="username" autofocus
15:                      required pattern=".*\S.*"
16:                      title="At least one non-white-space char">
17:                   </td>
18:                </tr>
19:                <tr>
20:                   <td style="text-align:right">Password:</td>
21:                   <td><input type="password" name="password"
22:                      required pattern=".*\S.*"
23:                      title="At least one non-white-space char">
24:                   <td>
25:                </tr>
26:                <tr>
27:                   <td></td>
28:                   <td><input type="submit" value="Go"></td>
29:                </tr>
30:             </tbody>
31:          </table>
32:       </form>
33:       <br>
34:       <strong>{{error_msg}}</strong>
35:    </body>
36: </html>
```

PennyAdmin06Auth/loggedout.html (Page 1 of 1)

```
 1: <!DOCTYPE html>
 2: <html>
 3:    <head>
 4:       <title>PennyAdmin</title>
 5:    </head>
 6:    <body>
 7:       <h2>You are logged out of PennyAdmin</h2>
 8:       <a href="/index">Log back in</a>
 9:    </body>
10: </html>
```

PennyAdmin06Auth/top.py (Page 1 of 1)

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

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

import flask

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

PennyAdmin06Auth/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 = 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

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

#-----------------------------------------------------------------------
```

PennyAdmin06Auth/penny.py (Page 2 of 3)

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

PennyAdmin06Auth/penny.py (Page 3 of 3)

```
131:         return report_results(username, 'Missing ISBN', '')
132:
133:     isbn = isbn.strip()
134:
135:     database.delete_book(isbn)
136:
137:     message1 = 'The deletion was successful'
138:     message2 = 'The database now does not contain a book with ISBN '
139:     message2 += isbn
140:
141:     return report_results(username, message1, message2)
```

PennyAdmin06Auth/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.request.cookies.get('original_url', '/index')
    response = flask.redirect(original_url)
    response.set_cookie('username', username)
    return response

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

@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.
```

PennyAdmin06Auth/auth.py (Page 2 of 2)

```
    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():

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

```

Authentication & Authorization

- Realistic, but…
- Typical enhancements:
  - Password changing (easy)
  - Existing user un-registration (easy)
  - Email verification (hard)
  - Forgotten password recovery (hard)
  - Two-factor authentication (hard)
    - Email, phone call, text message, authenticator app

Agenda

- Authentication & authorization
- **Cookie forgery 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)

```
#!/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 as rdondero

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 as rdondero

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)

```
 1: #!/usr/bin/env python
 2:
 3: #-----------------------------------------------------------------------
 4: # auth.py
 5: # Author: Bob Dondero
 6: #-----------------------------------------------------------------------
 7:
 8: import os
 9: import cryptocode
10: import flask
11: import dotenv
12: import database
13:
14: from top import app
15:
16: #-----------------------------------------------------------------------
17:
18: dotenv.load_dotenv()
19: secret_key = os.environ['APP_SECRET_KEY']
20:
21: #-----------------------------------------------------------------------
22:
23: def _valid_username_and_password(username, password):
24:
25:     stored_password = database.get_password(username)
26:     if stored_password is None:
27:         return False
28:     return password == stored_password
29:
30: #-----------------------------------------------------------------------
31:
32: @app.route('/login', methods=['GET'])
33: def login():
34:
35:     msg = flask.request.args.get('msg')
36:     if msg is None:
37:         msg = ''
38:     html = flask.render_template('login.html', msg=msg)
39:     response = flask.make_response(html)
40:     return response
41:
42: #-----------------------------------------------------------------------
43:
44: @app.route('/handlelogin', methods=['POST'])
45: def handle_login():
46:
47:     username = flask.request.form.get('username')
48:     password = flask.request.form.get('password')
49:     if (username is None) or (username.strip() == ''):
50:         return flask.redirect(
51:             flask.url_for('login', msg='Wrong username or password'))
52:     if (password is None) or (password.strip() == ''):
53:         return flask.redirect(
54:             flask.url_for('login', msg='Wrong username or password'))
55:     if not _valid_username_and_password(username, password):
56:         return flask.redirect(
57:             flask.url_for('login', msg='Wrong username or password'))
58:     original_url = flask.request.cookies.get('original_url', '/index')
59:     response = flask.redirect(original_url)
60:     encrypted_username = cryptocode.encrypt(username, secret_key)
61:     response.set_cookie('username', encrypted_username)
62:     return response
63:
64: #-----------------------------------------------------------------------
65:
```

PennyAdmin07Encrypt/auth.py (Page 2 of 2)

```
66: @app.route('/logout', methods=['GET'])
67: def logout():
68:
69:     html_code = flask.render_template('loggedout.html')
70:     response = flask.make_response(html_code)
71:
72:     # Delete cookies in the browser by setting them to expire at
73:     # a time that is in the past.
74:     response.set_cookie('username', '', expires=0)
75:     response.set_cookie('original_url', '', expires=0)
76:
77:     return response
78:
79: #-----------------------------------------------------------------------
80:
81: @app.route('/signup', methods=['GET'])
82: def signup():
83:
84:     error_msg = flask.request.args.get('error_msg')
85:     if error_msg is None:
86:         error_msg = ''
87:     html_code = flask.render_template('signup.html',
88:         error_msg=error_msg)
89:     response = flask.make_response(html_code)
90:     return response
91:
92: #-----------------------------------------------------------------------
93:
94: @app.route('/handlesignup', methods=['POST'])
95: def handle_signup():
96:
97:     username = flask.request.form.get('username')
98:     password = flask.request.form.get('password')
99:     if (username is None) or (username.strip() == ''):
100:         return flask.redirect(
101:             flask.url_for('signup', error_msg='Invalid username'))
102:     if (password is None) or (password.strip() == ''):
103:         return flask.redirect(
104:             flask.url_for('signup', error_msg='Invalid password'))
105:     successful = database.add_user(username, password)
106:     if not successful:
107:         return flask.redirect(
108:             flask.url_for('signup', error_msg='Duplicate username'))
109:     return flask.redirect(
110:         flask.url_for('login', msg='You now are signed up.'))
111:
112: #-----------------------------------------------------------------------
113:
114: def authenticate():
115:
116:     encrypted_username = flask.request.cookies.get('username')
117:
118:     if encrypted_username is None:
119:         response = flask.redirect(flask.url_for('login'))
120:         response.set_cookie('original_url', flask.request.url)
121:         flask.abort(response)
122:
123:     username = cryptocode.decrypt(encrypted_username, secret_key)
124:     if not username:
125:         response = flask.redirect(flask.url_for('login'))
126:         response.set_cookie('original_url', flask.request.url)
127:         flask.abort(response)
128:
129:     return username
```

Cookie Forgery Attacks

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

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.
$
```

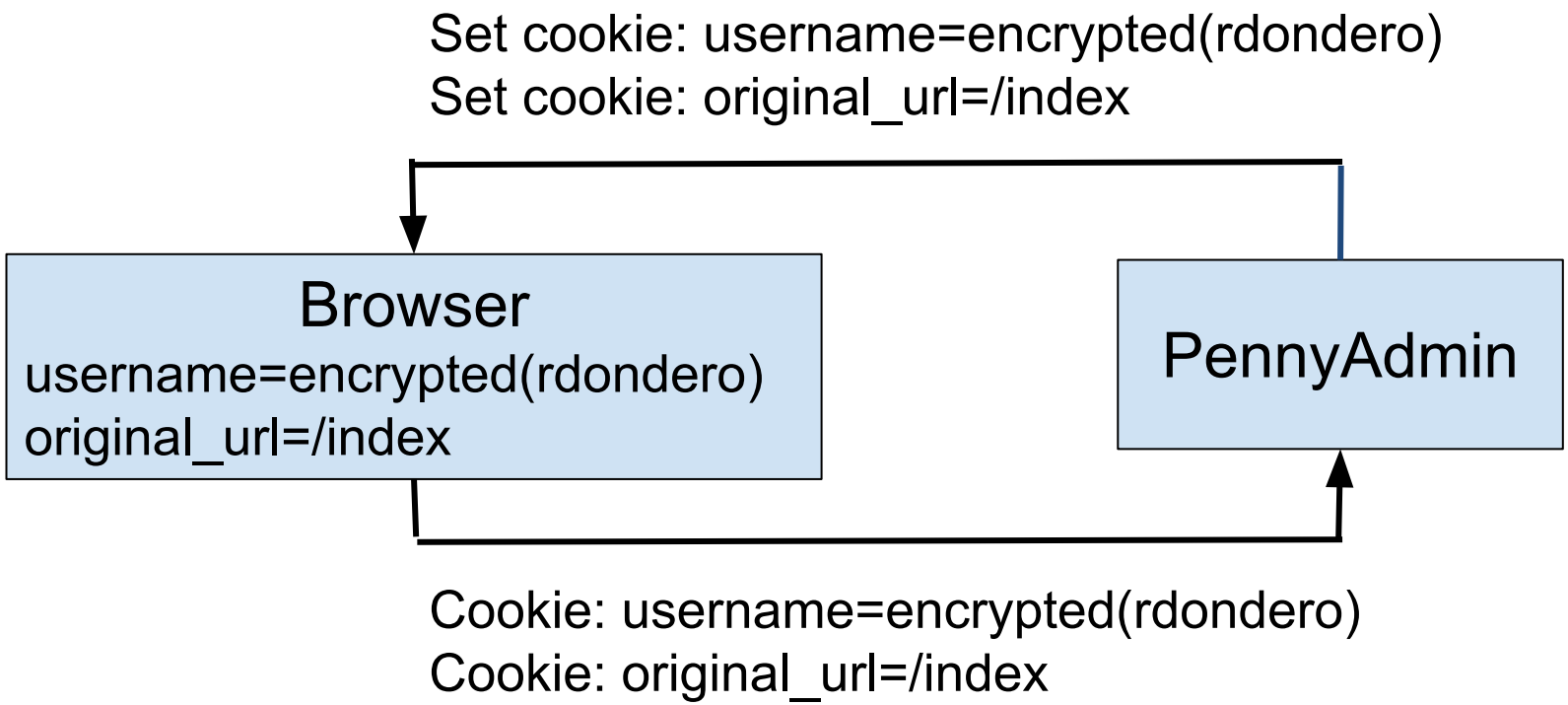
App rejects username, redirects to login page

Cookie Forgery Attacks

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

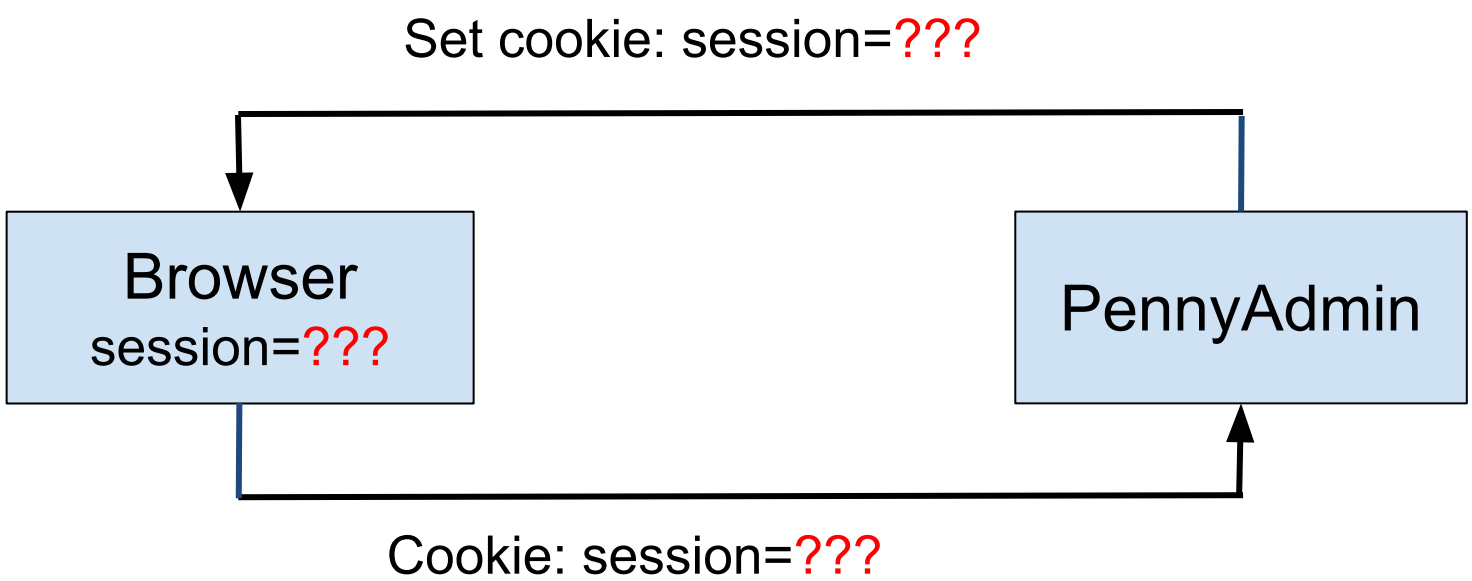
Cookie Forgery Attacks

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)

```
 1: #!/usr/bin/env python
 2:
 3: #----------------------------------------------------------------------
 4: # auth.py
 5: # Author: Bob Dondero
 6: #----------------------------------------------------------------------
 7:
 8: import flask
 9: import database
10:
11: from top import app
12:
13: #----------------------------------------------------------------------
14:
15: def _valid_username_and_password(username, password):
16:
17:     stored_password = database.get_password(username)
18:     if stored_password is None:
19:         return False
20:     return password == stored_password
21:
22: #----------------------------------------------------------------------
23:
24: @app.route('/login', methods=['GET'])
25: def login():
26:
27:     msg = flask.request.args.get('msg')
28:     if msg is None:
29:         msg = ''
30:
31:     html = flask.render_template('login.html', msg=msg)
32:     response = flask.make_response(html)
33:     return response
34:
35: #----------------------------------------------------------------------
36:
37: @app.route('/handlelogin', methods=['POST'])
38: def handle_login():
39:
40:     username = flask.request.form.get('username')
41:     password = flask.request.form.get('password')
42:     if (username is None) or (username.strip() == ''):
43:         return flask.redirect(
44:             flask.url_for('login', msg='Wrong username or password'))
45:     if (password is None) or (password.strip() == ''):
46:         return flask.redirect(
47:             flask.url_for('login', msg='Wrong username or password'))
48:     if not _valid_username_and_password(username, password):
49:         return flask.redirect(
50:             flask.url_for('login', msg='Wrong username or password'))
51:     original_url = flask.session.get('original_url', '/index')
52:     response = flask.redirect(original_url)
53:     flask.session['username'] = username
54:     return response
55:
56: #----------------------------------------------------------------------
57:
58: @app.route('/logout', methods=['GET'])
59: def logout():
60:
61:     flask.session.clear()
62:     html_code = flask.render_template('loggedout.html')
63:     response = flask.make_response(html_code)
64:     return response
65:
```

PennyAdmin08Session/auth.py (Page 2 of 2)

```
66: #----------------------------------------------------------------------
67:
68: @app.route('/signup', methods=['GET'])
69: def signup():
70:
71:     error_msg = flask.request.args.get('error_msg')
72:     if error_msg is None:
73:         error_msg = ''
74:
75:     html_code = flask.render_template('signup.html',
76:         error_msg=error_msg)
77:     response = flask.make_response(html_code)
78:     return response
79:
80: #----------------------------------------------------------------------
81:
82: @app.route('/handlesignup', methods=['POST'])
83: def handle_signup():
84:
85:     username = flask.request.form.get('username')
86:     password = flask.request.form.get('password')
87:     if (username is None) or (username.strip() == ''):
88:         return flask.redirect(
89:             flask.url_for('signup', error_msg='Invalid username'))
90:     if (password is None) or (password.strip() == ''):
91:         return flask.redirect(
92:             flask.url_for('signup', error_msg='Invalid password'))
93:     successful = database.add_user(username, password)
94:     if not successful:
95:         return flask.redirect(
96:             flask.url_for('signup', error_msg='Duplicate username'))
97:
98:     return flask.redirect(
99:         flask.url_for('login', msg='You now are signed up.'))
100:
101: #----------------------------------------------------------------------
102:
103: def authenticate():
104:
105:     username = flask.session.get('username')
106:     if username is None:
107:         response = flask.redirect(flask.url_for('login'))
108:         flask.session['original_url'] = flask.request.url
109:         flask.abort(response)
110:     return username
```

Cookie Forgery Attacks

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

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

Summary

- We have covered:
  - Authentication & authorization
  - Secret keys
  - Cookie forgery attacks