Security Issues in Web Programming (Part 3)

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

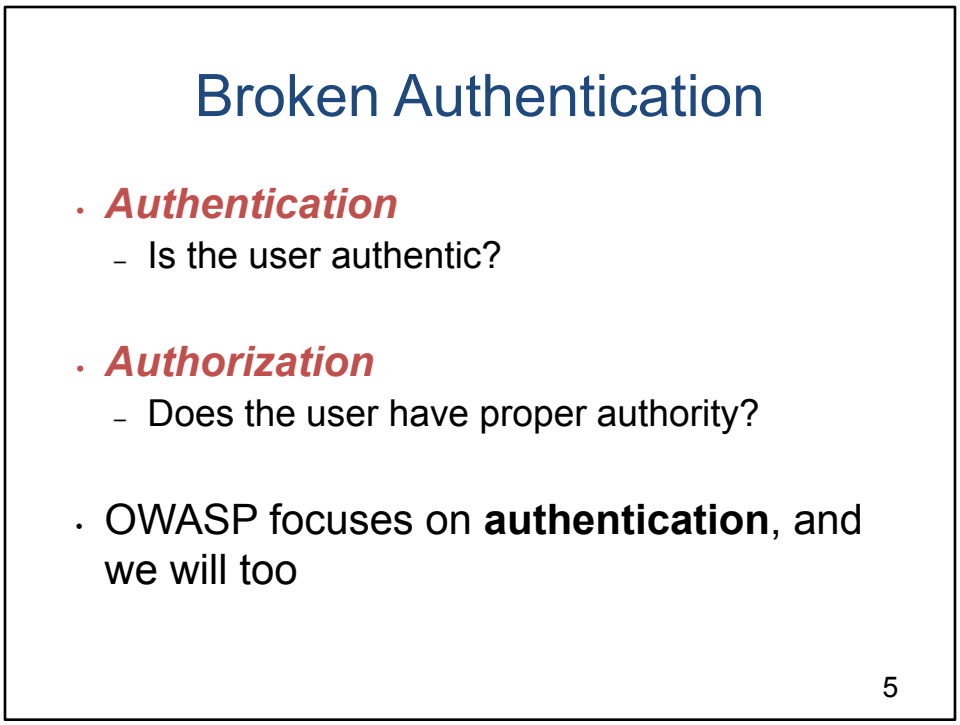
- **Broken authentication**
- Broken authentication: baseline example
- Broken authentication: basic access authentication
- Sessions
- Broken authentication: session-based authentication

Broken Authentication

Broken Authentication

“Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.”

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



Broken Authentication

Authentication
Is the user authentic?
Is the user who he/she says he/she is?
Authorization
Does the user have proper authority?
Does the user have permission to use the application in the manner he/she has requested?

Broken Authentication

- **Example risk:**
  - Attacker may gain access to app without proper authentication
- **Action item:**
  - Implement a vetted authentication approach…

Broken Authentication

- Approaches to **authentication**:
  - We'll study:
    - Basic access authentication
    - Session-based authentication
    - Central Authentication System (CAS)
    - Google authentication

7

Broken Authentication

[see slide]

Survey (show of hands)

> Which project teams are using CAS?
> Which teams have successfully implemented CAS?
> Who on those teams was directly involved in implementing CAS?
>
> The CAS part of today's class is not for you; it's for everybody else
> Please contribute!

Survey (show of hands)

> Which project teams are using Google authentication?
> Which teams have successfully implemented Google authentication?
> Who on those teams was directly involved in implementing Google authentication?
>
> The Google authentication part of this lecture is not for you; it's for everybody else
> Please contribute!

Agenda

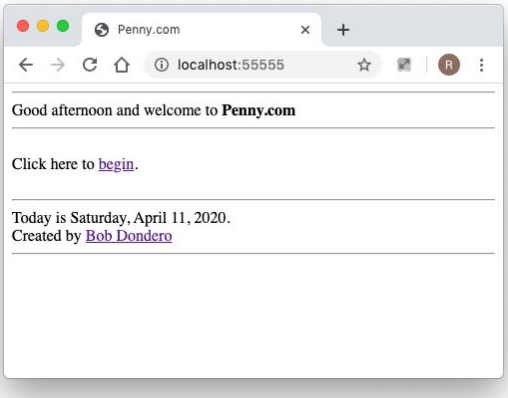
- Broken authentication
- **Broken authentication: baseline example**
- Broken authentication: basic access authentication
- Sessions
- Broken authentication: session-based authentication

Baseline Example

- Recall **PennyFlaskJinja** app
  - The job:
    - Three-page Penny app
    - Uses Flask and Jinja2
    - Stores state in cookies
    - Performs no authentication

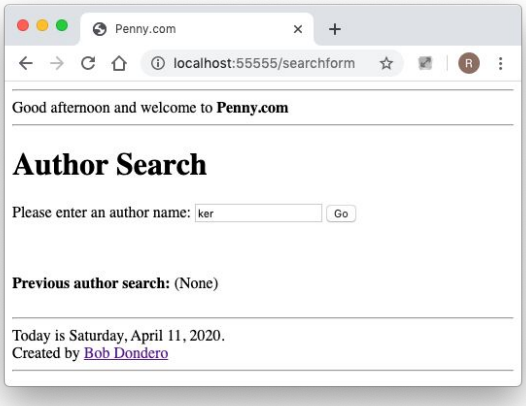
Baseline Example

- Recall **PennyFlaskJinja** app



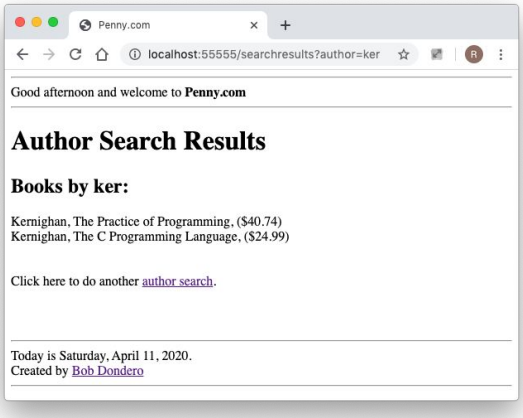
Baseline Example

- Recall **PennyFlaskJinja** app



Baseline Example

- Recall **PennyFlaskJinja** app



Baseline Example

- Recall **PennyFlaskJinja** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - header.html, footer.html
  - index.html, searchform.html, searchresults.html
  - **penny.py**

Agenda

- Broken authentication
- Broken authentication: baseline example
- **Broken authentication: basic access authentication**
- Sessions
- Broken authentication: session-based authentication

Basic Access Auth

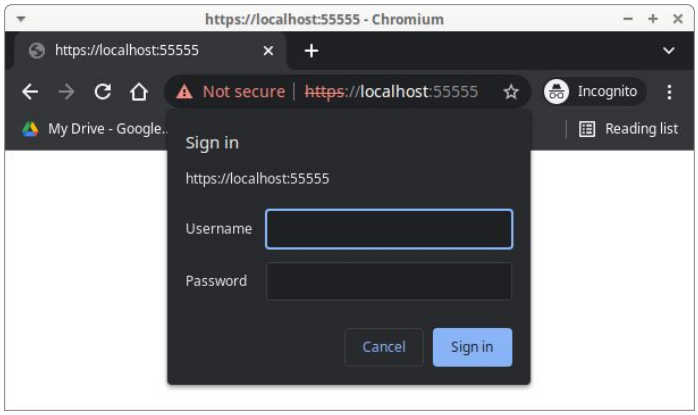
- ***Basic Access Authentication***
  - Authentication via HTTP headers

Basic Access Auth: Example

- See **PennyBasicAuth** app
  - The job:
    - Same as PennyFlaskJinja, except...
    - Uses basic access authentication

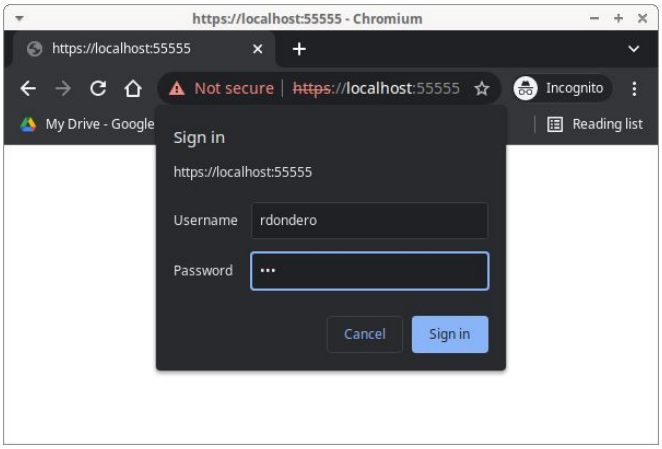
Basic Access Auth: Example

- See **PennyBasicAuth** app (cont.)



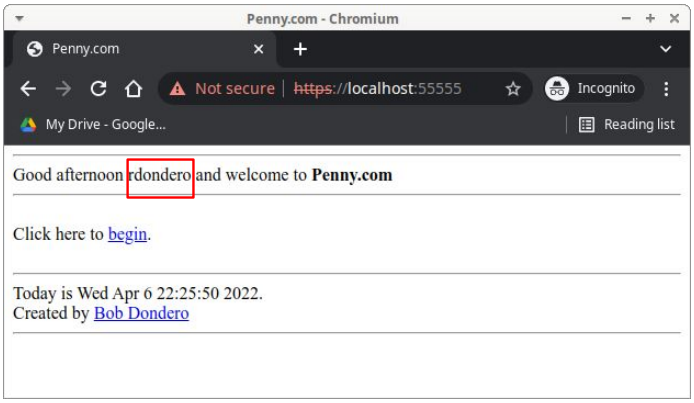
Basic Access Auth: Example

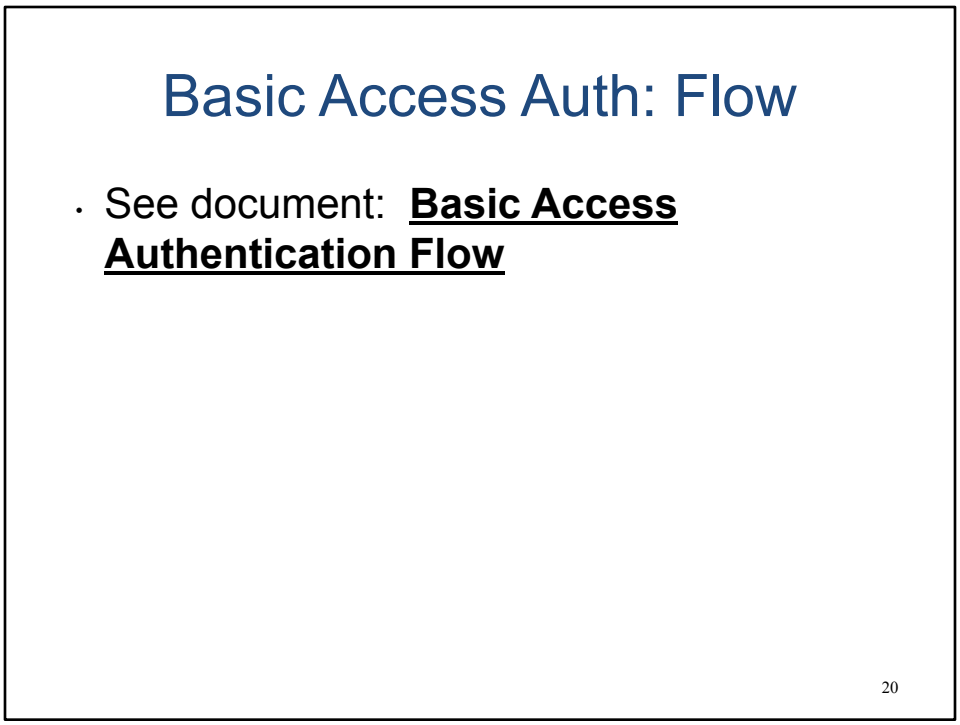
- See **PennyBasicAuth** app (cont.)



Basic Access Auth: Example

- See **PennyBasicAuth** app (cont.)





Basic Access Auth: Flow

What is the flow of information between the browser and the application when using basic access authentication?

Security programmers speak of the **protocol**, alias the **handshake**, alias the **dance**, alias the **flow**

[see slide]

Basic Access Auth: Code

- See **PennyBasicAuth** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - **header.html**, footer.html, index.html, searchform.html, searchresults.html
  - **auth.py**
  - **penny.py**

Basic Access Auth: Code

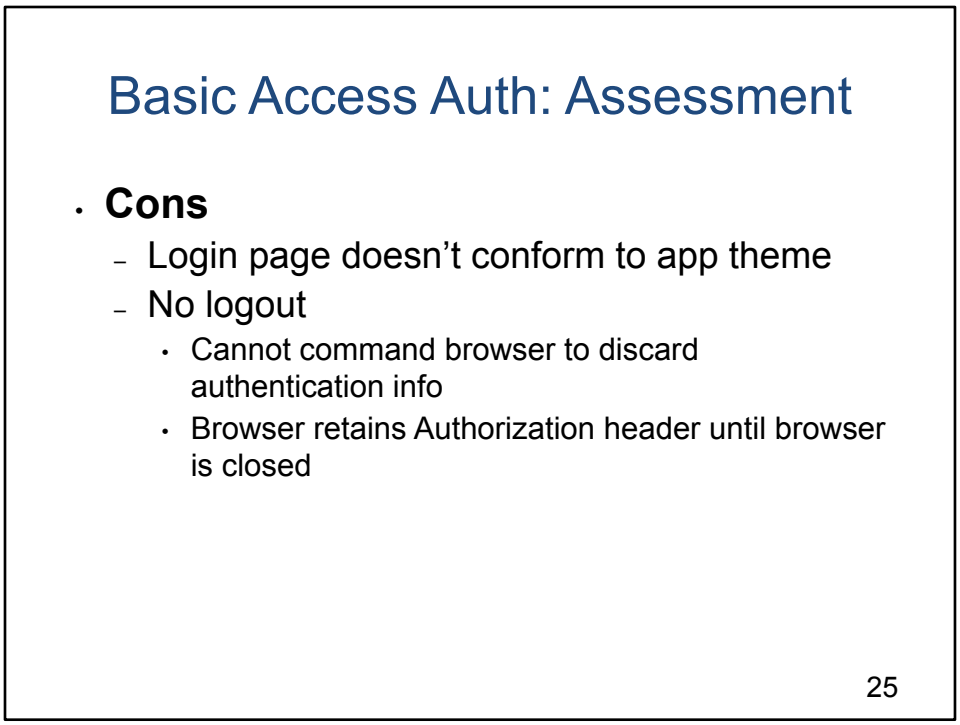
- See **PennyBasicAuth2** app
  - The job:
    - Same as PennyBasicAuth, except...
    - Uses basic access authentication **as implemented by Flask**

Basic Access Auth: Code

- See **PennyBasicAuth2** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - footer.html, index.html, searchform.html, searchresults.html
  - header.html
  - ~~**auth.py**~~
  - **penny.py**

Basic Access Auth: Assessment

- **Pros**
  - Little work for application programmer
  - Simple
  - Uses HTTP headers
    - Does not require HTTP variables, cookies, sessions, login pages
  - Works with any browser and HTTP server



Basic Access Auth: Assessment

Cons

Login page doesn't conform to app theme

No logout

Browser retains authentication info until browser is closed

No way for HTTP server to command browser to discard authentication info

No way for app to implement "logout"

Basic Access Auth: Assessment

- **When used**
  - Personal websites
  - (Maybe) internal websites
  - (Rarely) commercial websites

Agenda

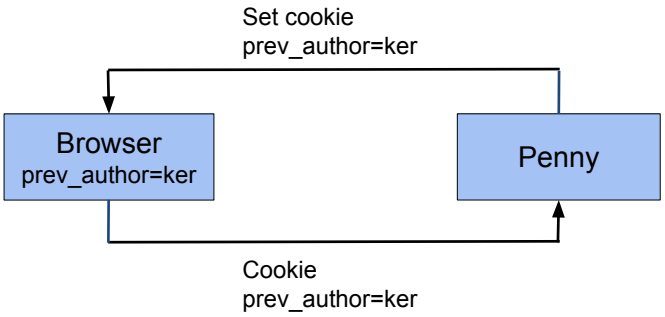
- Broken authentication
- Broken authentication: baseline example
- Broken authentication: basic access authentication
- **Sessions**
- Broken authentication: session-based authentication

Sessions

- Recall **PennyFlaskJinja** app
  - App is stateful
  - Stores `prev_author=ker` across HTTP requests

Sessions

State stored on client-side in a cookie



Sessions

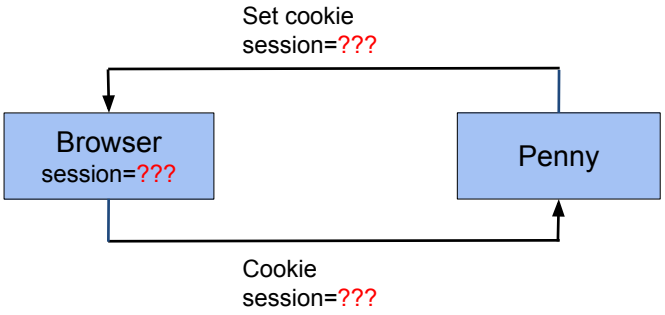
- **Alternative**: ***Sessions***
  - Two kinds:
    - Client-side sessions
    - Server-side sessions

Sessions: Client-Side

- **Client-side sessions**
  - Session data stored in cookie
  - Browser and app communicate cookie
  - Session data are encrypted

Sessions: Client-Side

Encrypted state stored on client-side



eyJwcmV2X2F1dGhvciI6ImtlciJ9.YY8-pg.uaeuSpNjJl-oUKglEFLMBzmfz5w

Using my key, decrypts to:

prev_author=ker

Sessions: Client-Side Example

- See **PennySessionsClient** app
  - The job:
    - Same as PennyFlaskJinja, except…
    - Uses client-side sessions

Sessions: Client-Side Example

- See **PennySessionsClient** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - header.html, footer.html, index.html, searchform.html, searchresults.html
  - **keys.py**
  - **penny.py**

Sessions: Client-Side Assessment

- Assessment of client-side sessions (vs simple cookies)
  - (con) More difficult to learn/debug
    - Cannot reasonably view cookies in browser
  - (pro) Safer
    - Cookies are encrypted

Sessions: Client-Side Assessment

[see slide]

Safety example:

Not a mainstream example, but connects to last lecture…

Suppose you must store persistently not just a character sequence, but a Python data structure, that is, a Python object and maybe nested objects

Could pickle the Python object to a byte sequence

Could store the byte sequence in an ordinary cookie => potential disaster
Could store the byte sequence in a client-side session => much safer

Sessions: Server-Side

- **Server-side sessions**
  - Session data stored on server-side
  - Browser and app communicate cookie
  - Cookie contains only session id

Sessions: Server-Side

State stored on server-side

In file or DB:

SESSIONID	Name:Value
???	prev_author:ker ...
...	...

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

Sessions: Server-Side

- Flask-Session module
  - `python -m pip install Flask-Session`
  - Can be configured to store data on server
    - In files, in database, ...

Sessions: Server-Side Example

- See **PennySessionsServer** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - header.html, footer.html, index.html, searchform.html, searchresults.html
  - ~~**keys.py**~~
  - **penny.py**

Sessions: Server-Side Assessment

- Server-side session (vs client-side session) assessment
  - (pro) Data can be **larger**
    - Client-side: data size limit is 4K
    - Server-side: no data size limit
  - (pro) Data are (somewhat) **safer**
    - Client-side: (encrypted) data are sent across network; (encrypted) data are stored in browser
    - Server-side: data are kept in server

Sessions: Server-Side Assessment

- Server-side session (vs client-side session) assessment
  - (con) More **configuration** is required
    - Must configure to store data in files, database, …
    - Heroku would delete files periodically
  - (con) Impediment to **scalability**
    - Stored data must be accessed and managed
- Remaining examples use **client-side** sessions

Agenda

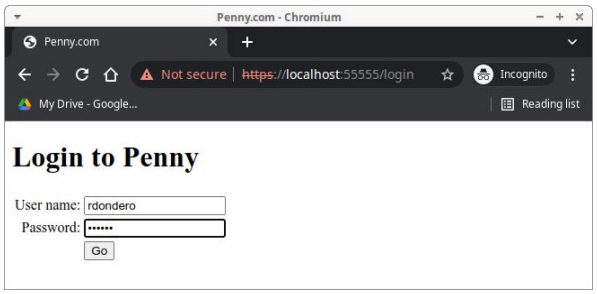
- Broken authentication
- Broken authentication: baseline example
- Broken authentication: basic access authentication
- Sessions
- **Broken authentication: session-based authentication**

Session-Based Auth

- ***Session-Based Authentication***
  - Uses only technologies that we covered already
  - Common programming pattern…

Session-Based Auth: Example

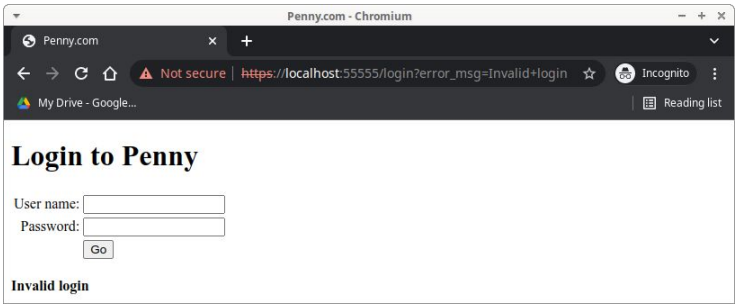
- See **PennySessionAuth** app



Login is accomplished via a form
Not using https => Chrome suggests changing password

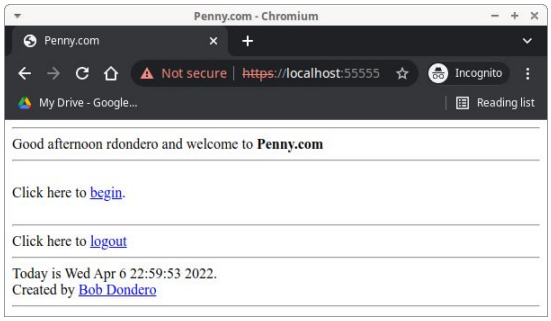
Session-Based Auth: Example

- See **PennySessionAuth** app



Session-Based Auth: Example

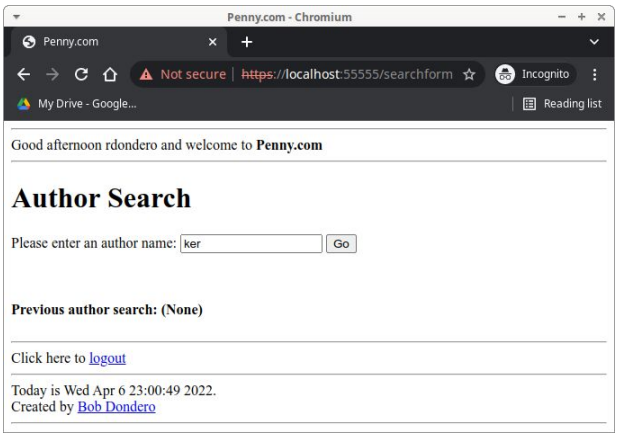
- See **PennySessionAuth** app (cont.)



Note username in header
Note logout link in footer

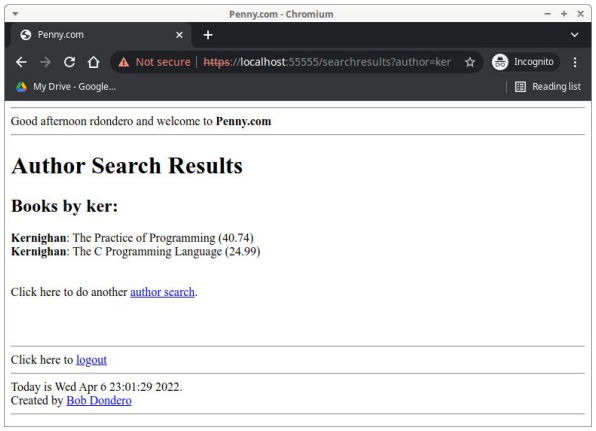
Session-Based Auth: Example

- See **PennySessionAuth** app (cont.)



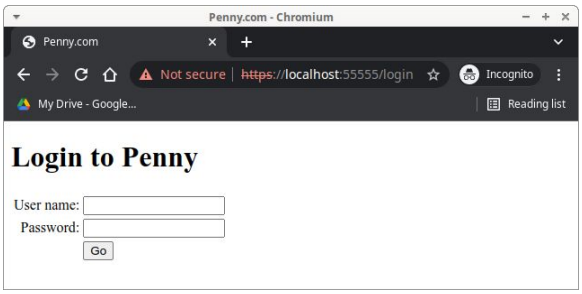
Session-Based Auth: Example

- See **PennySessionAuth** app (cont.)



Session-Based Auth: Example

- See **PennySessionAuth** app (cont.)



Session-Based Auth: Flow

- See **Session-Based Authentication Flow**

Session-Based Auth: Code

- See **PennySessionAuth** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - index.html, searchform.html, searchresults.html
  - **header.html, footer.html**
  - **login.html**
  - **keys.py**
  - **auth.py**
  - **penny.py**

Session-Based Auth: Assessment

- **Pros**:
  - Simple
  - Allows login page to be app-specific
  - Can implement logout
    - Delete username from session

Session-Based Auth: Assessment

- **Cons**:
  - Must implement yourself
  - Might need to add functionality:
    - Account creation, password change, forgotten password recovery, …
    - See Chapter 8 of *Flask Web Development* (Grinberg)
- The most widely used approach commercially

Summary

- We have covered:
  - Broken authentication: baseline example
  - Broken authentication: basic access authentication
  - Sessions
  - Broken authentication: session-based authentication
- See also:
  - **Appendix**: Base64 Encoding

Appendix:
Base64 Encoding

Base64 Encoding

- **Question**:
  - How to represent arbitrary byte sequence using only 64 (that is, 2^6) characters?
    - A-Z (26), a-z (26), 0-9 (10), + (1), / (1)
- **Answer**: ***Base64 encoding***
  - Email uses to represent images, etc.
  - Because those 64 characters are unlikely to be mangled during network transmission

Base64 Encoding

Example (from Wikipedia):

Original text:

```
Man is distinguished, not only by his reason, but by this singular passion from
other animals, which is a lust of the mind, that by a perseverance of delight
in the continued and indefatigable generation of knowledge, exceeds the short
vehemence of any carnal pleasure.
```

Base64 encoded text:

```
TWFuIGlzIGRpc3RpbmdlaXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=
```

Every 1-3 chars of original text is translated into 4 chars in base64 encoded text

Base64 Encoding

Index	Char	Index	Char	Index	Char	Index	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Base64 Encoding

Text content	M																							
ASCII	77 (0x4d)								0 (0x00)								0 (0x00)							
Bit pattern	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Index	19								16								0							
Base64-encoded	T								Q								=							

From Wikipedia

Base64 Encoding

Text content	M							a															
ASCII	77 (0x4d)							97 (0x61)								0 (0x00)							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
Index	19							22								4								0
Base64-encoded	T							W								E								=

From Wikipedia

Base64 Encoding

source ASCII (if <128)	M				a				n			
source octets	77 (0x4d)				97 (0x61)				110 (0x6e)			
Bit pattern	0 1 0 0 1 1 0 1				0 1 1 0 0 0 0 1				0 1 1 0 1 1 1 0			
Index	19				22				5			46
Base64-encoded	T				W				F			u
encoded octets	84 (0x54)				87 (0x57)				70 (0x46)			117 (0x75)

From Wikipedia