COMP 3015

Week 6: Authentication

Week 6 Topics

- Safely storing passwords
  - Cryptographic password hashing functions, salts
    - Bcrypt
- HttpOnly session cookies
- Using session data to determine application access
- Assignment 2 design
  - In-class exercise

Everything we’ve learned in the course so far will be used today.

Storing Passwords

Storing passwords

- Use some kind of encryption algorithm?

  ```
  let passwd  = “aUserPasswordHere321!”
  let key = “2948404D635166546A576E5A7234753777217A25432A462D4A614E645267556B”
  let ciphertext = encrypt(key, passwd) # encryption function such as AES etc.
  ```

- Encrypted data can be decrypted if the appropriate key is supplied

- We need to somehow manage securing the secret key, and admin users (e.g. devs with production access) potentially have access to the key

- If an attacker gains access to the system (or there is a rogue admin), the key and all ciphertext passwords are compromised

For these reasons, among others, we should not be encrypting passwords

Note that some systems encrypt on the client (password managers, for example).

Storing passwords (cont.)

Cryptographic hash functions: what you should be using to store passwords.

- Hash function: maps data of arbitrary size to data of fixed size.

- Cryptographic hash function
  - A hash function designed in a manner such that it is infeasible to reverse engineer the input, given a digest

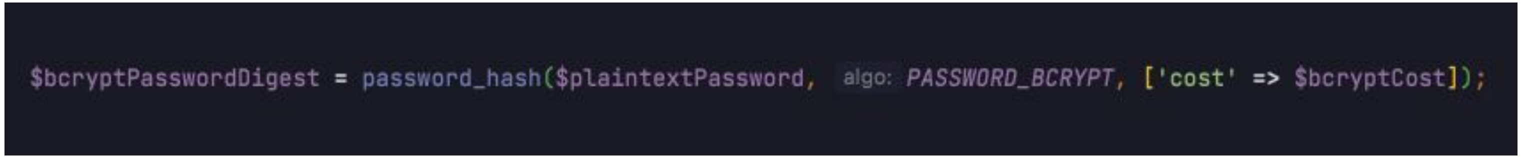
- The output of a cryptographic hash function is called a “digest” or “hash”

Selected Cryptographic Hash/Key Derivation Functions

- Bcrypt: https://en.wikipedia.org/wiki/Bcrypt ←We'll be using this one
- Argon2: https://en.wikipedia.org/wiki/Argon2 ←Key derivation function
- PBKDF2: https://en.wikipedia.org/wiki/PBKDF2 ←Key derivation function
- MD5: https://en.wikipedia.org/wiki/MD5 ←DO NOT USE: collision vulnerabilities
- Lots more. Do your research before choosing one.

Term: cryptographic primitive: low level cryptography algo such as an encryption function or a one way crypto. hash function.

Hashing passwords: Bcrypt



```
$bcryptPasswordDigest = password_hash($plaintextPassword,  algo: PASSWORD_BCRYPT, ['cost' => $bcryptCost]);
```

We can use PHP’s password_hash function in order to compute a Bcrypt digest.

https://www.php.net/manual/en/function.password-hash.php

Hashing passwords: Bcrypt

- Passing in a cost parameter causes Bcrypt to take longer (or shorter) in order to compute.

- Internally there are more iterations performed.

- 04-31 are valid, 10 is the default. Don’t use less than 10 (the default value).
  - **Note**: cost input is used as 2^{cost} internally to Bcrypt, so an input of 11 takes roughly twice as long to compute as an input of 10.

- As with many topics in security, there is a security-usability trade off.
  - Use a higher cost and take longer to compute?
    - This will cause registration, login and password reset times to increase.
  - Use a lower cost and it’s a little bit easier for an attacker to brute force.

Storing passwords: Bcrypt (cont.)

A Bcrypt digest:

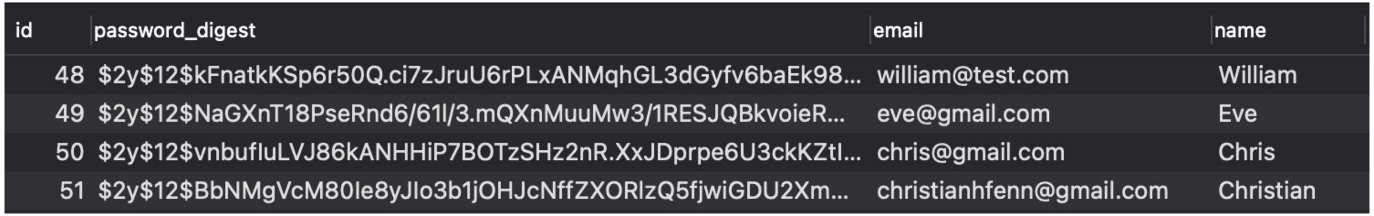
General form:

$2<a/b/x/y>$[cost]$[22 character salt][31 character hash]

Specific instance of a Bcrypt digest:

```
$2y$13$0VJHRpip5rDS3D83oC0yIOOIu6RL4Grk/oLK4naKRtIESOswgFULa
 \__/\__/\_____________________/\_____________________________/
Alg  Cost          Salt                        Hash
```

This is what we actually store in the database:



id	password_digest	email	name
48	$2y$12$kFnatkKSp6r50Q.ci7zJruU6rPLxANMqhGL3dGyfv6baEk98...	william@test.com	William
49	$2y$12$NaGXnT18PseRnd6/61l/3.mQXnMuuMw3/1RESJQBkvoieR...	eve@gmail.com	Eve
50	$2y$12$vnbufluLVJ86kANHHiP7BOTzSHz2nR.XxJDprpe6U3ckKZtl...	chris@gmail.com	Chris
51	$2y$12$BbNMgVcM80le8yJlo3b1jOHJcNffZXORlzQ5fjwiGDU2Xm...	christianhfenn@gmail.com	Christian

Storing Passwords: Salting

The same password with different salts result in different hashes:

password	random salt	input to hash function	resulting hash
bcitcomputing@dtc	afc19b3…	bcitcomputing@dtcafc19b3…	rsRYeAjtwWc…
bcitcomputing@dtc	a2329b3…	bcitcomputing@dtca2329a1…	12aGxQGHC…

Storing Passwords: Salting

Let H be a hash cryptographic hash function.
H(“plaintextPasswordForCOMP3015Demo”) -> OIu6RL4Grk/oLK4naKRtIESOswgFULa

H(“plaintextPasswordForCOMP3015Demo”) -> OIu6RL4Grk/oLK4naKRtIESOswgFULa
Without a salt, the same password results in the same digest.

With a randomly generated salt:
H(“plaintextPasswordForCOMP3015Demo”, “**0VJHRpip5rDS3D83oC0yIO**”)
-> BIu6gRL4Grk/oql4qapztIESOzwgcUnv

H(“plaintextPasswordForCOMP3015Demo”, “**ZVXhqpop92aZ3Dm4xv1ama**”)
-> VoX6RL3qrkmoLK4naKqtIESqswgiaLa
With a (unique) salt, two plaintext passwords result in different digests.
Intent:

- Protect against rainbow tables (precomputed tables of digests of passwords)
- The authentication system (or any admin of it) is not able to tell which users have the same password

Salt, and Pepper?

Yes, there are **peppers** too! An additional hardening technique.

This is a term used to describe an additional input to a cryptographic hash function that is secret.

We won’t use peppers in this course.

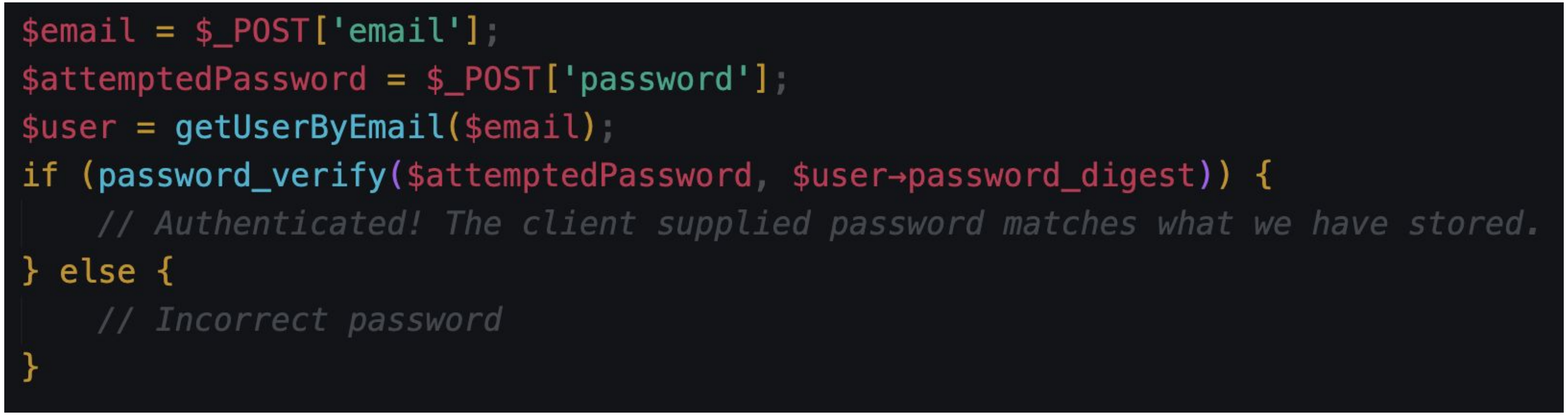
Verifying passwords: Bcrypt (cont.)

At this stage, we are not storing plaintext passwords in our database. Great!

When a user wants to login, how do we verify that the plaintext password entered, matches the digest associated with their username or email?

PHP provides us with password_verify:
https://www.php.net/manual/en/function.password-verify.php

PHP-ish-pseudocode for login



```php
$email = $_POST['email'];
$attemptedPassword = $_POST['password'];
$user = getUserByEmail($email);
if (password_verify($attemptedPassword, $user->password_digest)) {
    // Authenticated! The client supplied password matches what we have stored.
} else {
    // Incorrect password
}
```

Additional Info + tips/general rules

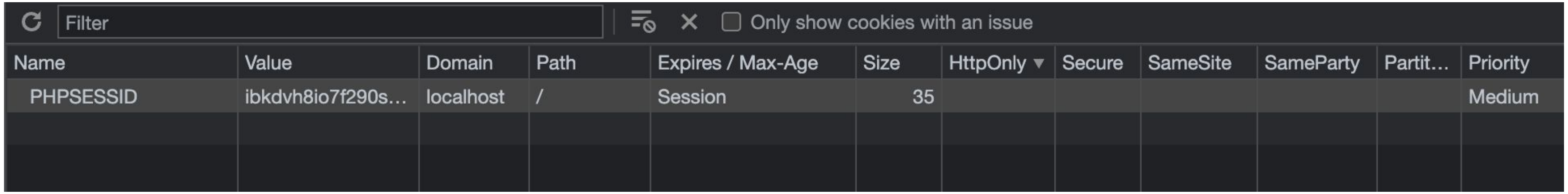
- Don’t write your own hash functions or any other cryptographic primitives
    - Anyone can write an encryption algo that they can’t break, or a hash functions that they can’t find a collision for
    - Use industry standard algorithms that have been rigorously tested and analyzed

- All cryptography systems and primitives should be public knowledge
    - Do not rely on “security through obscurity”

HttpOnly Session Cookies

Setting Session Cookies as HttpOnly

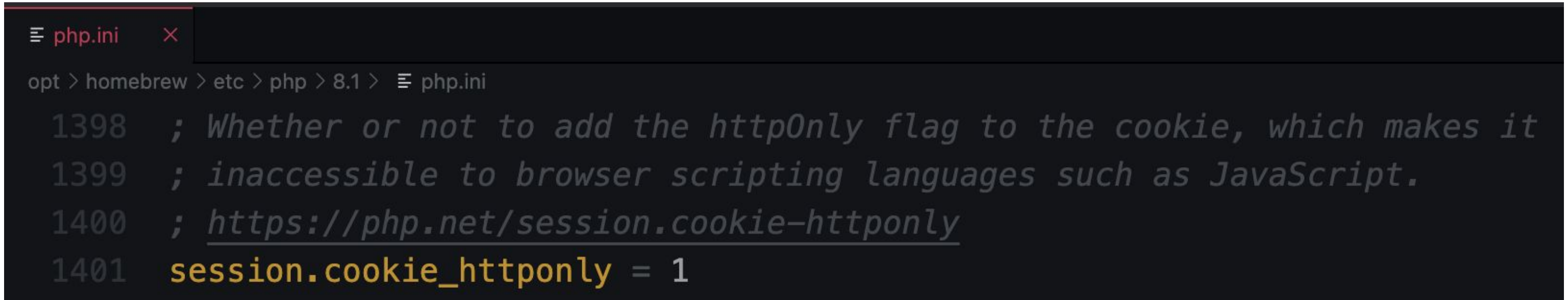
By default we may not be using HttpOnly session cookies:



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	SameParty	Partit…	Priority
PHPSESSID	ibkdvh8io7f290s…	localhost	/	Session	35						Medium

Edit your php.ini file (find it using **php --ini** on the command line)

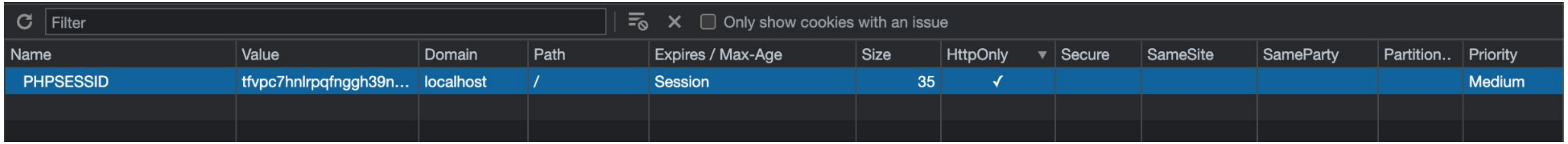
Set the **session.cookie_httponly** value to 1, meaning true:



```
; Whether or not to add the httpOnly flag to the cookie, which makes it
; inaccessible to browser scripting languages such as JavaScript.
; https://php.net/session.cookie-httponly
session.cookie_httponly = 1
```

Setting Session Cookies as HttpOnly (cont.)

After the **session.cookie_httponly** is set to 1 (meaning true), we can start a new session and see that the session cookie is now HttpOnly:



Filter | Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	SameParty	Partition..	Priority
PHPSESSID	tfvpc7hnlrpqfnggh39n...	localhost	/	Session	35	✓					Medium

Assignment 2

3 weeks to complete it.

Key goals:

- Extend assignment 1 to use MySQL instead of a file to store data
- Implement authentication and related business logic
  - Register
  - Login
  - Logout
  - A user can only edit and delete their own article submissions
  - A user can upload a profile picture, but has a default photo
  - Outside of the scope: forgot/reset password
    - We haven’t covered sending emails yet (topic for the next course, COMP 4669)