SWE 322 - Advanced Web Development
Software Engineering Department
College of Engineering and Architecture
Al Yamamah University

PHP and MySQL

Session Management

Instructor: Layla Albabtain

Fall 2019

User Session

- As your web projects grow larger and more complicated, you will find an increasing need to keep track of your users.
- Even if you aren’t offering logins and passwords, you will still often need to store details about a user’s current session and possibly also recognize them when they return to your site.
- Several technologies support this kind of interaction, ranging from simple browser cookies to session handling and HTTP authentication.
    - Between them, they offer the opportunity for you to configure your site to your users’ preferences and ensure a smooth and enjoyable transition through it.

Using Cookies in PHP

- A cookie is an item of data that a web server saves to your computer's hard disk via a web browser.
- It can contain almost any alphanumeric information (as long as it's under 4 KB) and can be retrieved from your computer and returned to the server.
- Common uses include:
  - Session tracking.
  - Maintaining data across multiple visits.
  - Holding shopping cart contents.
  - Storing login details
  - and more.

Using Cookies in PHP

- Because of their privacy implications, cookies can be read only from the issuing domain.
- In other words, if a cookie is issued by, for example, oreilly.com, it can be retrieved only by a web server using that domain.
- This prevents other websites from gaining access to details for which they are not authorized.

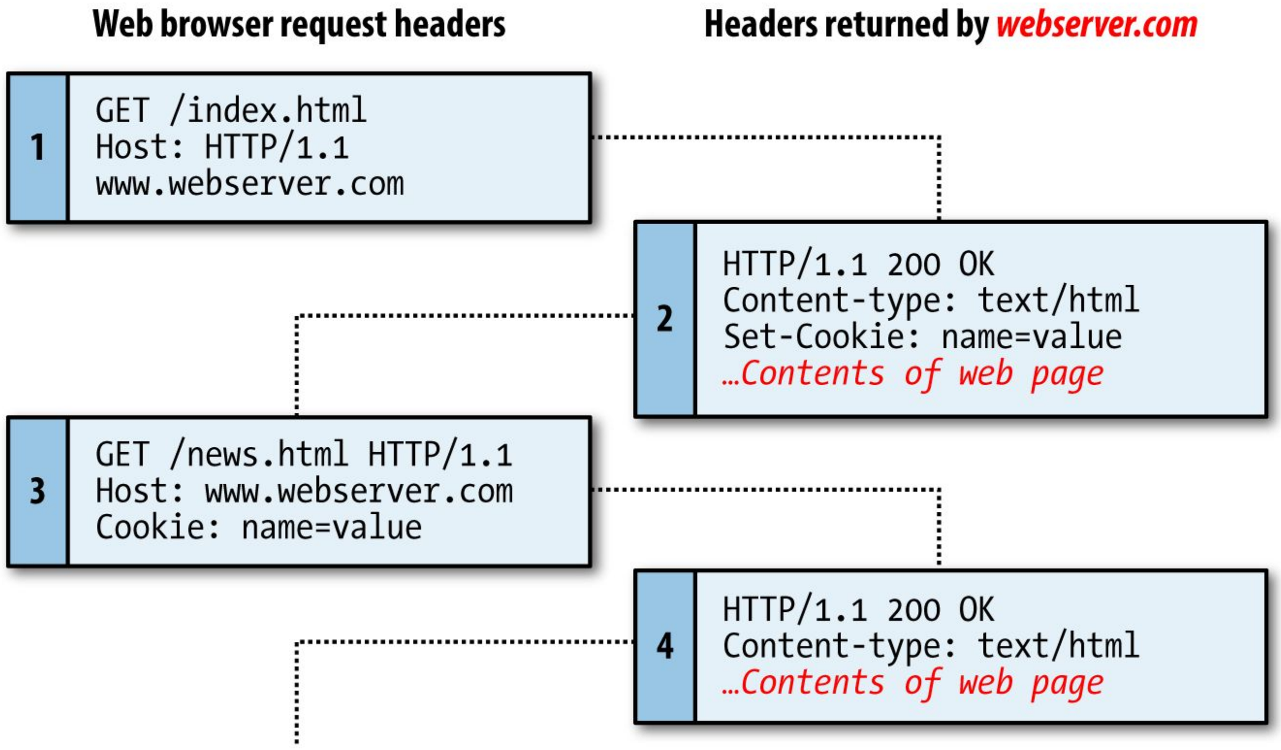
Using Cookies in PHP

- Because of the way the internet works, multiple elements on a web page can be embedded from multiple domains, each of which can issue its own cookies.
- When this happens, they are referred to as third-party cookies.
- Most commonly, these are created by **advertising companies** in order to track **users across multiple websites**.
- Because of this, most browsers allow users to turn cookies off either for the **current server's domain**, **third-party servers**, or **both**.
- Fortunately, most people who disable cookies do so only for third-party websites.

Using Cookies in PHP

- Cookies are exchanged during the transfer of headers, before the actual HTML of a web page is sent, and it is impossible to send a cookie once any HTML has been transferred.
- Therefore, careful planning of cookie usage is important.

Using Cookies in PHP



Web browser request headers
Headers returned by webserver.com
1
2
3
4

```
GET /index.html
Host: HTTP/1.1
www.webserver.com
```

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
…Contents of web page
```

```
GET /news.html HTTP/1.1
Host: www.webserver.com
Cookie: name=value
```

```
HTTP/1.1 200 OK
Content-type: text/html
…Contents of web page
```

Using Cookies in PHP

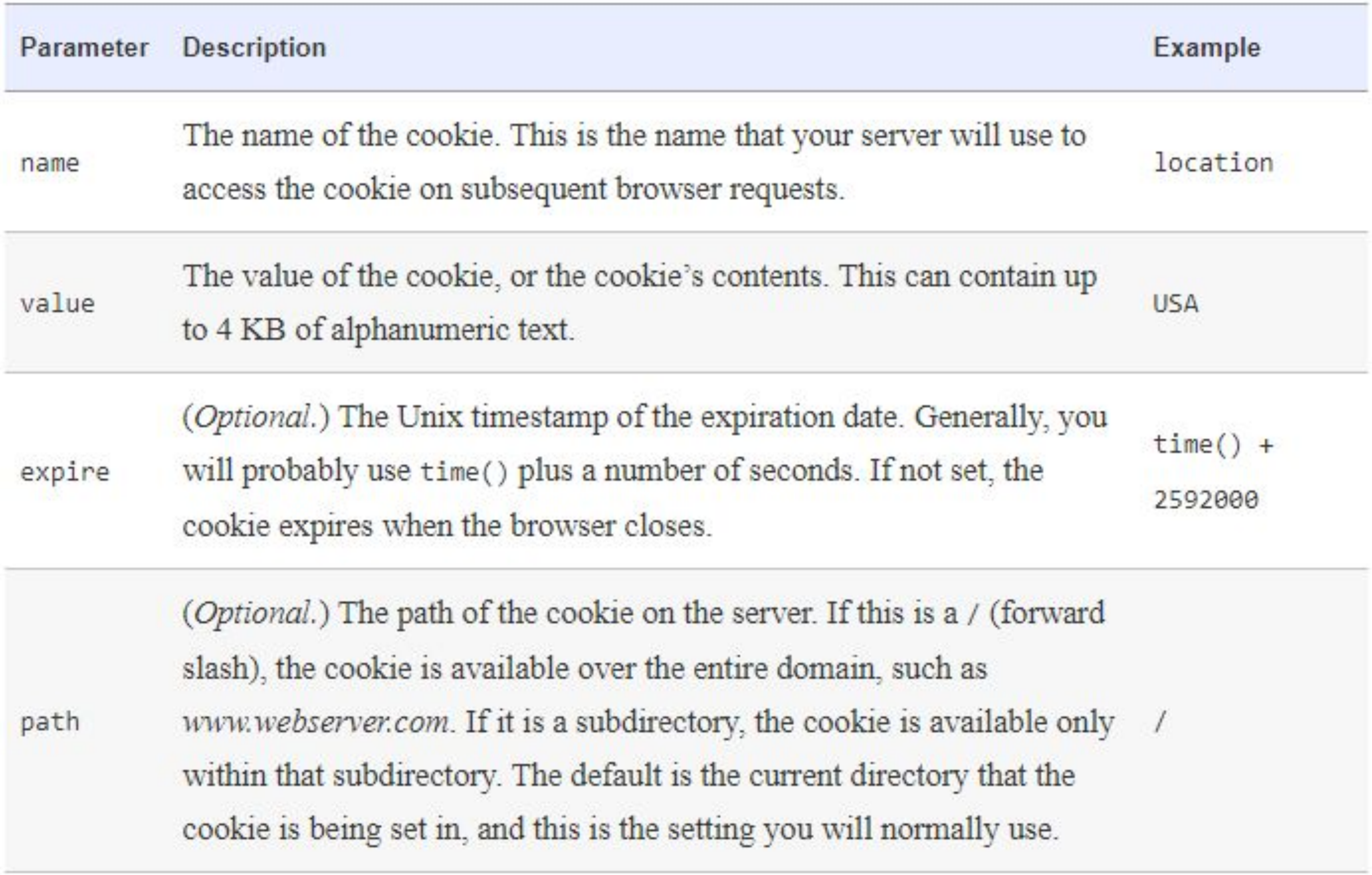
- This exchange shows a browser receiving two pages:
    - The browser **issues a request to:**
        - **Retrieve the main page**, index.html, **at the website http://www.webserver.com.**
            - **The first header specifies the file**.
            - **The second header specifies the server.**
    - When the web server at webserver.com receives this pair of headers, it returns some of its own.
        - The **second header defines the type of content to be sent (text/html)**.
        - **The third one sends a cookie of the name name and with the value value**.
        - Only then are the contents of the web page transferred.

Using Cookies in PHP

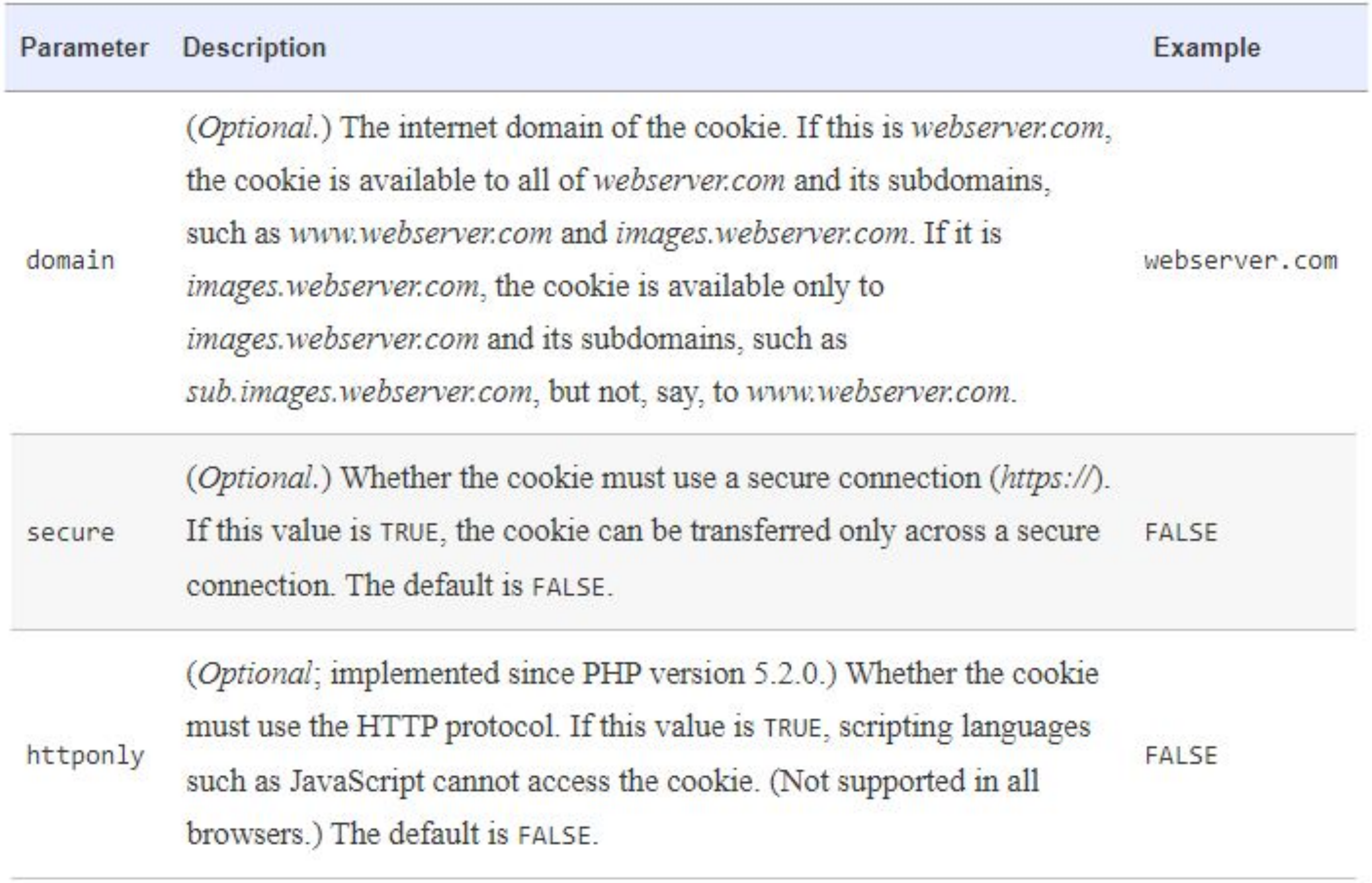
- Once the browser has received the cookie, **it will then return it with every future request made** to the **issuing server** until the **cookie expires or is deleted**.
- So, when the browser requests the new page /news.html, it also returns the cookie name with the value value.
- Because the **cookie has already been set**, when the server receives the request to send /news.html, **it does not have to resend the cookie**, but **just returns the requested page**.

Setting a Cookie

- Setting a cookie in PHP is a simple matter. As long as no HTML has yet been transferred, you can call the setcookie function, which has the following syntax:
  - `setcookie(name, value, expire, path, domain, secure, httponly);`



Parameter	Description	Example
`name`	The name of the cookie. This is the name that your server will use to access the cookie on subsequent browser requests.	`location`
`value`	The value of the cookie, or the cookie's contents. This can contain up to 4 KB of alphanumeric text.	`USA`
`expire`	(*Optional.*) The Unix timestamp of the expiration date. Generally, you will probably use `time()` plus a number of seconds. If not set, the cookie expires when the browser closes.	`time() + 2592000`
`path`	(*Optional.*) The path of the cookie on the server. If this is a `/` (forward slash), the cookie is available over the entire domain, such as *www.webserver.com*. If it is a subdirectory, the cookie is available only within that subdirectory. The default is the current directory that the cookie is being set in, and this is the setting you will normally use.	`/`



Parameter	Description	Example
`domain`	(*Optional*.) The internet domain of the cookie. If this is *webserver.com*, the cookie is available to all of *webserver.com* and its subdomains, such as *www.webserver.com* and *images.webserver.com*. If it is *images.webserver.com*, the cookie is available only to *images.webserver.com* and its subdomains, such as *sub.images.webserver.com*, but not, say, to *www.webserver.com*.	`webserver.com`
`secure`	(*Optional*.) Whether the cookie must use a secure connection (*https://*). If this value is `TRUE`, the cookie can be transferred only across a secure connection. The default is `FALSE`.	`FALSE`
`httponly`	(*Optional*; implemented since PHP version 5.2.0.) Whether the cookie must use the HTTP protocol. If this value is `TRUE`, scripting languages such as JavaScript cannot access the cookie. (Not supported in all browsers.) The default is `FALSE`.	`FALSE`

Example

```
setcookie('location', 'USA', time() + 60 * 60 * 24 * 7, '/');
```

Question:

- What is the name of the cookie?
- What is the value of the cookie?
- How long before the cookie expires?
- The cookies is available to …….?

Accessing a Cookie

- Reading the value of a cookie is as simple as accessing the `$_COOKIE` system array.
- For example, if you wish to see whether the current browser has the cookie called location already stored and, if so, to read its value, use the following:
  - `if (isset($_COOKIE['location'])) $location = $_COOKIE['location'];`
  - **Note:** you can read a cookie back only after it has been sent to a web browser.
  - This means that when you issue a cookie, you cannot read it in again until:
    - **The browser reloads the page** (or **another with access to the cookie**) from your website and passes the cookie back to the server in the process.

Destroying a Cookie

- To delete a cookie, you must **issue it again** and **set a date in the past**.
- It is important for all parameters in your new setcookie call except the timestamp to be **identical to the parameters when the cookie was first issued**; otherwise, the deletion will fail.
- Therefore, to delete the cookie created earlier, you would use the following:
  - `setcookie('location', 'USA', time() - 2592000, '/');`
  - As long as the time given is in the past, the cookie should be deleted.
  - However, the time of 2,592,000 seconds (one month) in the past is used in case the client computer's date and time are not correctly set.
  - You may also provide an empty string for the cookie value (or a value of FALSE), and PHP will automatically set its time in the past for you.

HTTP Authentication

```php
<?php
  $username = 'admin';
  $password = 'letmein';
  // if password has been entered, then check if it matches.
  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    if ($_SERVER['PHP_AUTH_USER'] === $username &&
        $_SERVER['PHP_AUTH_PW']   === $password)
          echo "You are now logged in";
    else die("Invalid username/password combination");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Please enter your username and password");
  }
?>
```

HTTP Authentication

- **Note:** When comparing usernames and passwords the === (identity) operator is used, rather than the == (equals) operator.
- This is because we are checking whether the two values match *exactly*.
    - For example, '0e123' == '0e456', and this is not a suitable match for either username or password purposes.
- A mechanism is now in place to authenticate users, but only for a single username and password.
- Also, the password appears in clear text within the PHP file, and if someone managed to hack into your server, they would instantly know it.
- So, let’s look at a better way to handle usernames and passwords.

Storing Usernames and Passwords

- Obviously, MySQL is the natural way to store usernames and passwords.
- But again, we don’t want to store the passwords as **clear text**. **Why?**
  - Because our website could be compromised if the database were accessed by a hacker.
  - Instead, we’ll use a neat trick called a one-way function.
- This type of function is easy to use and converts a string of text into a seemingly random string.
- Because of their one-way nature, such functions are impossible to reverse, so their output can be safely stored in a database—and anyone who steals it will be none the wiser as to the passwords used.

Storing Usernames and Passwords

- Previously, to store a password securely, you would have needed to salt the password.
- **What is salt**?
  - **Salt** is a term for adding extra characters to a password that the user did not enter (to further obscure it).
- You then needed to run the result of that through a one-way function to turn it into a seemingly random set of characters, which used to be hard to crack.
- For example, code such as the following:
  - `echo hash('ripemd128', 'saltstringmypassword');`
  - Will give: `9eb8eb0584f82e5d505489e6928741e7`

Storing Usernames and Passwords

- This method is now insecure and can be hacked easily. **HOW?**
  - Because modern graphics processing units have such speed and power.
- So what to do?!
  - Use **`PASSWORD_HASH`**.

USING PASSWORD_HASH

- From version 5.5 of PHP, there's a far better way to create salted password hashes: the `password_hash` function.
- Supply `PASSWORD_DEFAULT` as its second (required) argument to ask the function to select the most secure hashing function currently available.
- `password_hash` will also choose a **random salt** for every password.
- Don't be tempted to add any more salting of your own, as this could compromise the algorithm's security.
- Example code:
  - `echo password_hash("mypassword", PASSWORD_DEFAULT);`
  - output: `$2y$10$k0YljbC2dmmCq8WKGf8oteBGiXlM9Zx0ss4PEtb5kz22EoIkXBtbG`

Note:

- If you are letting PHP choose the hashing algorithm for you, you should allow for the returned hash to expand in size over time as better security is implemented.
- The developers of PHP recommend that you store hashes in a database field that can expand to at least 255 characters (even though 60–72 is the average length right now).
- Should you wish, you can manually select the `BCRYPT` algorithm to guarantee a hash string of only 60 characters, by supplying the constant `PASSWORD_BCRYPT` as the second argument to the function.
- However, it is not recommend this unless you have a very good reason.
- For more check this link.

Verifying a Hashed Password

- To verify that a password matches a hash, use the password_verify function, passing it the password string a user has just entered, and the stored hash value for that user’s password (generally retrieved from your database).
- So, assuming your user had previously entered the (very insecure) password of mypassword, and you now have their password’s hash string (from when the user created their password) stored in the variable $hash, you could verify that they match like this:

```
if (password_verify("mypassword", $hash))
  echo "Valid";
```

Verifying a Hashed Password

- If the correct password for the hash has been supplied, password_verify returns the value TRUE, so this if statement will display the word “Valid.”
- If it doesn’t match, then FALSE is returned and you can ask the user to try again.

Example

```
<?php // setupusers.php
     require_once 'db_login.php'; // why not include?
     $connection = new mysqli($db_host, $db_username, $db_password, $db_database);

    if ($connection->connect_error) die("Fatal Error");
    //create user
    $user_name = 'Layla_n';
     $password = 'Layla';
     $email = 'Layla@Layla_n';
     $hash = password_hash($password, PASSWORD_DEFAULT);
```

Example

```
    function add_user($connection, $un, $pw, $em){
            $stmt = $connection->prepare('INSERT INTO user (user_name,
        password, email) VALUES(?,?,?)');
        $stmt->bind_param('sss', $un, $pw, $em);
        $stmt->execute();
        $stmt->close();
    }

    add_user($connection, $user_name, $hash, $email);

?>
```

Resources

- Learning PHP & MySQL Book.
- W3 School.
- PHP
- Learning PHP, MySQL, JavaScript, CSS & HTML5, 3rd Edition.