FRESNO STATE
Discovery. Diversity. Distinction.

Web Programming (CSci 130)

Department of Computer Science
College of Science and Mathematics
California State University Fresno
H. Cecotti

Learning outcomes

- **Goals**
  - Simple login system with PHP
  - User authentication with PHP
  - Security = **key issue**
    - Beyond filling a form with 2 fields (login+password) and retrieving their values

- Reading
  - More information
    - Book JWT Handbook by Sebastian Peyrott (free book - pdf)
      - Read chapter 1, 2, and 3.

Sessions (PHP)

- Project with several pages
  - Session variables = solution to the problem
    - Store user information to be used across **multiple** pages
      - e.g. username, favorite settings..
      - Default: **session** variables last until the user closes the browser
    - The information is not stored on the users computer
      - Cookie = client side
- **PHP**
  - Functions
    - session_start, session_destroy …
  - Variable: $_SESSION['name']=value;
  - Examples
    - create_session.php
    - check_session.php

create_session.php

- Code:

```php
<?php
    // Start the session
    session_start();
    // creates a session or resumes the current one based on a session identifier passed via a GET or POST request, or passed via a cookie.
    // When session_start() is called or when a session auto starts, PHP will call the open and read session save handlers.
?>
<!DOCTYPE html>
<html>
<title>CSci 130 - Web Programming</title>
<body>

<?php
    // Set session variables
    $_SESSION["login"] = "hcecotti@csufresno.edu";
    $_SESSION["password"] = "notmyrealpassword";
    echo "Session variables are set.";
?>

</body>
</html>
```

check_session.php

- Code:

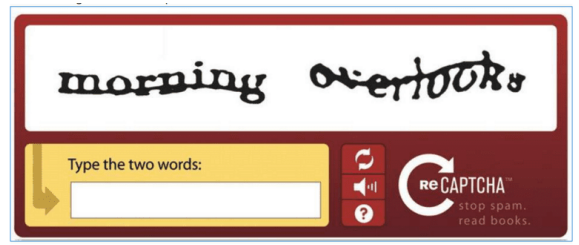
```php
<?php
    session_start();
    // in order to use the variables you need to start the session
?>
<!DOCTYPE html>
<html>
<title>CSci 130 - Web Programming</title>
<body>

<?php
    // Echo session variables that were set with the other page
    if(isset($_SESSION["login"]))
        echo "Login is " . $_SESSION["login"] . ".<br>";
    else
        echo "Login not defined <br>";


    if(isset($_SESSION["password"]))
        echo "Password is " . $_SESSION["password"] . ".";
    else
        echo "Passowrd not defined <br>";

    session_destroy(); // Destroys all data registered to a session

    // session_destroy: Destroys all data registered to a session
    // session_unset: Frees all session variables

?>
</body>
</html>
```

Rationale

- In many websites
  - With an interaction between the user and the system
    - Necessary to record a profile of the user on the **server**
    - To go beyond what a simple cookie may offer
  - To secure information
    - Data **sent** to the server
    - Data **received** from the server
    - Data **on** the server

Typical login page

- Form with inputs
  - ➢ The user exists already
    - Login (textbox)
    - Password (textbox)
  - ➢ The user does not exist
    - Link to a webpage to fill information about the characteristics of the user
      - Including:
        - Login
          - Email address, chosen by the user
        - Password
          - Parse the password to verify that it contains: upper case AND lower case AND digits
          - Password does not belong to a list of existing words
    - CAPTCHA:
      - Completely Automated Public Turing test to tell Computers and Humans Apart
        - Text that cannot be read by an OCR (Optical Character Recognition)
        - Images that cannot be detected by Computer Vision
  - ➢ Lost password
    - Send an email to the user with a new password …



A simple Login system with PHP

- **Files**:
  - simple_loginpage.php
  - simple_logoutpage.php
- Creation of a session based on the output of a form
  - Login + Password

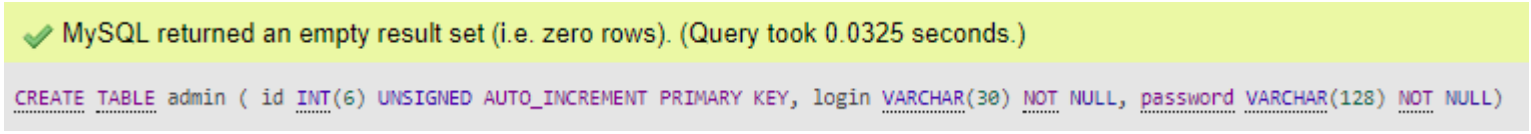
```php
<?php
    ob_start();  // Turn on output buffering
    session_start();
?>

<!DOCTYPE html>
<html lang = "en">
    <head>
        <title>CSci 130 - Example Login + Password</title>
    </head>
    <body>
        <h2>Enter Username and Password</h2>
        <div>
            <?php
                $msg = '';
                if (isset($_POST['login']) && !empty($_POST['username'])
                    && !empty($_POST['password'])) {
                    if ($_POST['username'] == 'myusername' &&
                        $_POST['password'] == '1234') {
                        $_SESSION['valid'] = true;
                        $_SESSION['timeout'] = time();
                        $_SESSION['username'] = 'myusername';
                        echo 'You have entered valid use name and password';
                    }else {
                        $msg = 'Wrong username or password';
                    }
                }
            ?>
        </div>
        <div>
            <form role = "form" action = "<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>" method = "post">
                <h4><?php echo $msg; ?></h4>
                <input type = "text" name = "username" placeholder="username = myusername" required autofocus></br>
                <input type="password" name="password" placeholder="password = 1234" required>
                <button type="submit"  name="login">Login</button>
            </form>
            Clean <a href = "simple_logoutpage.php" tite = "Logout">Session.
        </div>
    </body>
</html>
```

```php
<?php
    session_start();
    unset($_SESSION["username"]);
    unset($_SESSION["password"]);
    echo 'You have cleaned session :)';
    header('Refresh: 2; URL = simple_loginpage.php');
    // http://php.net/manual/en/function.header.php
    // Example:
    // $page = $_SERVER['PHP_SELF'];
    // $sec = "10";
    // header("Refresh: $sec; url=$page");
?>
```

A simple login system with PHP-MySQL

- **Step 1**
  - ➢Create the table
    - ○ We are going to code the password → use enough characters to code the password

```
CREATE TABLE admin (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
login  VARCHAR(30) NOT NULL,
password VARCHAR(128) NOT NULL)
```



A simple login system with PHP-MySQL

- A walkthrough the different functionalities
  - The files
    - config_mysql.php
      - Connection to the server with the different parameters
    - register_mysql.php
      - Register a new user if he doesn't exist
    - login_mysql.php
      - Login page to access the page welcome
    - logout_mysql.php
      - Logout page to close the connection
    - welcome_mysql.php
      - Main page

A simple login system with PHP-MySQL

- Example:

Username:hubert
Password: 1234563
Hashed password: $2y$10$RPu/jenmV2Uf5Lo0xctPJOCyQJgJ0Ite6bZefA4abUmETdV4q3ugu

Login

Please fill in your credentials to login.

Username:* hubert
Password:* •••••• The password you entered was not valid.
Submit

Don't have an account? Sign up now.

Showing rows 0 - 0 (1 total, Query took 0.0013 seconds.)

```
SELECT * FROM `admin`
```

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

	id	login	password
Edit Copy Delete	1	hubert	$2y$10$RPu/jenmV2Uf5Lo0xctPJOCyQJgJ0Ite6bZefA4abUm...

Hi, hubert. Welcome to this site.

Sign out of your Account

Token based authentication

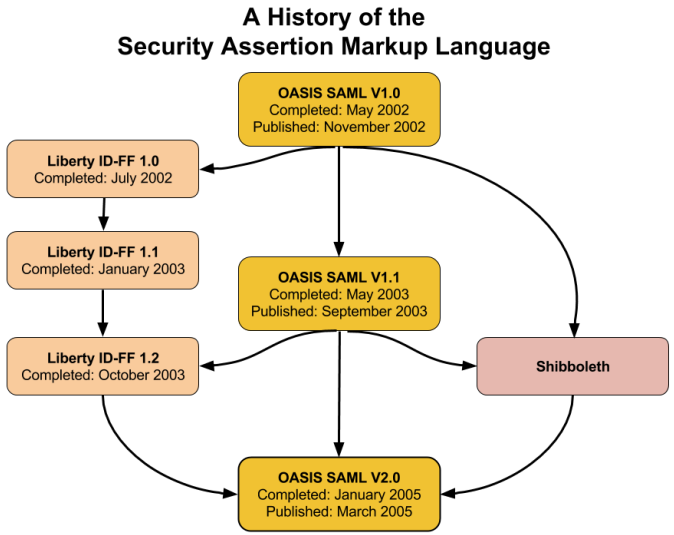
- **Goal**:
  - To enable users to obtain a token that allows them to access a service and/or fetch a specific resource **without** using their username and password to authenticate every request
  - How?
    - The token can be a self-contained entity that conveys all the required information for authenticating the request → **stateless authentication**

JSON Web Tokens (JWT)

- JSON **Web Tokens** (https://jwt.io/)
  - Open, simplicity, compactness
  - Industry standard RFC 7519 method for representing claims securely between two parties
  - Possibility to include signature
    - **JSON Web Signature** (JWS RFC 7515)
  - Possibility to encrypt the information
    - **JSON Web Encryption** (JWE RFC 7516)
- Example
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
    - Compact, printable representation of data, with a signature
    - { “alg”: “HS256”, “typ”:”JWT”, "sub": “1234567890”,”name”: “John Doe”,”admin”: true }

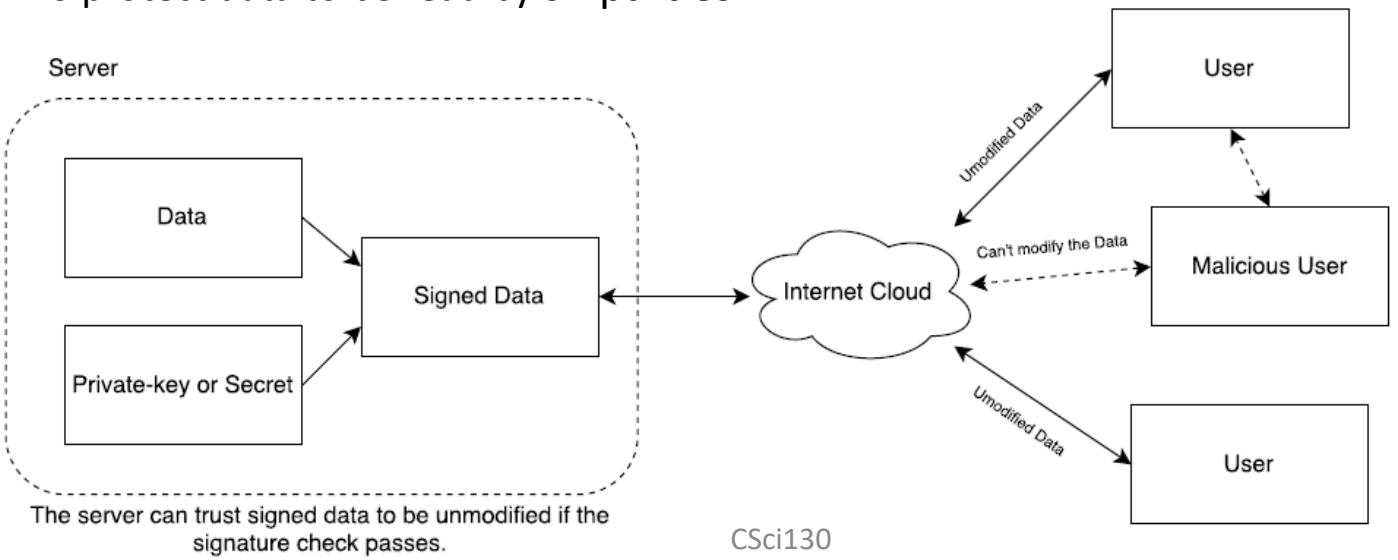
JWT for what?

- Standardization effort
  - ➢Simple, optionally validated and/or encrypted, container format.
- Ad hoc solutions to this same problem have been implemented both privately and publicly in the past…
  - ➢Older standards for establishing claims about certain parties are also available.
- Standard container format for
  - ➢login systems
    - Authentication
    - Authorization
    - Federated identity
    - Client-side sessions (“stateless” sessions)
    - Client-side secrets



Applications of JWT

- Client-side/Stateless sessions
  - ➢ Client side data
    - **Signing**
      - To validate the data against tampering
    - **Encryption**
      - To authenticate and protect the contents of the **session**
      - To protect data to be read by 3rd parties



Client side signed data

- **Security issue**
  - ➢Attack of a signed JWT → remove the signature
  - ➢Signed JWT (3 parts encoded separately)
    1. **Header**
       - It identifies which algorithm is used to generate the signature
    2. **Payload**
       - the claims to make
         - E.g. Information about login and password
    3. **Signature**
       - It is calculated by base64url encoding the header and payload and concatenating them with a period as a separator

Client signed side data

- Signature stripping example

JWT

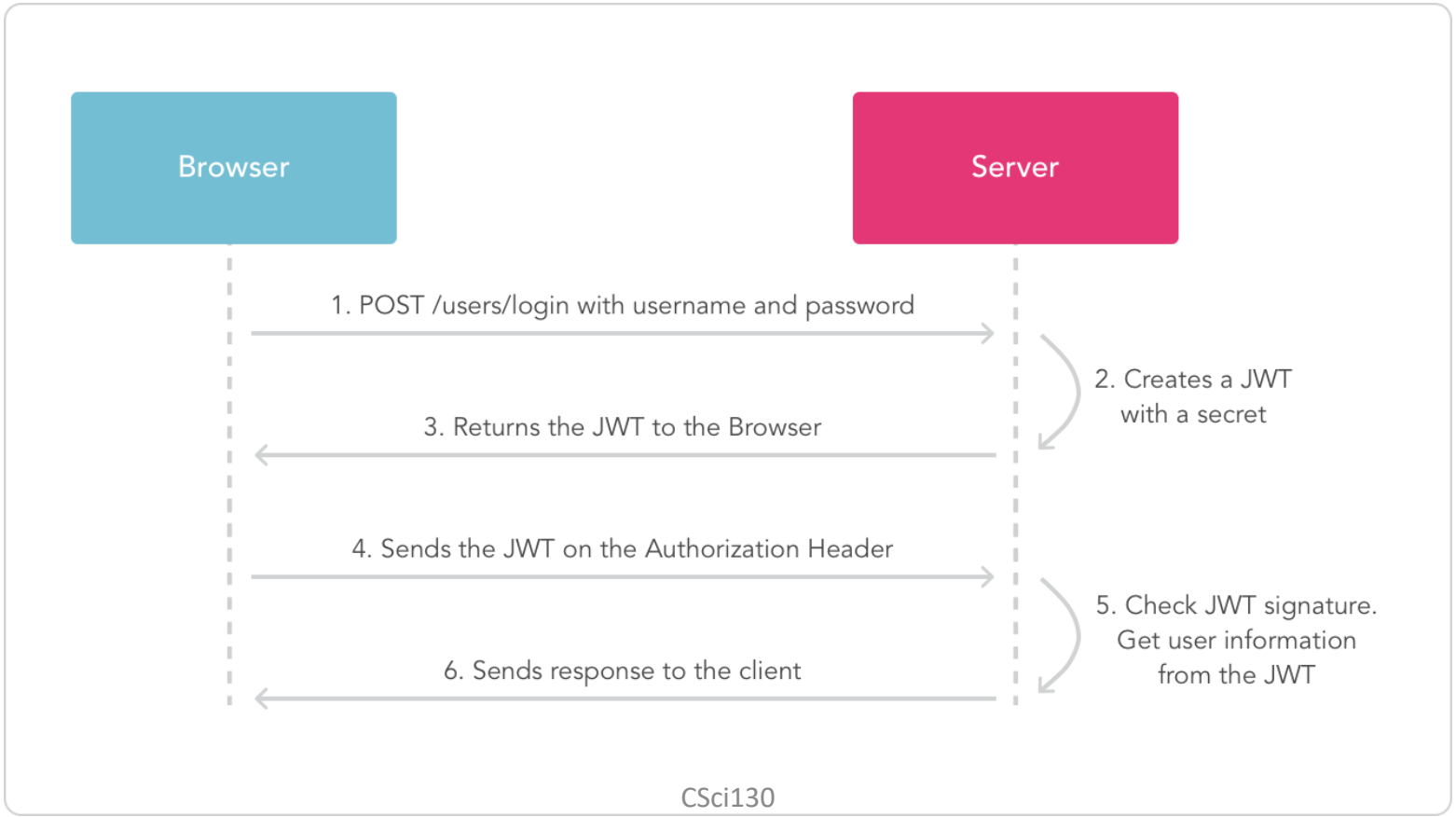
- JWT
  - ➢→Easy way to secure an API
- When a user authenticates first on a server
  - ➢using for instance a standard login form, the server creates a token.
    - token includes **some personal data**
      - Username, email address.
    - This token
      - signed server-side (to prevent token integrity)
      - sent back to the user.
  - ➢Within **each** next request, user sends the token to establish emitter identity

JWT

- **The solution**
  - JSON web tokens are signed by the server
  - **If** the client tampers with the data →
    - **Then** the token's signature will no longer match and an error can be raised !!
- JWT PHP class
  - to create a token after the client successfully logs in
    - $token = array(); $token['id'] = $id; echo JWT::encode($token, 'mysecret_server_key');
  - … later API calls the token can be retrieved and verified by this code:
    - $token = JWT::decode($_POST['token'], 'secret_server_key'); echo $token->id;
- **If** (tampered token) →
  - **then** the $token will be empty there will not be an id available!
- The JWT class makes sure that invalid data is never made available !!
- **If** (tampered token) →
  - **then** it will be unusable.

JWT - Authentication

- 2 key steps



JWT - Authentication

- When the user successfully logs in using their credentials
  - ➢ JSON Web Token will be returned
    - It must be saved locally
      - **local storage or cookies**
      - **instead** of the traditional approach of creating a session in the server and returning a cookie.
- Whenever the user wants to access a protected route or resource
  - ➢ the user agent should **send** the JWT
    - in the Authorization header using the Bearer schema.
- Stateless authentication mechanism
  - ➢ The user state is **never** saved in server memory!
  - ➢ The server's protected routes will check for a valid JWT in the Authorization header
    - **If** (present)
      - **Then** the user will be allowed to access protected resources

JWT - Authentication

- JWTs: self-contained
  - → All the necessary information is there
  - reducing the need to query the database multiple times.
- It allows you to fully rely on data APIs that are stateless and even make requests to downstream services
  - **Definition**: **Statelessness** restriction: It should not keep a client state on the server.
    - It is the responsibility of the client to pass its **context** to the server and then the server can store this context to process the client's further request.
  - →It doesn't matter which domains are serving your APIs
- Cross-Origin Resource Sharing (CORS) won't be an issue because it doesn't use cookies
  - CORS: tricking the user's browser into sending a request from a different site

JSON Web Token implementation

- File
  - jwt_helper.php

Conclusion

- **/!\ Rule #1**: The client cannot be trusted
  - At any moment through the different exchanges
- Signatures
  - useful to validate the data against tampering
- Encryption
  - useful to protect the data from being read by third parties.
- Different approaches
  - Fast development of new frameworks, libraries…
  - 1. **PHP Sessions**
  - 2. **JSON Web tokens**
    - Signed by the server