Урок 7

Авторизация и аутентификация

Знакомство с механизмами Cookies и Сессий в PHP. Реализация сохранения параметров. Знакомство и реализация аутентификации пользователей на сайте. Обзор логики шифрования данных.

Если посмотреть на современный Internet, то мы увидим, что наиболее популярными ресурсами являются соцсети, форумы, интернет-магазины. Все они имеют одну важную схожую черту – они используют регистрацию и последующую авторизацию пользователей. Вполне можно говорить о том, что возможность определять пользователя, отличать пользователя А от пользователя Б - это практически неотъемлемая часть каждого ресурса, если только это не домашняя страничка безымянного кодера. На этом уроке речь пойдёт о том, как реализовать такой функционал.

Принципы работы

Для начала стоит определиться с терминами, которые чаще всего путают и применяют зачастую совершенно неправильно.
Аутентификация - это проверка соответствия субъекта и того, за кого он пытается себя выдать, с помощью некой уникальной информации (отпечатки пальцев, цвет радужки, голос и тд.), в простейшем случае - с помощью имени входа и пароля. Т.е. когда вы входите на сайт, используя свои логин и пароль, то совершаете тем самым процесс аутентификации себя на ресурсе.
Авторизация - это проверка и определение полномочий на выполнение некоторых действий в соответствии с ранее выполненной аутентификацией. Таким образом, после логина, ресурс, согласно внутренней серверной логике, авторизует вас на выполнение тех или иных действий. К примеру, простой пользователь авторизован просматривать опубликованные страницы, но не авторизован их редактировать.
Мы знаем, что PHP-скрипт «живёт» только с момента запуска до момента выполнения последней инструкции. Это означает, что, при переходе между страницами, сервер просто так не сможет знать, кто производил предыдущий запрос. Каждый человек кажется серверу уникальным посетителем, потому что сервер ориентируется только на текущий HTTP-запрос. А это заставит пользователя вводить логин и пароль на каждой странице. «Есть же база данных!» - скажете вы, но база данных есть на сервере. А что делать клиенту, который не имеет доступа к БД?
Для этого создан очень удобный механизм Cookies. Cookies (или просто «куки», от англ. «печенье») – это небольшой кусок текстовой информации, которую сервер сохраняет в браузере пользователя. Благодаря им мы можем идентифицировать пользователя, когда тот заходит на наш сайт. Мы этого не замечаем, но многие сайты сохраняют метки у нас на компьютере. Пожалуй, самым ярким и распространённым примером использования кук является галочка «запомнить меня» при авторизации на сайте.
Каждый элемент куки представляет собой пару «имя=значение». Куки привязываются к доменам, чтобы определенный куки не мог быть использован на сайте, который его не выдавал. Выглядит это примерно так:

```
mysite.com:
        param1 = value1
        param2 = value2
```

Куки хранятся на компьютере пользователя, но все команды по их изменению отправляет сервер. Разумеется, пользователь может подменить куки, и именно поэтому вся информация, которая хранится в них не является доверенной.
Важно: любая информация, пришедшая на сервер от пользователя, по умолчанию не является доверенной и требует проверки и валидации.
Каждая кука содержит срок действия, путь и доменное имя. Домен и путь говорят браузеру, в каких случаях кука должна быть отправлена на сервер. Дата истечения указывает браузеру когда удалить куку. Если срок истечения не указан, кука удаляется по окончании пользовательского сеанса, то есть, с закрытием браузера.

```
Set-Cookie: login=qwerty; 31-Dec-2016 23:59:59 GMT; path=/; domain=.site.ru
```

Куки из примера выше имеет имя «login» и значение «qwerty». Срок её хранения истечёт 31 декабря 2016 года в 23:59:59. Путь «/» и домен «site.ru» показывают браузеру, что нужно отправить куки при просмотре любой страницы данного сайта.
Итак, при работе с куками в PHP необходимо уметь совершать три основные операции:

- отправка куки пользователю;
- проверка наличия куки;
- чтение данных из куки.

Помимо cookies в PHP есть отличный механизм хранения информации в рамках одного сеанса работы пользователя. Сессии - это механизм, позволяющий однозначно идентифицировать браузер и создающий для этого браузера файл на сервере, в котором хранятся переменные сеанса. В отличие от Cookies, сессии хранятся на стороне сервера, позволяя иметь легковесное хранилище информации, не требующей постоянного хранения (например, в БД).
Представьте, что Вам необходимо решить следующую задачку.
Есть две страницы: index.php и login.php. На странице логина пользователь вводит свои данные в поля формы и отправляет запрос. Эти данные мы могли сохранить в переменные и затем показать их на страничке login.php. Однако теперь пользователь может перейти по ссылке на index.php, и там мы также должны выводить эти переменные. Разумеется, так сделать не получится, потому что переменные были стёрты из памяти при завершении скрипта login.php. Получается, что эти данные нам нужно куда-нибудь сохранить. Мы видели, как сохранять информацию в куках. Однако их следует использовать только для важных вещей, так как куки постоянно передаются от клиента к серверу в рамках запросов к сайту. И если кук много, то серьёзно засоряется HTTP-пакет. Более того, куки ограничены по памяти в 4 килобайта.
Общая идея заключается в следующем: для того, чтобы однозначно идентифицировать клиента, достаточно одной метки, а вся информация теперь хранится на сервере в специальном файле. Кука лишь указывает на принадлежность файла конкретному пользователю.
При работе с сессиями различают следующие этапы:

- открытие сессии;
- регистрация переменных сессии и их использование;
- закрытие сессии.

Теперь попробуем всё это применить на практике.

Практическое применение

Начнём с cookies. Для установки куки существует специальная функция, которая называется setcookie. Она может принимать различное количество параметров, но чаще всего работа с ней выглядит следующим образом:

```
setcookie($name, $value, $time);
```

где $name – имя куки, $value – её значение, $time – срок истечения.
Например:

```
setcookie("login", "lamer", time() + 3600 * 24 * 7);
setcookie("password", "qwerty", time() + 3600 * 24 * 7);
```

Последний параметр - время истечения куки - указывается в формате timestamp. Timespamp - это число секунд, прошедших с 00:00:00 1 января, 1970 года. Функция time() возвращает текущее время, а мы прибавляем к нему неделю (3600 * 24 * 7 секунд).
После того, как куки установлены, при каждом заходе пользователя на страницу PHP-интерпретатор формирует глобальный массив $_COOKIE. Ключами в нём являются названия кук, по которым хранятся их значения. В итоге, чтения и проверка наличия кук сводятся к стандартной работе с ассоциативным массивом:

```
if (isset($_COOKIE['login']))
        echo $_COOKIE['login'];
```

В данном фрагменте, когда мы сначала проверили, существует ли вообще у пользователя кука с именем login, и, в случае истинности условия, прочитали её и вывели на экран.
Для того, чтобы куки удалилась, необходимо её «просрочить», т.е. поставить её expire time в виде прошедшей даты. Это автоматически уничтожит её на стороне пользователя.
Перейдём к сессиям. Для создания сессии существует специальная функция session_start(). Эта функция проверяет, существует ли идентификатор сессии, и, если нет, то создаёт его. Если идентификатор текущей сессии уже существует, то загружаются зарегистрированные переменные сессии. После инициализации сессии появляется возможность сохранять информацию в суперглобальном массиве $_SESSION. Также на стороне пользователя появляется куки с именем PHPSESSID, хранящей идентификатор сессии.
Всю работу по записи и чтению информации из файла PHP выполняет автоматически.
Нам лишь остаётся работать с сессией как с ассоциативным массивом. Например, так мы можем сохранить некие персональные данные:

```
session_start();
$_SESSION['name'] = $_POST['name'];
$_SESSION['age'] = $_POST['age'];
```

И теперь для того, чтобы поприветствовать пользователя на странице index, достаточно написать:

```
session_start();
if(isset($_SESSION['name']))
echo 'Привет, ' . ($_SESSION['name'];
```

Когда сессия больше не нужна, например, пользователь нажал кнопку "Выход", следует уничтожить её. Для этого существует функция session_destroy(). Однако перед её вызовом нужно удалить все сохраненные в сессии данные.

```
unset($_SESSION['name']);
unset($_SESSION['age']);
session_destroy();
```

Сессии в PHP являются механизмом, который гармонично дополняет куки. И то, и другое служит для запоминания данных о пользователе, но у каждого из них есть своё назначение. Куки – это долговременное хранилище, а сессии – кратковременное. Сессия сама уничтожается после закрытия браузера пользователем.

Аутентификация и авторизация пользователя при помощи cookies и сессий

Классическим способом применения кук и сессий является аутентификация пользователя на сайте. Для начала мы реализуем самый простой пример. На одной из страниц пользователь вводит логин, пароль и ставит галочку, следует ли его запомнить. При обработке отправки формы мы точно запишем данные в

сессию и сохраним в куку, если пользователь галочку поставил. После этого на каждой странице сайта нужно проверять наличие сохранённого значения в сессии и куках, так как неавторизованный пользователь также может пытаться попасть на страницу, просто вбив её URL-адрес.
При проверке наличия значения в сессии и куках может возникнуть четыре ситуации:

1. Пользователь не авторизован – сессии и куки нет.
2. Пользователь авторизован, не ставил галочку «запомнить» - сессия есть, куки нет.
3. Пользователь долго ничего не делал на сайте, сессия удалилась – сессии нет, куки есть.
4. Пользователь авторизован, поставил галочку «запомнить» - сессия есть, куки есть.

На основе информации, которую мы найдём в массивах $_COOKIE и $_SESSION, уже и будем принимать решение, что дальше делать с пользователем. Форма будет выглядеть примерно так:

```
<form method="post">
    <p><input type="text" name="login" /></p>
    <p><input type="password" name="password" /></p>
    <p><input type="checkbox" name="rememberme" /></p>
    <p><input type="submit" value="Войти" /></p>
</form>
```

Всё очень здорово, но что делать с логином и паролем? Как проверить, что они корректны? Для этого нам потребуется база данных. В ней мы создадим таблицу user, хранящую сущности пользователей, каждому из которых будет сопоставлено два поля:

1. логин.
2. хеш сгенерированной на основании пароля строки.

Hash (хеш функция) — (свёртка) функция однозначного отображения строки (любой длины) на конечное множество (строку заданной длины). Само число (строка) хеш — результат вычисления хеш-функции над данными. В веб-приложениях, в числе прочего, хеш-функции используются для безопасного хранения секретов (паролей) в базе данных. Именно хеш-функция становится вашим последним оплотом, если злоумышленник смог свести нападение к локальной атаке на систему аутентификации.
Ваш хеш в базе должен удовлетворять:

- стойкость к атакам перебора (прямой перебор и перебор по словарю);
- невозможность поиска одинаковых паролей разных пользователей по хешам.

Для выполнения первого требования нужно использовать стойкие в настоящее время (а не в 90-х годах!) хеш-функции – например, Blowfish. Такие хэширующие алгоритмы как MD5, SHA1 и SHA256 были спроектированы очень быстрыми и эффективными. При наличии современных технологий и оборудования, стало довольно просто выяснить результат этих алгоритмов методом "грубой силы" для определения оригинальных вводимых данных.
Для выполнения второго — к паролю перед хешированием добавляется случайная строка (соль). Таким образом, у двух пользователей с паролем «123456» будут разные соли «соль1» и «соль2», а соответственно и результаты работы хеш-функции от «123456соль1» и «123456соль2» в базе тоже будут разные.
Также есть ещё «вторая соль» - она дописывается ко всем (паролям+соль) конструкциям, и является одинаковой для всех хешей в базе. В чем же трюк? В том, что локального параметра в базе нет. Это константа системы, которая хранится в памяти приложения, куда она попадает из конфига (любым способом, только не из базы). Очень простая и действенная мера, которая позволяет практически полностью исключить атаку перебора по данным только одного хранилища хешей (без знания локального параметра).

Итак, мы получаем следующую таблицу:

```
CREATE TABLE `user` (
        `id_user` INT(11) NOT NULL AUTO_INCREMENT,
        `user_name` VARCHAR(50) NOT NULL,
        `user_login` VARCHAR(50) NOT NULL,
        `user_password` VARCHAR(60) NOT NULL,
        `user_last_action` TIMESTAMP NOT NULL,
        PRIMARY KEY (`id_user`)
)
ENGINE=InnoDB;
```

В случае, если пользователь уже был залогинен, то мы выбросим его на главную. Если у пользователя была куки, то мы проверяем её содержимое и также выбрасываем на главную, если всё корректно.
Иначе мы начинаем проверку введённых логина и пароля, после чего выбрасываем пользователя на главную страницу.
Применённые алгоритмы хеширования пароля при помощи crypt реализуют шифрование Blowfish и достаточную безопасность сайта. Стоит заметить, что данный способ эффективнее связки md5+соль, так как за свою работу требует гораздо больше времени работы процессора. А это делает процесс подбора долгим и неэффективным.

Итоги

В рамках урока мы научились идентифицировать пользователя по его логину/паролю, сессии и куки, что позволяет нам в будущем сделать сайт удобным для пользователя, меняя его работу в зависимости от действий отдельно взятого посетителя.

Домашнее задание

1. Создать модуль корзины. В неё можно добавлять товары, а можно удалять товары из неё.

Корзина						
Запрос	**Данные запроса**	**Данные ОК ответа**	**Данные ответа с ошибкой**	**Данные ОК ответа JSON**	**Данные ответа JSON с ошибкой**	**Комментарий**
Добавить товар в корзину	{ "id_product" : 123, "quantity" : 1 }	(string) 1	(string) 0	{ result: 1 }	{ result: 0, errorMessage : "Сообщение об ошибке" }	Подразумеваетс я, что целевая корзина пользователя идентифицирует ся на стороне сервера
Удалить товар из корзины	{ "id_product" : 123 }	(string) 1	(string) 0	{ result: 1 }	{ result: 0, errorMessage : "Сообщение об ошибке" }	

   Использовать также сущность good в качестве товара!
2. Создать модуль личного кабинета, на который будет перенаправляться пользователь после логина. Вывести там имя, логин и приветствие.

3. *Создать модуль регистрации пользователя (см. ссылку в доп. материалах).

Дополнительные материалы

1. http://www.itlessons.info/php/hash-password-by-crypt-and-blowfish/
2. https://habrahabr.ru/sandbox/20718/
3. http://php.net/manual/ru/function.crypt.php

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Изучаем PHP и MySQL - Линн Бейли, Майкл Моррисон

2. “PHP 5 в подлиннике” - Д.Котеров