*Лекция 12: Авторизация доступа с помощью сессий*

### **Авторизация доступа**

Что такое *авторизация доступа*? Попробуем объяснить на примере из обычной жизни. Вы хотите взять в библиотеке книгу. Но эта услуга доступна только тем, у кого есть читательский билет. Можно сказать, что с помощью этого билета производится *"авторизация доступа"* к библиотечным ресурсам. Библиотекарь после предъявления ему читательского билета знает, кто берет книгу, и в случае необходимости (например, книгу долго не возвращают) может принять меры (позвонить должнику домой). Библиотекарь имеет гораздо больше прав, чем обычный посетитель: он может давать или не давать книги определенному посетителю, может выставлять напоказ новинки и убирать в *архив* редко читаемые книги и т.п.

В информационных технологиях все примерно так же. В сети существует огромное количество ресурсов, т.е. множество "библиотек". У каждой из них свой "библиотекарь", т.е. человек или *группа* людей, отвечающих за содержание ресурса и предоставление пользователям информации. Их называют администраторами. Функции администратора, как правило, включают добавление новой информации, удаление и редактирование существующей, настройка способов отображения информации пользователю. А в *функции пользователя* (простого посетителя ресурса) входит только *поиск* и просмотр информации.

Как же отличить пользователя от администратора? В реальной библиотеке это как-то очевидно, но если роли библиотекаря и посетителя библиотеки перенести в виртуальную реальность, то эта очевидность исчезает. Библиотекарь, как и посетитель, имеет *доступ* к библиотечным ресурсам через *Internet*. А согласно протоколу *HTTP* все клиенты абсолютно равноправны. Как же понять, кто зашел на *сайт*? Обычный *пользователь* (посетитель) или *администратор* (библиотекарь)? Если это простой *пользователь*, то как сохранить это *знание*, чтобы не допустить посетителя в закрытые архивы сайта? То есть возникает вопрос, как идентифицировать клиента, который послал *запрос*, и сохранять сведения о нем, пока он находится на сайте?

Самый простой вариант, который приходит в голову, - это *регистрация* человека в системе и выдача ему аналога читательского билета, а именно логина и пароля для входа в административную часть системы. Эта *информация* хранится на компьютере-сервере, и при входе в систему проверяется соответствие введенных пользователем логина и пароля тем, что хранятся в системе. Правда, здесь *по*сравнению с реальной библиотекой ситуация изменяется: читательский билет требуется библиотекарю для входа в закрытую часть системы, а читатель может заходить на *сайт* свободно. В принципе можно регистрировать и простых посетителей. Тогда всех зарегистрированных пользователей нужно разделить на группы: библиотекари (администраторы) и читатели (простые пользователи), наделив их соответствующими правами. Мы не будем вдаваться в эти тонкости и воспользуемся самым простым вариантом, когда ввод логина и пароля требуется для доступа к некоторым страницам сайта.

**Пример 12.1**.

У нас имеется *файл* index.html - домашняя страничка Васи Петрова

<html>

<head><title>My home page</title></head>

<body>

Привет всем!

Меня зовут Вася Петров и

это моя домашняя страничка.

<a href="secret\_info.html">Для Пети</a>

</body></html>

Листинг 12.1. Домашняя страничка Васи Петрова

и *файл* secret\_info.html, который содержит секретную информацию, читать которую разрешено только Васиному другу Пете.

<html>

<head><title>Secret info</title></head>

<body>

<p>Здесь я хочу делиться секретами

с другом Петей.</p>

</body></html>

Листинг 12.1a. secret\_info.html

Если оставить оба эти файла как есть, то любой посетитель, кликнув на ссылку "Для Пети", попадет на секретную страничку. Чтобы этого избежать, нужно добавить промежуточный *скрипт*, который будет проверять, действительно ли Петя хочет попасть на секретную страничку. И сделать так, чтобы главный *файл* ссылался не сразу на secret\_info.html, а сначала на этот *скрипт*.

<html>

<head><title>My home page</title></head>

<body>

<p>Привет всем!

Меня зовут Вася Петров и

это моя домашняя страничка.

</p>

<a href="authorize.php">Для Пети</a>

</body>

</html>

Листинг 12.1b. Index.html

Сам *скрипт* авторизации должен предоставлять форму для ввода логина и пароля, проверять их правильность и перенаправлять на секретную страничку, если проверка прошла успешно, и выдавать *сообщение об ошибке* в противном случае.

<?

if (!isset($\_GET['go'])){

// проверяем, отправлены ли данные формой

echo "<form>

// форма для авторизации

//(ввода логина и пароля)

Login: <input type=text name=login>

Password: <input type=password

name=passwd>

<input type=submit name=go value=Go>

</form>";

}else {

// если форма заполнена, то сравниваем логин

// и пароль с правильными логином и паролем

if ($\_GET['login']=="pit" &&

$\_GET['passwd']=="123") {

Header("Location: secret\_info.html");

//и перенаправляем на секретную страницу

}else echo "Неверный ввод,

попробуйте еще раз<br>";

}

?>

Листинг 12.1c. authorize.php

Вроде бы все достаточно просто. Но допустим, у нас не одна секретная страничка, а несколько. Причем они связаны между собой перекрестными ссылками. Тогда возникает необходимость постоянно помнить *пароль* и логин посетителя сайта (если он таковой имеет). Чтобы решить эту проблему, можно в каждую страницу встроить *скрипт*, который будет передавать логин и *пароль* от страницы к странице в качестве скрытых параметров формы. Но такой способ не совсем безопасен: эти параметры можно перехватить и подделать. В *PHP* существует более удобный и *безопасный метод* решения проблемы хранения данных о посетителе в течение сеанса его работы с сайтом - это механизм *сессий*.

### **Механизм сессий**

Cессии - это механизм, который позволяет создавать и использовать переменные, сохраняющие свое *значение* в течение всего времени работы пользователя с сайтом.

Эти переменные для каждого пользователя имеют различные значения и могут использоваться на любой странице сайта до выхода пользователя из системы. При этом каждый раз, заходя на *сайт*, *пользователь* получает новые значения переменных, позволяющие идентифицировать его в течение этого сеанса или *сессии* работы с сайтом. Отсюда и название механизма - *сессии*.

Задача *идентификации пользователя* решается путем присвоения каждому пользователю уникального номера, так называемого *идентификатора сессии* (*SID*, *Session* IDentifier). Он генерируется *PHP* в тот момент, когда *пользователь* заходит на *сайт*, и уничтожается, когда *пользователь* уходит с сайта, и представляет собой строку из 32 символов (например, ac4f4a45bdc893434c95dcaffb1c1811). Этот *идентификатор* передается на *сервер* вместе с каждым запросом клиента и возвращается обратно вместе с ответом сервера.

Существует несколько способов передачи *идентификатора сессии*:

* С помощью *cookies*.

*Cookies* были созданы специально как метод однозначной идентификации клиентов и представляют собой расширение протокола HTTP. В этом случае *идентификатор сессии* сохраняется во временном файле на компьютере клиента, пославшего запрос. Метод, несомненно, хорош, но многие пользователи отключают поддержку *cookies* на своем компьютере из-за проблем с безопасностью.

* С помощью параметров адресной строки.

В этом случае *идентификатор сессии* автоматически встраивается во все запросы (URL), передаваемые серверу, и хранится на стороне сервера.

Например: *адрес* <http://green.nsu.ru/test.php> превращается в *адрес* <http://green.nsu.ru/test.php?PHPSESSID=ac4f4a45bdc893434c95dcaffb1c1811>

Этот способ передачи идентификатора используется автоматически, если у браузера, отправившего *запрос*, выключены *cookies*. Он достаточно надежный - передавать параметры в адресной строке можно всегда. С другой стороны, *идентификатор сессии* можно подглядеть, воспользоваться сохраненным вариантом в строке браузера или подделать. Хотя, конечно, все эти проблемы либо надуманны либо их можно решить. Например, кто сможет запомнить строку из 32 различных символов? А если правильно организовать работу с *сессиями* (вовремя их уничтожать), то даже сохранившийся в браузере номер *сессии* ничего не даст. К вопросам безопасности мы еще вернемся в конце лекции.

Кроме перечисленных вариантов передачи *идентификатора сессии*, известно еще несколько, но мы их рассматривать не будем ввиду их сложности.

### **Настройка сессий**

Прежде чем начать работать с *сессиями*, следует разобраться в том, как корректно настраивать их обработку интерпретатором *PHP*. Сама работа с *сессиями* в *PHP* поддерживается *по* умолчанию. Это значит, что устанавливать никаких дополнительных элементов не нужно. А вот знать, что записано в настройках этого модуля, полезно, чтобы избежать ошибок при работе с ним.

Настройки *PHP*, в том числе и для работы с *сессиями*, прописываются в файле *php.ini*. Обратимся к этому файлу.

Как мы уже знаем, *идентификатор сессии* (число, *по* которому можно уникально идентифицировать клиента, пославшего *запрос*) сохраняется либо на компьютере-сервере, либо на компьютере-клиенте, либо и там, и там.

Параметр ***session.save\_path*** в *php.ini*, определяет, где на сервере будут храниться данные *сессии* . Из-за него чаще всего возникают проблемы для Windows-серверов, потому что *по* умолчанию *значение* *session.save\_path* установлено в /tmp. И если в корневой директории сервера такой папки нет, то при запуске *сессий* будет выдаваться ошибка.

*Сервер* может обрабатывать большое количество *сессий* одновременно, и все их временные файлы будут храниться в директории, заданной параметром *session.save\_path*. Если система плохо работает с папками большого размера, то удобно использовать поддиректории. Для этого, кроме названия папки, в *значение* параметра добавляют еще и число, определяющее глубину вложенности поддиректорий в этой папке: N;/dir. Это *значение* нужно обязательно взять в кавычки, поскольку точка с запятой является одним из символов комментариев в *файле настроек* *PHP*. Все директории и поддиректории для хранения данных *сессии* нужно создать самостоятельно.

Например: 2;/Temp определяет, что переменные *сессий* будут храниться в папках вида c:\Temp\0\a\, c:\Temp\0\b\ и т.п.

*Хранение данных* на стороне клиента осуществляется с помощью *cookies*. Работу *PHP* с *cookies* можно настроить, в частности, с помощью параметров *session.use\_cookies*, *session.cookie\_lifetime* и т.п.

Параметр ***session.use\_cookies*** определяет, использовать ли *cookies* при работе с *сессиями*. По умолчанию эта опция включена (т.е. принимает значение "1" ).

Параметр ***session.cookie\_lifetime*** задает длительность жизни *cookies* в секундах. По умолчанию это "0", т.е. данные в *cookies*считаются правильными до закрытия окна браузера.

Кроме этих параметров, полезными могут оказаться *session.name*, определяющий имя *сессии*, ***session.auto\_start***, позволяющий автоматически запускать *сессии*, ***session.serialize\_handler***, задающий способ кодировки данных *сессии*, и *параметр****session.cache\_expire***, определяющий, через сколько минут устаревает документ в кэше.

Имя *сессии* *session.name* *по* умолчанию устанавливается как PHPSESSID и используется в *cookies* как *имя переменной*, в которой хранится *идентификатор сессии*. Автоматический *запуск* *сессий* *по* умолчанию отключен, но его можно задать, сделав *значение**session.auto\_start* равным "1". Для кодирования данных *сессии* *по* умолчанию используется *php*. Устаревание данных, сохраненных в кэше, происходит через 180 минут.

Существует еще множество настроек, с которыми можно познакомиться в документации или непосредственно в *файле настроек* *php.ini*. На наш взгляд, знакомства с перечисленными выше параметрами достаточно для работы с *сессиями* в *PHP*. Так что приступим.

### **Работа с сессиями**

#### **Создание сессии**

Первое, что нужно сделать для работы с *сессиями* (если они уже настроены администратором сервера), это запустить механизм *сессий*. Если в настройках сервера переменная *session.auto\_start* установлена в значение "0" (если *session.auto\_start*=1, то *сессии*запускаются автоматически), то любой скрипт, в котором нужно использовать данные *сессии*, должен начинаться с команды

session\_start();

Получив такую команду, сервер создает новую *сессию* или восстанавливает текущую, основываясь на *идентификаторе сессии*, переданном по запросу. Как это делается? Интерпретатор PHP ищет переменную, в которой хранится *идентификатор сессии* (по умолчанию это PHPSESSID) сначала в *cookies*, потом в переменных, переданных с помощью POST- и GET-запросов. Если идентификатор найден, то пользователь считается идентифицированным, производится замена всех URL и выставление *cookies*. В противном случае пользователь считается новым, для него генерируется новый уникальный идентификатор, затем производится замена URL и выставление *cookies*.

Команду session\_start() нужно вызывать во всех скриптах, в которых предстоит использовать переменные *сессии*, причем до вывода каких-либо данных в браузер. Это связано с тем, что *cookies* выставляются только до вывода информации на экран.

Получить идентификатор текущей *сессии* можно с помощью *функции session\_id()*.

Для наглядности *сессии* можно задать имя с помощью функции *session\_name*([имя\_сессии]). Делать это нужно еще до инициализации *сессии*. Получить имя текущей *сессии* можно с помощью этой же функции, вызванной без параметров: *session\_name();*

**Пример 12.2**. Создание сессии

Переименуем наш файл index.html, чтобы обрабатывались *php-скрипты*, например в Index.php, создадим *сессию* и посмотрим, какой она получит идентификатор и имя.

<?

session\_start();

// создаем новую сессию или

// восстанавливаем текущую

echo session\_id();

// выводим идентификатор сессии

?>

<html>

<head><title>My home page</title></head>

... // домашняя страничка

</html>

<?

echo session\_name();

// выводим имя текущей сессии.

// В данном случае это PHPSESSID

?>

Пример 12.2. Создание сессии

Если проделать то же самое с файлом authorize.php, то значения выводимых переменных (id *сессии* и ее имя) будут такими же, если перейти на него с index.php и не закрывать перед этим окно браузера (тогда *идентификатор сессии* изменится).

#### **Регистрация переменных сессии**

Однако от самих идентификатора и имени *сессии* нам пользы для решения наших задач немного. Мы же хотим передавать и сохранять в течение *сессии* наши собственные переменные (например, логин и пароль). Для того чтобы этого добиться, нужно просто зарегистрировать свои переменные:

session\_register(имя\_переменной1,

имя\_переменной2, ...);

Заметим, что регистрируются не значения, а имена переменных. Зарегистрировать переменную достаточно один раз на любой странице, где используются *сессии*. Имена переменных передаются *функции session\_register()* без знака $. Все зарегистрированные таким образом переменные становятся глобальными (т.е. доступными с любой страницы) в течение данной *сессии* работы с сайтом.

Зарегистрировать переменную также можно, просто записав ее значение в ассоциативный *массив $\_SESSION*, т.е. написав

$\_SESSION['имя\_переменной'] =

'значение\_переменой';

В этом массиве хранятся все зарегистрированные (т.е. глобальные) переменные *сессии*.

Доступ к таким переменным осуществляется с помощью массива $\_SESSION['имя\_переменной'] (или $HTTP\_SESSION\_VARS['имя\_переменной'] для версии PHP 4.0.6 и более ранних). Если же в настройках php включена опция *register\_globals*, то к сессионным переменным можно обращаться еще и как к обычным переменным, например так: $имя\_переменной.

Если *register\_globals*=off (отключены), то пользоваться *session\_register()* для регистрации переменных, переданных методами POST или GET, нельзя, т.е. это просто не работает. И вообще, не рекомендуется одновременно использовать оба метода *регистрации переменных*, *$\_SESSION* и *session\_register()*. (Начиная с версии PHP 5.3.0 не рекомендуется для *регистрации переменных сессии*использовать функцию session\_register() ; более того, начиная с версии PHP 6.0.0, эта функция станет недоступна. Вместо этого, для *регистрации переменных сессии* рекомендуется пользоваться массивом $\_SESSION.)

**Пример 12.3**. Регистрация переменных

Зарегистрируем логин и пароль, вводимые пользователем на странице авторизации.

<?

session\_start();

// создаем новую сессию или

// восстанавливаем текущую

if (!isset($\_GET['go'])){

echo "<form>

Login: <input type=text name=login>

Password: <input type=password

name=passwd>

<input type=submit name=go value=Go>

</form>";

}else {

$\_SESSION['login']=$\_GET['login'];

// регистрируем переменную login

$\_SESSION['passwd']=$\_GET['passwd'];

// регистрируем переменную passwd

// теперь логин и пароль - глобальные

// переменные для этой сессии

if ($\_GET['login']=="pit" &&

$\_GET['passwd']=="123") {

Header("Location: secret\_info.php");

// перенаправляем на страницу

// secret\_info.php

}else echo "Неверный ввод,

попробуйте еще раз<br>";

}

print\_r($\_SESSION);

// выводим все переменные сессии

?>

Листинг 12.3a. authorize.php

Теперь, попав на страничку secret\_info.php, да и на любую другую страницу сайта, мы сможем работать с введенными пользователем логином и паролем, которые будут храниться в *массиве $\_SESSION*. Таким образом, если изменить код секретной странички (заметьте, мы переименовали ее в secret\_info.php) так:

<?php

session\_start();

// создаем новую сессию или

// восстанавливаем текущую

print\_r($\_SESSION);

// выводим все переменные сессии

?>

<html>

<head><title>Secret info</title></head>

<body>

<p>Здесь я хочу делиться секретами

с другом Петей.

</body>

</html>

Листинг 12.3b. secret\_info.php

То мы получим в браузере на секретной странице следующее:

Array ( [login] => pit [passwd] => 123 )

Здесь я хочу делиться секретами

с другом Петей.

В итоге получим *список переменных*, зарегистрированных на authorize.php и, собственно, саму секретную страничку.

Что это нам дает? Допустим, хакер хочет прочитать секреты Васи и Пети. И он как-то узнал, как называется секретная страничка (или странички). Тогда он может попытаться просто ввести ее адрес в строке браузера, минуя страницу авторизации (ввода пароля). Чтобы избежать такого проникновения в наши тайны, нужно дописать всего пару строк в код секретных страничек:

<?php

session\_start();

// создаем новую сессию или

// восстанавливаем текущую

print\_r($\_SESSION);

// выводим все переменные сессии

if (!($\_SESSION['login']=="pit" &&

$\_SESSION['passwd']==123))

// проверяем правильность

// пароля-логина

Header("Location: authorize.php");

// если ошибка, то перенаправляем на

// страницу авторизации

?>

<html>

<head><title>Secret info</title></head>

... // здесь располагается

//секретная информация :)

</html>

Листинг 12.3c. 2-я версия secret\_info.php

#### **Удаление переменных сессии**

Кроме умения регистрировать переменные *сессии* (т.е. делать их глобальными на протяжении всего сеанса работы), полезно также уметь удалять такие переменные и *сессию* в целом.

***Функция session\_unregister(имя\_переменной)*** удаляет глобальную переменную из текущей *сессии* (т.е. удаляет ее из списка зарегистрированных переменных). Если регистрация производилась с помощью *$\_SESSION* ( *$HTTP\_SESSION\_VARS* для версии PHP 4.0.6 и более ранних), то используют языковую конструкцию unset(). Она не возвращает никакого значения, а просто уничтожает указанные переменные.

Где это может пригодиться? Например, для уничтожения данных о посетителе (в частности, логина и пароля) после его ухода с секретной странички. Если правильные логин и пароль сохранятся и окно браузера после посещения сайта не закрыли, то любой другой пользователь этого компьютера сможет прочитать закрытую информацию.

**Пример 12.4**. Уничтожение переменных сессии

В файл secret\_info.php добавим строчку для выхода на главную страницу:

<?php

// ... php код

?>

<html>

<head><title>Secret info</title></head>

... // здесь располагается

// секретная информация :)

<a href="index.php">На главную</a>

</html>

Пример 12.4a. secret\_info.php

В Index.php уничтожим логин и пароль, введенные ранее:

<?

session\_start();

session\_unregister('passwd');

// уничтожаем пароль

unset($\_SESSION['login']);

// уничтожаем логин

print\_r($\_SESSION);

// выводим глобальные переменные сессии

?>

<html>

<head><title>My home page</title></head>

... // домашняя страничка

</html>

Пример 12.4b. Index.php

Теперь, чтобы попасть на секретную страницу, нужно будет опять вводить логин и пароль.

Для того чтобы сбросить значения всех переменных *сессии*, можно использовать *функцию session\_unset();*

Уничтожить текущую *сессию* целиком можно командой *session\_destroy();* Она не сбрасывает значения глобальных переменных *сессии* и не удаляет *cookies*, а уничтожает все данные, ассоциируемые с текущей *сессией*.

<?

session\_start(); // инициализируем сессию

$test = "Переменная сессии";

$\_SESSION['test']= $test;

// регистрируем переменную $test.

// если register\_globals=on,

// то можно использовать

// session\_register('test');

print\_r($\_SESSION);

// выводим все глобальные переменные

echo session\_id();

// выводим идентификатор сессии

echo "<hr>";

session\_unset();

// уничтожаем все глобальные

// переменные сессии

print\_r($\_SESSION);

echo session\_id();

echo "<hr>";

session\_destroy(); // уничтожаем сессию

print\_r($\_SESSION);

echo session\_id();

?>

Пример 12.5. Уничтожение сессии и глобальных переменных

В результате работы этого скрипта будут выведены три строки: в первой - массив с элементом test и его значением, а также *идентификатор сессии*, во второй - пустой массив и *идентификатор сессии*, в третьей - пустой массив. Таким образом, видно, что после уничтожения *сессии* уничтожается и ее идентификатор, и мы больше не можем ни регистрировать переменные, ни вообще производить какие-либо действия с *сессией*.

### **Безопасность**

Вообще говоря, cледует понимать, что использование механизма *сессий* не гарантирует полной *безопасности системы*. Для этого нужно принимать дополнительные меры. Обратим внимание на проблемы с безопасностью, которые могут возникнуть при работе с *сессиями*и, в частности, с теми программами, что мы написали.

Во-первых, опасно передавать туда-сюда *пароль*, его могут перехватить. Кроме того, мы зарегистрировали его как глобальную переменную *сессии*, значит, он сохранился в *cookies* на компьютере-клиенте. Это тоже плохо. И вообще, пароли и логины *по*-хорошему должны храниться в базе данных. Пусть *информация* о пользователях хранится в базе данных "test" (в таблице "users"), а мы имеем к ней *доступ* под логином my\_user и паролем my\_passwd.

Во-вторых, что делать, если кто-то написал *скрипт* подбора пароля для секретной страницы? В этом случае на страницу авторизации много раз должен стучаться какой-то посторонний *скрипт*. Поэтому нужно просто проверять, с нашего ли сайта пришел *запрос* на авторизацию, и если нет, то не пускать его дальше. *Адрес* страницы, с которой поступил *запрос*, можно получить с помощью глобальной переменной *$\_SERVER['HTTP\_REFERER'])*. Хотя, конечно, если за взлом сайта взялись всерьез, то *значение* этой переменной тоже подменят (например, с помощью того же *PHP*). Тем не менее проверку ее значения можно считать одним из важнейших шагов на пути к обеспечению безопасности своего сайта.

<?

session\_start();

// создаем новую сессию или

// восстанавливаем текущую

$conn = mysql\_connect("localhost",

"my\_user","my\_passwd");

// устанавливаем соединение с сервером БД

mysql\_select\_db("test");

// выбираем рабочую базу данных

$SERVER\_ROOT = "http://localhost/~nina/tasks/sessions/";

// где находятся наши скрипты

/\* с помощью регулярного выражения

^$SERVER\_ROOT и функции eregi проверяем,

начинается ли адрес ссылающегося скрипта,

т.е. строка $\_SERVER['HTTP\_REFERER'])

со строки $SERVER\_ROOT (как у нас) \*/

if(eregi("^$SERVER\_ROOT",

$\_SERVER['HTTP\_REFERER'])){

// если да, то делаем почти то же, что и

// раньше, пароль регистрировать не будем

if (!isset($\_POST['go'])){

echo "<form method=POST >

Login: <input type=text name=login>

Password: <input type=password name=passwd>

<input type=submit name=go value=Go>

</form>";

}else {

/\* запрос к базе данных: выбираем из таблицы

users login, который совпадает с переданным

по запросу, причем пароль у него тоже должен

совпасть с введенным пользователем.

Если этого нет, то считаем, что логин и

пароль введены неверно \*/

$sql = "SELECT login FROM users

WHERE login='" . $\_POST['login'] . "' AND passwd='" . $\_POST['passwd'] . "';";

$q = mysql\_query($sql,$conn); // отправляем запрос к БД

$n = mysql\_num\_rows($q); // число строк в ответе на запрос

if (!$n==0){

$\_SESSION['user\_login']=$\_POST['login'];

// регистрируем переменную login

Header("Location: secret\_info.php");

// перенаправляем на страницу secret\_info.php

}else echo "Неверный ввод, попробуйте еще раз<br>";

}

print\_r($\_SESSION); // выводим все переменные сессии

}

?>

Листинг 12.6. authorize.php

Вроде бы первые две проблемы решены. Но есть еще одна. Что делать, если *хакер* просто допишет в строку запроса *значение* какой-нибудь глобальной переменной (например, логина)? Вообще это возможно, только если *register\_globals*=On. Просто иначе мы используем для работы с глобальными переменными *массив $\_SESSION* и с ним такие фокусы не проходят. Все же попробуем решить и эту проблему. Для этого нужно очистить строку запроса перед тем, как сравнивать значения параметров. То есть сначала сбросим *значение* $user\_login. Потом данную переменную нужно опять зарегистрировать, но не как новую, а как уже существующую. Для этого знак доллара при регистрации НЕ опускается. Вот что получилось:

<?php

unset($user\_login); // уничтожаем переменную

session\_start(); // создаем новую сессию или

// восстанавливаем текущую

session\_register($user\_login);

// регистрируем переменную

// как уже существующую

if (!($user\_login=="pit")) // проверяем логин

Header("Location: authorize.php");

// если ошибка, то перенаправляем

// на страницу авторизации

?>

<html>

<head><title>Secret info</title></head>

... // здесь располагается

// секретная информация :)

</html>

Листинг 12.7. secret\_info.php

### ***Заключение***

Итак, мы познакомились с *сессиями* и основными способами работы с ними, проблемами, возникающими при их использовании, и возможными решениями этих проблем. Надеюсь, что после прочтения лекции читателям стало ясно, насколько удобны и просты в использовании *сессии*, а приведенные примеры пригодятся на практике.