

Регистрация и COOKIE

- Введение
- Информационная безопасность
- Практика регистрации с открытым паролем
- Кодировка, шифрование, хеширование
- Практика регистрации с хешированием

Введение

Текущий первый в серии, посвященный вопросам персонификации пользователя в системе. Практически всегда работа пользователя в некоторой системе начинается с процесса регистрации.

Именно процесс регистрации рассмотрим в большом количеством демонстрационных примеров.

Примечание. Некоторые представленные практики не используются в реальных приложениях, но дают наглядное представление происходящим процессам.

Обратите внимание, что для передачи данных между сценариями ранее мы использовали **GET-параметры** адресной строки или элементы управления **форм**. Теперь для этих целей будем использовать текстовые данные файлов **cookie**.

Важно понимать, один способ обмена данными не отменяет, а скорее дополняет другой.

Информационная безопасность

Понятие информационная безопасность и защиты личных данных пользователей актуальный вопрос для многих сервисов, начиная с простого облачного хранилища файлов и заканчивая онлайн банками.

Нарушение правил информационной безопасности может привести к катастрофическим последствиям. Например, злоумышленник, получив доступ к банковской карте, начинает распоряжаться средствами вашего счета. Или захочет переоформить недвижимость, оформленную на ваше имя.

Для сохранения информации используется базовые процессы любой современной системы:

- **Регистрация**
- **Идентификация**
- **Аутентификация**
- **Авторизация.**

С этими терминами пользователи сталкиваются ежедневно, но не каждый понимает, что означает каждый из них.

В текуще разберемся, в чем разница между этими процессами и выполним практику регистрации пользователя в системе.

Регистрация

Регистрация – процесс **создания** в системе **учетной записи** пользователя (аккаунта). С точки зрения баз данных это создание новой **записи**, принадлежащей зарегистрированному пользователю в таблице пользователей базы.

Набор первичных данных регистрации определяется индивидуально в каждой системе. Чаще всего это будут **идентификатор** и **пароль** пользователя. В дальнейшем система может предложить расширить список личных данных пользователя.

Идентификация

Это процесс, во время которого сервис определяет зарегистрирован ли данный человек в системе или нет. Для выяснения этого обстоятельства используется **идентификатор**.

В качестве **идентификатора** используется:

- логин (имя пользователя, под которым он зарегистрировался в системе);
- номер телефона;
- электронная почта;
- и пр.

В принципе, для идентификации может быть использован любой **признак**, при условии, что, что он есть только у одного пользователя.

Важно. Система не может зарегистрировать двух пользователей с одинаковыми признаками (... или так – вы не можете написать программу, которая зарегистрирует двух пользователей с одинаковыми признаками).

Например, при регистрации нередко появляется сообщение: "Данный логин уже занят", для продолжения нужно выбрать другой, **уникальный для этого приложения** или **сайта** логин.

Бывают приложения, в которых есть возможность выбора. К примеру, залогиниться по имени пользователя или по номеру телефона. Важно то, что этот идентификатор будет являться уникальным значением.

В базе данных это, вероятнее всего, будет столбец в таблице с **уникальным** полем (Unique Key).

Аутентификация

Такой признак, как логин или номер телефона не составляет большой тайны, эти сведения зачастую находятся в открытом доступе, и любой человек может войти в аккаунт другого пользователя. Для дополнительной проверки права входа в аккаунт используется **аутентификация**.

Аутентификация – это процесс, для которого создается уникальный ключ для подтверждения своего права входления в учетную запись. В качестве такого ключа чаще всего используется комбинация буквенных и цифровых символов, то есть, **пароль**.

Знание идентификатора без знания пароля бессмысленно, впрочем, верно и обратное утверждение – использование пароля без идентификатора бесполезно.

Подтверждение права на вход путем ввода пароля является **однофакторной аутентификацией**. Этот вариант используется в большинстве ресурсов, которые не содержат конфиденциальной информации.

Обратите внимание. Если доступ к системе предоставляется после введения **логина и пароля**, то это будет **однофакторная**

аутентификация - самая простая.

Однако для безопасности в системах, в которых запрашиваются и сохраняются личные данные, требуется прохождение дополнительных этапов, то есть используется **двух или трехфакторная аутентификация**.

Авторизация

Теперь разберемся, чем отличается авторизация от аутентификации и идентификации.

Авторизация – это процесс открытия доступа к определенным правам или привилегиям.

Этот процесс необходим, чтобы исключить вмешательство обычных пользователей в работу системы. Например, право добавлять / редактировать материал предоставляется администратору ресурса, право оставлять комментарии, отправлять email –зарегистрированным пользователям. Обычные пользователи ресурса выполнить такие действия не могут.

Чтобы отличить права администратора от зарегистрированного или рядового пользователя, используется авторизация.

Примечание. В некоторых системах может выполняться еще один процесс – **верификация**. Верификация проводится при необходимости дополнительного подтверждения права доступа субъекта к учетной записи.

Если подвести итоги сказанного, то рассматриваемые процессы можно описать следующими, более простыми словами:

- **Регистрация** — **создание в системе учетной записи** пользователя.
- **Идентификация** используется для определения, **существует ли конкретный пользователь в системе**. В качестве идентификатора может использоваться, например, логин, номер телефона, email и другие признаки.
- **Аутентификация** — это процесс **подтверждения права на доступ** с помощью ввода и проверки на соответствие идентификатора и пароля. Могут использоваться биометрические данные и другие способы.
- **Авторизация** определяет **набор привилегий и прав**, доступных конкретному пользователю. Например, открывает доступ к написанию комментариев, редактированию материала, просмотру и отправке электронных писем.

Процессы установления личности и предоставления прав можно описать следующими вопросами:

- **Идентификация** – кто вы такой?
- **Аутентификация** – вы точно тот, за кого себя выдаете?
- **Авторизация** – к чему вы имеете доступ?

Работа в некоторой системе всегда начинается с регистрации.

Проверка всегда начинается с идентификации, за ней идёт аутентификация, и в конце — авторизация.

Практика регистрации с открытым паролем

Простая регистрация

В многочисленных простых примерах покажу пошаговое выполнение процесса регистрации. Отслеживайте файлы **cookies** и состояние **базы данных** по мере выполнения каждого из демонстрационных примеров.

Начнем с самого простого примера регистрации. Поехали.

example_1. index.html

```
<form action="reg.php" method="post">

    Логин: <input type="text" name="login" value=""><p>

    Пароль: <input type="text" name="pwd" value=""><p>

    <button>Регистрация</button>

</form>
```

Обработчик формы регистрации представлен в файле **reg.php**.

example_1. reg.php.

```
<?php

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare ('INSERT INTO `user` SET `login` = ?, `pwd` = ?');

    $stmt->bind_param ('ss', $_POST['login'], $_POST['pwd']);

    $stmt->execute ();

    if ($stmt->affected_rows) {

        echo "<h3>Вы успешно зарегистрировались</h3>";
        echo "<a href='/'>Назад к форме регистрации</a>";
    }

?>
```

Простая регистрация с выводом информации

В демонстрационном примере **example_2** после регистрации выведем информацию о зарегистрированном пользователе.

Форма регистрации не изменилась.

example_2. index.html

```
<form action="reg.php" method="post">

    Логин: <input type="text" name="login" value=""><p>
    Пароль: <input type="text" name="pwd" value=""><p>
    <button>Регистрация</button>
</form>
```

А вот обработчик формы содержит теперь два запроса. Запрос на вставку данных и запрос на выборку вставленных данных для вывода в браузер.

example_2. reg.php

```
<?php

    // подключение к базе данных
    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса
    $stmt = $mysqli->prepare ('INSERT INTO `user` SET `login` = ?, `pwd` = ?');

    $stmt->bind_param ('ss', $_POST['login'], $_POST['pwd']);

    // если запрос выполнен успешно
    if ($stmt->execute ()) {
        // получаем id вставленной записи
        $id = $stmt->insert_id;
        // подготовка / привязка параметров / выполнение запроса
        $stmt = $mysqli->prepare ('SELECT * FROM `user` WHERE `id` = ?');
        $stmt->bind_param ('i', $id);
        $stmt->execute ();
        $result = $stmt->get_result ();
        $row = $result->fetch_assoc ();
        echo "Пользователь $row[login] успешно зарегистрирован";
    }
}
```

```

// вывод новой записи о пользователе из базы

$stmt = $mysqli->prepare('SELECT * FROM `user` WHERE
`id_user` = ?');

$stmt->bind_param('d', $id);

$stmt->execute();

// получение результата запроса

$result = $stmt->get_result();

$row = $result->fetch_assoc();

echo <<<HERE

Пользователь <br>

<ul>

<li>Логин: {$row['login']}</li>

<li>Пароль: {$row['pwd']}</li>

</ul>

<b>успешно зарегистрирован</b><p>

<a href='/'>Назад к форме регистрации</a>

HERE;

// если ошибка вставки

} else {

echo '<h3>Что-то пошло не так :(</h3>';

echo '<a href="/">Попробовать еще раз</a>';

};

?>

```

Регистрация с проверкой подтверждения пароля

По предыдущим мы помним, что пользователь системы – потенциальная опасность работоспособности нашего приложения. Поэтому пришло время контролировать, что пользователь вводит.

Начнем с контроля правильности ввода подтверждения пароля.

Примечание. Хорошо бы для элементов управления формы написать **валидацию**. А еще неплохо, если валидация будет и на стороне клиента, и на стороне сервера. Серверную валидацию рассматривали в одно из предыдущих .

Обратите внимание, форма регистрации содержит PHP-код.

example_3. index.php

```
<?php

if (isset($_COOKIE["error-pwd"])) {

    // вывод сообщения в браузер

    echo "<div id='msg'>Не совпадают пароли. Попробуйте еще
раз...</div>";

}

?>

<form action="reg.php" method="post">

    Логин: <input type="text" name="login" value=""><p>

    Пароль: <input type="text" name="pwd" value=""><p>

    Повторите пароль: <input type="text" name="cnfpwd"
value=""><p>

    <button>Регистрация</button>

</form>
```

example_3. reg.php

```
<?php

    // если не совпадают пароли

    if ($_POST["pwd"] != $_POST["cnfpwd"]) {

        // установим cookie

        setcookie("error-pwd", 1, 0, '/');
        header("Location:/");
        exit;

    } else {

        // удалим cookie

        setcookie("error-pwd", 0, time()-3600, '/');

    }

?>

<!-- ... -->

<?php

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('INSERT INTO `user` SET `login` = ?, `pwd` = ?');

    $stmt->bind_param('ss', $_POST['login'], $_POST['pwd']);

    // если запрос выполнен успешно

    if ($stmt->execute()) {

        // получаем id вставленной записи

        $id = $stmt->insert_id;

        // подготовка / привязка параметров / выполнение запроса

        // вывод новой записи о пользователе из базы
```

```

$stmt = $mysqli->prepare('SELECT * FROM `user` WHERE
`id_user` = ?');

$stmt->bind_param('d', $id);

$stmt->execute();

// получение результата запроса

$result = $stmt->get_result();

$row = $result->fetch_assoc();

echo <<<HERE

Пользователь <br>

<ul>

<li>Логин: {$row['login']}</li>

<li>Пароль: {$row['pwd']}</li>

</ul>

<b>успешно зарегистрирован</b><p>

<a href='/'>Назад к форме регистрации</a>

HERE;

// если ошибка вставки

} else {

    echo '<h3>Что-то пошло не так :(</h3>';
    echo '<a href="/">Попробовать еще раз</a>';
}

?>

```

Регистрация с проверкой согласия на обработку данных

Следующий шаг – проверка согласия на обработку персональных данных.

example_4. index.php

```
<?php

if (isset($_COOKIE["error-assent"])) {
    // вывод сообщения в браузер
    echo "<div id='msg'>Вы не дали согласие на обработку
персональных данных.</div>";
}

?>

<form action="reg.php" method="post">
    Логин: <input type="text" name="login" value=""><p>
    Пароль: <input type="text" name="pwd" value=""><p>
    Согласие на обработку ПДн: <input type="checkbox"
name="assent"><p>
    <button>Регистрация</button>
</form>
```

Код сценария файла **reg.php** в части вставки и вывода данных пользователя не изменился, но здесь и далее я привожу РНР-код в полном объеме.

example_4. reg.php

```
<?php

// если нет согласия на обработку

if (empty($_POST["assent"])) {
    // установим cookie
    setcookie("error-assent", 1, 0, '/');
    header("Location:/");
    exit;
} else {
```

```
// удалим cookie

setcookie("error-assent", 0, time()-3600, '/');

}

?>

<!-- ... -->

<?php

// подключение к базе данных

$mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

// подготовка / привязка параметров / выполнение запроса

$stmt = $mysqli->prepare('INSERT INTO `user` SET `login` = ?, `pwd` = ?');

$stmt->bind_param('ss', $_POST['login'], $_POST['pwd']);

// если запрос выполнен успешно

if ($stmt->execute()) {

    // получаем id вставленной записи

    $id = $stmt->insert_id;

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('SELECT * FROM `user` WHERE `id_user` = ?');

    $stmt->bind_param('d', $id);

    $stmt->execute();

    // получение результата запроса

    $result = $stmt->get_result();

    $row = $result->fetch_assoc();

    echo <<<HERE

        Пользователь <br>

        <ul>
```

```

<li>Логин: {$row['login']} </li>
<li>Пароль: {$row['pwd']} </li>
</ul>
<b>успешно зарегистрирован</b><p>
<a href='/'>Назад к форме регистрации</a>
HERE;
// если ошибка вставки
} else {
    echo '<h3>Что-то пошло не так :(</h3>';
    echo '<a href="/">Попробовать еще раз</a>';
}
?>

```

Регистрация с проверкой логина

Проверка существования идентификатора самый трудоемкий процесс, без дополнительного запроса к базе данных не обойтись.

example_5. index.php

```

<?php
if (isset($_COOKIE["error-log"])) {
    // вывод сообщения в браузер
    echo "<div id='msg'>Пользователь с таким логином
существует</div>";
}
?>
<form action="reg.php" method="post">
    Логин: <input type="text" name="login" value=""><p>

```

```

Пароль: <input type="text" name="pwd" value=""><p>
<button>Регистрация</button>
</form>

```

example_5. reg.php

```

<?php

    // проверим наличие логина в базе

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare ('SELECT `login` FROM `user` WHERE
`login` = ?');

    $stmt->bind_param ('s', $_POST['login']);

    $stmt->execute ();

    // получаем результат

    $result = $stmt->get_result();

    // если логин в базе уже есть

    if ($result->num_rows) {

        // установим cookie

        setcookie ("error-log", 1, 0, '/');

        header ("Location:/");

        exit;

    } else {

        // удалим cookie

        setcookie ("error-log", 0, time () - 3600, '/');

    }

?>

```

```

<!-- ... -->

<?php

    // проверим наличие логина в базе

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('SELECT `login` FROM `user` WHERE
    `login` = ?');

    $stmt->bind_param('s', $_POST['login']);

    $stmt->execute();

    // получаем результат

    $result = $stmt->get_result();

    // если логин в базе уже есть

    if ($result->num_rows) {

        // установим cookie

        setcookie("error-log", 1, 0, '/');

        header("Location:/");

        exit;

    } else {

        // удалим cookie

        setcookie("error-log", 0, time() -3600, '/');

    };

?>

```

Собираем проверки в одном сценарии

Соберём все проверки в одном сценарии.

example_6. index.php

```
<?php

if (isset($_COOKIE["error"])) {
    // вывод сообщений в браузер
    echo "<div id='msg'>{$_COOKIE['error']}</div>";
}

?>

<form action="reg.php" method="post">

Логин: <input type="text" name="login" value=""><p>
Пароль: <input type="text" name="pwd" value=""><p>
Повторите пароль: <input type="text" name="cnfpwd"
value="DenisDream@%-"><p>
Согласие на обработку ПДн: <input type="checkbox"
name="assent"><p>
<button>Регистрация</button>
</form>
```

example_6. reg.php

```
<?php

// проверка совпадения паролей

if ($_POST["pwd"] != $_POST["cnfpwd"]) {
    setcookie("error", 'Введенные пароли не совпадают.
Попробуйте еще раз...', 0, '/');
    header("Location:/");
    exit;
} else {

    setcookie("error", 0, time()-3600, '/');
```

```
}

// проверка наличия согласия на обработку

if (empty($_POST["assent"])) {

    setcookie("error", 'Не получено согласие на обработку
персональных данных', 0, '/');
    header("Location:/");
    exit;
} else {

    setcookie("error", 0, time()-3600, '/');
}

// проверка наличия в базе логина

$mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

// подготовка / привязка параметров / выполнение запроса

$stmt = $mysqli->prepare('SELECT `login` FROM `user` WHERE
`login` = ?');

$stmt->bind_param('s', $_POST['login']);

$stmt->execute();

$result = $stmt->get_result();

if ($result->num_rows) {

    setcookie("error", 'Пользователь с таким логином
существует', 0, '/');
    header("Location:/");
    exit;
} else {

    setcookie("error", 0, time()-3600, '/');
};

?>

<!-- ... -->
```

```
<?php

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('INSERT INTO `user` SET `login` = ?, `pwd` = ?');

    $stmt->bind_param('ss', $_POST['login'], $_POST['pwd']);

    // если запрос выполнен успешно

    if ($stmt->execute()) {

        // получаем id вставленной записи

        $id = $stmt->insert_id;

        // подготовка / привязка параметров / выполнение запроса

        $stmt = $mysqli->prepare('SELECT * FROM `user` WHERE `id_user` = ?');

        $stmt->bind_param('d', $id);

        $stmt->execute();

        // получение результата запроса

        $result = $stmt->get_result();

        $row = $result->fetch_assoc();

        echo <<<HERE

            Пользователь <br>

            <ul>

                <li>Логин: {$row['login']}</li>

                <li>Пароль: {$row['pwd']}</li>

            </ul>

            <b>успешно зарегистрирован</b><p>

            <a href='/'>Назад к форме регистрации</a>

        HERE
```

```
    HERE;  
};  
?>
```

В одной из самостоятельных работ выполним сбор ошибок в специально выделенном массиве. Это даст возможность **не прерывать сценарий** после каждой возможной ошибки пользователя.

Кодировка, шифрование, хеширование

До сих пор мы хранили пароли в **открытом виде**. Но пароли, хранящиеся в базе вашего приложения, могут быть украдены (потеряны, проданы и т.д.), после этого могут быть применены для компрометации вашего приложения.

Кодировку, шифрование и хеширование объединяет одно – преобразование информации. Разбираемся.

Кодировка

Кодирование — это способ представить данные в другом виде. Двоичный код — последовательность нулей и единиц — самый очевидный пример кодирования. Число, символ алфавита, изображение или звуковой файл — все это можно закодировать в виде единиц и нулей.

Пример кодирования данных из повседневной жизни — эмодзи. Вставляя символ эмодзи, мы кодируем свои чувства с помощью символа понятного получателю.

Шифрование

В отличие от кодирования, **шифрование** данных не просто видоизменяет информацию, но делает ее недоступной для третьих сторон. В шифровании **используются ключи**, известные только отправителю и получателю. Только сопоставив зашифрованный текст с ключом, можно получить исходную информацию.

Один из первых примеров шифрования в истории — **шифр Цезаря**, или **шифр со сдвигом**. Берем алфавит и ключ — количество позиций сдвига порядкового номера шифруемой буквы.

Например, мы собираемся зашифровать слово "кодинг". Ключ, который мы выбрали — 3. Это значит, что порядковый номер каждой буквы сдвигается на 3 символа. Осталось сопоставить обычный алфавит с шифроалфавитом и заменить буквы обычного русского алфавита на буквы шифроалфавита. В результате получится слово "нсзлрж".

Из этого примера следует, что ключ нужно хранить в секрете. Если кому-то станет известно, что наше сообщение написано при помощи шифра Цезаря со сдвигом 3, вся секретность переписки пропадет.

Поэтому перед людьми всегда стояла проблема передачи ключей.

В 1976 два математика Диффи и Хеллман придумали алгоритм шифрования DH, его модификация **DHE** сейчас используется повсеместно.

Еще один известный алгоритм придумали математики: Ривест, Шамир и Адлеман в 1977 году. По первым буквам их фамилий и был назван алгоритм **RSA**.

В RSA-шифровании участвует не один, а два ключа, один из которых секретный, приватный, хранится на устройстве, а второй, публичный, передается любому желающему. По этому принципу работают SSL-сертификаты. Внутри SSL-сертификата — публичный ключ, а его пара —

приватный, хранится на хостинг-сервере и доступен только владельцу сайта.

Классификация шифрования

Итак, формула секретной коммуникации = алгоритм + ключ. Теперь поговорим о видах шифрования в зависимости от количества ключей и не только.

Симметричное шифрование

Это самый понятный и самый старый способ зашифровать информацию. Шифр Цезаря, о котором мы говорили выше — пример симметричного шифрования. Его суть в том, что для шифрования и дешифровки используется один и тот же ключ. Если мы возьмем засекреченное слово "нсзлрж" и сопоставим его с шифроалфавитом со сдвигом 3, получится "кодинг".

Асимметричное шифрование

Асимметричное шифрование изобрели математики, о которых уже упоминалось: Ривест, Шамир и Адлеман. Его отличие в том, что для шифрования и дешифровки данных используют **два ключа: приватный и публичный**. Эти ключи математически тождественны, то есть обозначают одно и то же число. К примеру, публичный ключ — это число "15", а приватный ключ — выражение " 5×3 ".

Для SSL-сертификатов длина публичного и приватного ключа огромна и начинается от 2048 бит.

Публичный ключ находится внутри SSL-сертификата и любой браузер получает его от сервера при подключении к сайту. Приватный ключ хранится на сервере и никогда не передается по сети.

Если браузер хочет отправить серверу зашифрованное сообщение, он берет публичный ключ из сертификата, шифрует им данные и

отправляет серверу. Дешифровать информацию сможет только тот, у кого есть математическое тождество публичного ключа, а именно приватный ключ. Сервер берет свой приватный ключ и получает исходное сообщение.

То есть то, что зашифровано публичным ключом, дешифровать можно только приватным, и наоборот.

Обратимое шифрование

Обратимое шифрование предполагает, что из зашифрованной последовательности символов можно получить **исходное сообщение**, как в шифре Цезаря.

Необратимое шифрование

При необратимом шифровании после того, как данные зашифрованы, **исходное сообщение получить невозможно**. Такие алгоритмы не используется с целью секретной переписки. У них другая роль, например, служить эталоном нетронутости информации. По-другому **необратимое шифрование называют хешированием**.

Хеширование

Итак, разница между хешированием и шифрованием — зашифрованное сообщение можно дешифровать и получить исходный текст. С хешированным так не получится.

Хеширование данных — приведение любого блока информации к **строке с фиксированной длиной**. Эта строка — **хеш**, или **хеш-сумма**. Хеш файла или сообщения — эталон, который может подтвердить, что информация не была повреждена или подменена в процессе передачи.

Хеширование данных используется на сайтах, чтобы проверить правильность паролей, которые мы вводим для входа в личный кабинет.

Владельцы сайтов из соображений безопасности не хранят пароли пользователей в открытом виде.

Важно. Когда клиент отправляет сайту пароль в первый раз, сайт сохраняет не сам пароль, а его **хеш (хеш-сумму)**. Дальше при каждом новом логине пользователя, сайт сверяет **сохраненный хеш с хеш-суммой только что введенного пароля**.

Если эти значения не совпадают, мы видим сообщение "неверный логин/пароль" (или похожее).

Подводя итоги

Кодирование — представление информации в другом виде: перевод числа из десятичной системы в двоичную, сохранение изображений в памяти компьютера и т.д.

Шифрование — представление информации в другом виде с целью ее засекретить: шифр Цезаря.

Хеширование — необратимое приведение информации в другой вид с целью получения *уникального отпечатка*, по которому можно проверить целостность данных.

Практика регистрации с хешированием

Хеширование паролей является одним из самых основных соображений безопасности, которые необходимо сделать, при разработке приложения, принимающего пароли от пользователей. Без хеширования, пароли, хранящиеся в базе вашего приложения, могут быть украдены, например, если ваша база данных была скомпрометирована, а затем немедленно могут быть применены для компрометации не только вашего приложения, но и аккаунтов ваших

пользователей на других сервисах, если они не используют уникальных паролей.

Применяя хеширующий алгоритм к пользовательским паролям перед сохранением их в своей базе данных, вы делаете невозможным разгадывание оригинального пароля для атакующего вашу базу данных, в то же время сохраняя возможность сравнения полученного хеша с оригинальным паролем.

Важно. Хеширование паролей защищает их только от компрометирования в вашей **базе данных**, но не обязательно от вмешательства вредоносного кода в вашем **приложении**.

Хеширование пароля. Функция `crypt()`

`crypt`

`crypt()` — необратимое **хеширование строки**. Функция небезопасна для обработки данных в двоичной форме.

Описание

```
crypt($string, $salt);
```

`crypt()` возвращает **хешированную строку**, полученную с помощью стандартного алгоритма UNIX, основанного на DES или другого алгоритма.

Параметры

- **string** – хешируемая строка.
- **salt** – параметр с солью, на которой будет основано хеширование (соль). Если не указан, поведение определяется по наличию реализованных алгоритмов в системе и может привести к

неожиданным результатам. Отсутствие параметра **salt** выдаёт ошибку уровня E_NOTICE.

Возвращаемые значения

Возвращает **хешированную строку** или строку короче 13 символов, гарантированно отличающуюся от соли в случае возникновения ошибки.

Вид хеширования определяется переданным аргументом **salt** (соль).

Если соль не указана, будет **автоматически сгенерирована стандартная случайная** двухсимвольная или двенадцатисимвольная **соль**, в зависимости от доступности алгоритма MD5 в crypt()

Предупреждение. Функция **crypt()** создаёт **слабый хеш** без параметра **salt**.

Убедитесь, что используете достаточно сложную соль для лучшей безопасности.

Примечание. Альтернатива **crypt()** – функция **password_hash()**.
Функция **password_hash()** создает сложный хеш – генерирует сложную соль и применяет правильно количество раундов хеширования автоматически.

Функция **password_hash()** является простой обёрткой над **crypt()** и совместима с существующими хешами паролей. Поэтому приветствуется использование **password_hash()** (рассмотрим далее).

В демонстрационном примере **example_7** рассмотрим создание хеша пользовательского пароля с использованием функции **crypt()**.

example_7. index.html

```
<form action="reg.php" method="post">  
Логин: <input type="text" name="login" value="DenisDream"><p>
```

```

Пароль: <input type="text" name="pwd"
value="DenisDream@%"><p>
<button>Регистрация</button>
</form>

```

Обратите внимание, в качестве **соли** для функции используем **логин пользователя**.

example_7. reg.php

```

<?php
    // подключение к базе данных
    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса
    $stmt = $mysqli->prepare ('INSERT INTO `user` SET `login` = ?,
    `pwd` = ?');

    // хешируем пароль, ключ (соль) хеша - логин пользователя
    $pwd = crypt($_POST['pwd'], $_POST["login"]);

    $stmt->bind_param('ss', $_POST['login'], $pwd);

    // если запрос выполнен успешно
    if ($stmt->execute ()) {
        // получаем id вставленной записи
        $id = $stmt->insert_id;

        // подготовка / привязка параметров / выполнение запроса
        $stmt = $mysqli->prepare ('SELECT * FROM `user` WHERE
        `id_user` = ?');

        $stmt->bind_param('d', $id);

        $stmt->execute ();

        // получение результата запроса
        $result = $stmt->get_result ();

```

```
$row = $result->fetch_assoc();  
  
echo <<<HERE  
  
Пользователь <br>  
  
<ul>  
  
<li>Логин: {$row['login']}</li>  
  
<li>Пароль: {$row['pwd']}</li>  
  
</ul>  
  
<b>успешно зарегистрирован</b><p>  
  
<a href='/'>Назад к форме регистрации</a>  
  
HERE;  
  
};  
  
?>
```

Соль на основе пользовательского логина вряд ли можно назвать достаточно сложной. А поэтому идем далее.

str_shuffle

str_shuffle() — переставляет символы в строке случайным образом.

Описание

str_shuffle(\$string);

str_shuffle() перемешивает символы в строке. Выбирается одна возможная перестановка из всех возможных.

Примечание. Функция **не создаёт криптографически защищённые значения** и не должна использоваться для криптографических целей или целей, требующих, чтобы возвращаемые значения были недоступны для разгадывания.

Если нужна **криптографически безопасная случайная последовательность**, для простых сценариев существуют функции **random_int()** и **random_bytes()** с криптографически безопасного генератора псевдослучайных чисел (CSPRNG).

Параметры

- **string** – входная строка.

Возвращаемые значения

Возвращает перемешанную строку.

В демонстрационном примере **example_8** реализуем хеширование функцией **crypt()** с солью, на основе последовательности случайных символов.

example_8. index.php

```
<form action="reg.php" method="post">

    Логин: <input type="text" name="login" value=""><p>
    Пароль: <input type="text" name="pwd" value=""><p>

    <button>Регистрация</button>

</form>
```

example_8. reg.php

```
<?php

    // подключение к базе данных

    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('INSERT INTO `user` SET `login` = ?,
        `pwd` = ?, `salt` = ?') ;
```

```

// набор символов для получения ключа хеширования (соли)

$chars = '0123456789abcdefghijklmnopqrstuvwxyz';

// получаем случайные 10 символов

$salt = substr(str_shuffle($chars), 0, 10);

// хешируем пароль, ключ (соль) хеша - случайные символы

$pwd = crypt($_POST['pwd'], $salt);

$stmt->bind_param('sss', $_POST['login'], $pwd, $salt);

// если запрос выполнен успешно

if ($stmt->execute()) {

    // получаем id вставленной записи

    $id = $stmt->insert_id;

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('SELECT * FROM `user` WHERE
`id_user` = ?');

    $stmt->bind_param('d', $id);

    $stmt->execute();

    // получение результата запроса

    $result = $stmt->get_result();

    $row = $result->fetch_assoc();

    echo <<<HERE

        Пользователь <br>

        <ul>

            <li>Логин: {$row['login']}</li>

            <li>Пароль: {$row['pwd']}</li>

        </ul>

        <b>успешно зарегистрирован</b><p>

        <a href='/'>Назад к форме регистрации</a>

```

```
    HERE;  
};  
?>
```

Хеширование пароля. Функция **password_hash()**

Функция **password_hash()** создает сложный хеш – генерирует сложную соль и применяет правильно количество раундов хеширования автоматически.

Функция **password_hash()** является обёрткой над **crypt()** и совместима с существующими хешами паролей. Использование функции **password_hash()** является предпочтительным способом создания хеша.

password_hash

password_hash — создаёт хеш пароля.

Описание

```
password_hash($password, $algo, $options = []);
```

Функция **password_hash()** создаёт хеш пароля, используя сильный необратимый алгоритм хеширования. Функция поддерживает несколько видов алгоритмов хеширования.

Параметры

- **password** – пользовательский пароль.
- **algo** – константа, обозначающая используемый алгоритм хеширования пароля. В качестве алгоритма хеширования по умолчанию используется константа **PASSWORD_DEFAULT**. Алгоритм может быть изменён в новых версиях PHP на более надёжный.

- **options** – ассоциативный массив с опциями. Одна из опций – **алгоритмическая стоимость**, которую следует использовать. Если не задано, то будет использована стандартная стоимость, и соль будет сгенерирована автоматически.

Возвращаемые значения

Возвращает **хешированный пароль**.

Выбранный **алгоритм, стоимость и соль** будут возвращены как часть хеша. Таким образом, информация, необходимая для проверки хеша, будет в него включена. Это позволит функции **password_verify()** проверять хеш без отдельного хранения информации о соли.

Важно. Функция **password_hash()** не требует хранения информации о используемой соли (**salt**).

В следующем примере рассмотрим хеширование функцией **password_hash()**.

example_9. index.php

```
<form action="reg.php" method="post">

    Логин: <input type="text" name="login" value=""><p>
    Пароль: <input type="text" name="pwd" value=""><p>
    <button>Регистрация</button>

</form>
```

example_9. reg.php

```
<?php
    // подключение к базе данных
```

```

$mysqli = new mysqli ('localhost', 'root', '', 'db_reg');

// подготовка / привязка параметров / выполнение запроса

$stmt = $mysqli->prepare('INSERT INTO `user` SET `login` = ?,
`pwd` = ?');

// хешируем пароль

$pwd = password_hash($_POST['pwd'], PASSWORD_DEFAULT);

$stmt->bind_param('ss', $_POST['login'], $pwd);

// если запрос выполнен успешно

if ($stmt->execute()) {

    // получаем id вставленной записи

    $id = $stmt->insert_id;

    // подготовка / привязка параметров / выполнение запроса

    $stmt = $mysqli->prepare('SELECT * FROM `user` WHERE
`id_user` = ?');

    $stmt->bind_param('d', $id);

    $stmt->execute();

    // получение результата запроса

    $result = $stmt->get_result();

    $row = $result->fetch_assoc();

    echo <<<HERE

        Пользователь <br>

        <ul>

            <li>Логин: {$row['login']} </li>

            <li>Пароль: {$row['pwd']} </li>

        </ul>

        <b>успешно зарегистрирован</b><p>

        <a href='/'>Назад к форме регистрации</a>

```

```
    HERE;  
};  
?>
```

Регистрация с навигацией по сайту

В демонстрационном примере **example_10** выполним регистрацию и сохраним состояние успешной регистрации для навигации по страницам сайта.

example_10. index.php

```
<form action="/reg.php" method="post">  
  
Логин: <input type="text" name="login" value=""><p>  
  
Пароль: <input type="text" name="pwd" value=""><p>  
  
<button>Регистрация</button>  
  
</form>  
  
<p><a href="/reg.php">Пропустить регистрацию</a></p>
```

example_10. reg.php

```
<?php  
  
if (isset($_POST['login'])) {  
  
    // подключение к базе данных  
  
    $mysqli = new mysqli ('localhost', 'root', '', 'db_reg');  
  
    // подготовка / привязка параметров / выполнение запроса  
  
    $stmt = $mysqli->prepare ('INSERT INTO `user` SET `login`  
    = ?, `pwd` = ?');  
  
    // хешируем пароль  
  
    $pwd = password_hash($_POST['pwd'], PASSWORD_DEFAULT);
```

```

$stmt->bind_param('ss', $_POST['login'], $pwd);

// если запрос выполнен успешно

if ($stmt->execute()) {

    // получаем id вставленной записи

    $id = $stmt->insert_id;

    // подготовка / привязка параметров / выполнение
    // запроса

    $stmt = $mysqli->prepare('SELECT * FROM `user` WHERE
        `id_user` = ?');

    $stmt->bind_param('d', $id);

    $stmt->execute();

    // получение результата запроса

    $result = $stmt->get_result();

    $row = $result->fetch_assoc();

    // устанавливаем cookie для директории admin

    setcookie("auth", 1, 0, '/admin/');

    $msg = '<h3>Вы успешно зарегистрировались</h3>';

}

?>

<!-- ... -->

<?php if (isset($msg)) echo $msg; ?>

<p><a href="/admin/">Перейти в личный кабинет</a></p>
<p><a href="/page/">Интересная страница сайта</a></p>

```

Файл **admin/index.html** доступен для просмотра только зарегистрированным пользователям.

example_10. admin/index.php

```
<?php

    // проверяем наличие у пользователя регистрации

    if(isset($_COOKIE['auth'])) {

        echo '<h2>Добро пожаловать! ; )</h2>';

    } else {

        echo '<h2>Контент только для зарегистрированных
пользователей :(</h2>';

        echo '<p><a href="/">Регистрация</a></p>';

    }

?>
```

Файл **page/index.html** доступен для просмотра всем пользователям ресурса.

example_10. page/index.php

```
<?php

    // проверяем наличие у пользователя регистрации

    if(isset($_COOKIE['auth'])) {

        // cookie для этой директории недоступен

        echo '<h2>Вот это вряд ли... ; )</h2>';

    } else {

        echo '<h2>Здравствуйте!</h2>';

    }

?>
```

Подробнее о навигации по страницам сайта [следующе](#).

