Содержание

Фильтрация и валидация форм

- Во время разработки, выводите ошибки на экран, на продакшене скрывайте вывод, вместо этого ведите логи. Ошибки не враги!
- Проверяйте белые списки а не черные.
- Основной принцип - что не разрешено, то запрещено.

Валидация данных пришедших от пользователя очень важна. Оптимальным вариантом является использование встроенных функции PHP специально введенных в 5.2 версии именно для этих целей.

filter_has_var	*Проверяет существование переменной указанного типа*
filter_id	*Проверяет существование переменной указанного типа*
filter_input_array	*Принимает несколько переменных извне PHP и, при необходимости, фильтрует их*
filter_input	*Принимает переменную извне PHP и, при необходимости, фильтрует ее*
filter_list	*Возвращает список всех поддерживаемых фильтров*
filter_var_array	*Принимает несколько переменных и, при необходимости, фильтрует их*
filter_var	*Фильтрует переменную с помощью определенного фильтра*

Они позволяют выполнить валидацию данных и обезопасить их от возможных вредоносных конструкций. Существует два основных типа фильтрации: проверка и очистка:

Проверка используется для определения соответствия данных определённым критериям. Например, применение FILTER_VALIDATE_EMAIL позволяет проверить, являются ли введённые данные адресом email, однако, сами данные при этом останутся нетронутыми.

Очистка используется для извлечения из данных нежелательных конструкций. Например, применение FILTER_SANITIZE_EMAIL удалит все символы, которые не должен содержать email-адрес. То есть, проверки данных не происходит.

Поведение этих функций зависит от установок в php.ini.

Параметры конфигурации Filter

Имя	По умолчанию	Меняемо	Список изменений
filter.default	"unsafe_raw"	PHP_INI_PERDIR	PHP_INI_ALL в filter <= 0.9.4. Доступна с PHP 5.2.0.
filter.default_flags	NULL	PHP_INI_PERDIR	PHP_INI_ALL в filter <= 0.9.4. Доступна с PHP 5.2.0.

---

filter_input

```
mixed filter_input ( int $type , string $variable_name [, int $filter = FILTER_DEFAULT [, mixed $options ]] )
```

$type - *константы* ***INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV***

$variable_name - имя получаемой переменной

$filter - один из фильтров по умолчанию используется FILTER_DEFAULT, который равнозначен FILTER_UNSAFE_RAW

$options - ассоциативный массив параметров либо логическое ИЛИ флагов. Если фильтр принимает параметры, флаги могут быть указаны в элементе массива "flags".

Значение запрашиваемой переменной в случае успеха, FALSE, если фильтрация завершилась неудачей, или NULL, если переменная variable_name не определена. Если установлен флаг FILTER_NULL_ON_FAILURE, функция возвращает FALSE, если переменная не определена и NULL, если фильтрация завершилась неудачей.

```
<form action="" method="POST">
    <label>E-mail:</label>
    <input type="email" name="email">

    <label>Имя:</label>
    <input type="text" name="name">

    <input type="submit" value="Отправить" />
</form>
```

```
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
$name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
```

filter_input_array

```
mixed filter_input_array ( int $type [, mixed $definition [, bool $add_empty = true ]] )
```

$type - *константы* ***INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV***

$definition - массив, определяющий аргументы. Допустимый ключ - строка string, содержащая имя переменной, и допустимое значение - или тип filter, или массив array, при

необходимости определяющий фильтр, флаги и параметры. Если значение является массивом, допустимыми ключами являются filter, который определяет (тип фильтра), flags, который определяет любые флаги, применяемые к фильтру и options, который определяет любые параметры, применяемые к фильтру.

$add_empty - добавляет в результат отсутствующие ключи со значением NULL (с версии 5.4.0)

```
<form action="" method="POST">
    <label>E-mail:</label>
    <input type="email" name="email">

    <label>Имя:</label>
    <input type="text" name="name">

    <input type="submit" value="Отправить" />
</form>
```

```
$fields = [
    'email' => FILTER_SANITIZE_EMAIL,
    'name' => FILTER_SANITIZE_STRING
];

$inputs = filter_input_array(INPUT_POST, $fields);
```

Сторонние библиотеки:

http://htmlpurifier.org/ - очищает любой html код от всех вредоносных, невалидных, запрещенных (вашей конфигурацией) частей кода

Безопасность работы с БД

Внедрение SQL-кода (англ. SQL injection) — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

```
$id = $_POST['id']; // 0 OR 1 = 1'

$pdo = new PDO('mysql:host=localhost;dbname=my_db', 'root', 'password');
$stmt = $pdo->query("SELECT * FROM users WHERE id = $id");
```

Использование UNION:

Язык SQL позволяет объединять результаты нескольких запросов при помощи оператора UNION. Это предоставляет злоумышленнику возможность получить несанкционированный доступ к данным.

```
$id = $_POST['id']; // -1 UNION SELECT 1,username,password,1 FROM admin

$pdo = new PDO('mysql:host=localhost;dbname=my_db', 'root', 'password');
$stmt = $pdo->query("SELECT * FROM users WHERE id = $id");
```

Экранирование хвоста запроса:

Зачастую, SQL-запрос, подверженный данной уязвимости, имеет структуру, усложняющую или препятствующую использованию union.

```
$id = $_POST['id']; // -1 UNION SELECT password FROM admin/*
```

Расщепление SQL-запроса:

Для разделения команд в языке SQL используется символ ; (точка с запятой), внедряя этот символ в запрос, злоумышленник получает возможность выполнить несколько команд в одном запросе, однако не все диалекты SQL поддерживают такую возможность.

```
$id = $_POST['id']; // -1;INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');
```

Решение:

1) Предварительная валидация

```php
if (filter_input(INPUT_POST, 'id', FILTER_VALIDATE_INT))
{
    $id = filter_input(INPUT_POST, 'id', FILTER_SANITIZE_NUMBER_INT);
}
```

2) PDO - bindValue() или bindParam()

```php
$id = filter_input(INPUT_POST, 'id', FILTER_SANITIZE_NUMBER_INT);
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = :id");
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
```

3) Mysqli - bind_param()

```php
$stmt = $mysqli->prepare("SELECT * FROM users WHERE id = ?");
$stmt->bind_param('i', $id);
$id = filter_input(INPUT_POST, 'id', FILTER_SANITIZE_NUMBER_INT);
```

4) Явно приводить к типу, например если переменная - целое число можно так:

```php
$id = (int) $id;
$stmt = $pdo->query("SELECT * FROM users WHERE id = $id");
```

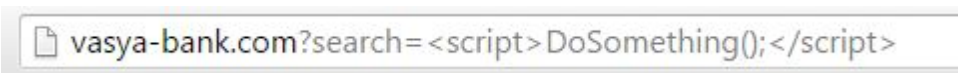
Защита от XSS

XSS - (англ. Cross Site Scripting — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «внедрение кода». *Не путать с CSS*. (OWASP XSS)

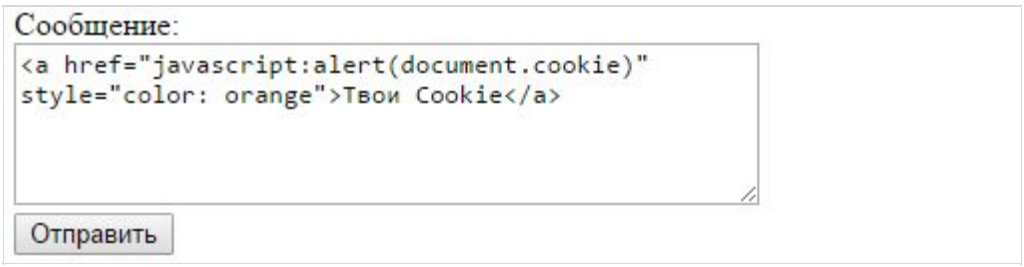
XSS-атака может быть направлена на кражу личных данных, таких как cookies, паролей и т.д., или внедрять код скриптов и ссылок на страницы сайта.
XSS находится на третьем месте в рейтинге ключевых рисков Web-приложений согласно OWASP 2013

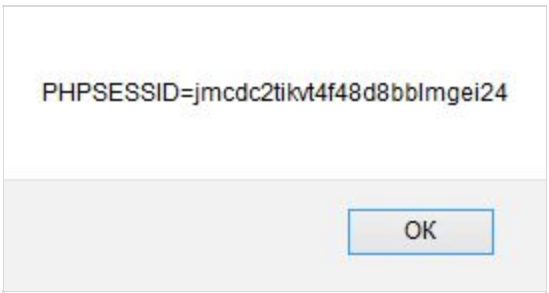
Не существует чёткой классификации, но большинство экспертов различают по крайней мере два типа XSS: «отраженные» («reflected XSS» или «Type 1») и «хранимые» («stored XSS» или «Type 2»):

1. Отражённые (Непостоянные) - атака, основанная на отражённой уязвимости, на сегодняшний день является самой распространенной XSS-атакой. Эти уязвимости появляются, когда данные, предоставленные веб-клиентом, чаще всего в параметрах HTTP-запроса или в форме HTML, исполняются непосредственно серверными скриптами для синтаксического анализа и отображения страницы результатов для этого клиента, без надлежащей обработки. Отражённая XSS-атака срабатывает, когда пользователь переходит по специально подготовленной ссылке. Отражённые атаки, как правило, рассылаются по электронной почте или размещаются на Web-странице. URL приманки не вызывает подозрения, указывая на надёжный сайт, но содержит вектор XSS. Если доверенный сайт уязвим к вектору XSS, то переход по ссылке может привести к тому, что браузер жертвы начнет выполнять встроенный скрипт.



2. Хранимый XSS является наиболее разрушительным типом атаки. Хранимый XSS возможен, когда злоумышленнику удается внедрить на сервер вредоносный код, выполняющийся в браузере каждый раз при обращении к оригинальной странице. Классическим примером этой уязвимости являются форумы, на которых разрешено оставлять комментарии в HTML формате без ограничений, а также другие сайты Веб 2.0 (блоги, вики, имиджборд), когда на сервере хранятся пользовательские тексты и рисунки. Скрипты вставляются в эти тексты и рисунки. Например имеется форма отправки сообщений, сообщения сохраняются в базу без обработки и выводятся без обработки - в результате все пользователи которые будут читать сообщение будут подвержены атаке.





3. XSS в DOM-модели возникает на стороне клиента во время обработки данных внутри JavaScript сценария. Данный тип XSS получил такое название, поскольку реализуется через DOM (Document Object Model) — не зависящий от платформы и языка программный интерфейс, позволяющий программам и сценариям получать доступ к содержимому HTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов. При некорректной фильтрации возможно модифицировать DOM атакуемого сайта и добиться выполнения JavaScript-кода в

контексте атакуемого сайта.

```
<body>
<script>document.write(location.href);</script>
</body>
```

- *Используйте экранирование входных\выходных данных: htmlspecialchar(), htmlentities().*
- *filter_input(), filter_var() и т.д. с параметрами SANITIZE*
- *strip_tags()*
- *Использование сторонних библиотек например http://htmlpurifier.org/*
- *Указывать кодировку на всех страницах (если злоумышленник подставит свою кодировку например - UTF-7, то он сможет обойти фильтрацию '<' и '"')*
- *HttpOnly (session.cookie_httponly и session_set_cookie_params ()) чтобы куки не доступными через JavaScript*
- *Content Security Policy (CSP) - заголовок, позволяющий объявлять источники для подгрузки различных данных: JS, CSS и т.д.*

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
    $message = isset($_POST['message']) ? $_POST['message'] : '';

    $msgs['strip_tags'] = strip_tags($message);
    $msgs['htmlentities'] = htmlentities($message, ENT_QUOTES, 'UTF-8');
    $msgs['htmlspecialchars'] = htmlspecialchars($message, ENT_QUOTES, 'UTF-8');
    $msgs['filter_var'] = filter_var($message, FILTER_SANITIZE_ENCODED);
    $msgs['filter_input'] = filter_input(INPUT_POST, 'message', FILTER_SANITIZE_ENCODED);
}
```

Безопасная загрузка файлов

Настройки:
file_uploads - Разрешать или не разрешать закачивание файлов
upload_max_filesize - Максимальный размер закачиваемого файла
post_max_size - Устанавливает максимально допустимый размер данных, отправляемых методом POST
max_file_uploads - Максимально разрешенное количество одновременно закачиваемых файлов
upload_tmp_dir - Временная директория, используемая для хранения файлов во время закачивания
memory_limit - Эта директива задает максимальный объем памяти в байтах, который разрешается использовать скрипту

```
<form action="" method="POST" enctype="multipart/form-data">
    <input type="file" name="myfile" /><br>
    <input type="submit" value="Загрузить" />
</form>
```

```
// Оригинальное имя файла на компьютере клиента.
$_FILES['myfile']['name'];

// Mime-тип файла, в случае, если браузер предоставил такую информацию.
$_FILES['myfile']['type'];

// Размер в байтах принятого файла.
$_FILES['myfile']['size'];

// Временное имя, с которым принятый файл был сохранен на сервере.
$_FILES['myfile']['tmp_name'];

// Код ошибки, которая может возникнуть при загрузке файла.
$_FILES['myfile']['error'];
```

Проверки:
1) Проверка расширения файла (Например загрузка картинок)

```php
$file = empty($_FILES['myfile']) ? null : $_FILES['myfile'];
$ext = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
$img_exts = ['jpeg', 'jpg', 'png', 'gif'];

if ( ! in_array($ext, $img_exts))
{
    die('Это не картинка!!!');
}
```

2) Проверка на ошибки (код ошибок)

```php
$file = empty($_FILES['myfile']) ? null : $_FILES['myfile'];

if ($file['error'] !== UPLOAD_ERR_OK)
{
    die('Ошибка загрузки');
}
```

3) Проверка размера

```php
function convToByte($size)
{
    $size = strtoupper(trim($size));
    $length = strlen($size) - 1;

    switch ($size[$length])
    {
        case 'G':
            $size *= 1024;
        case 'M':
            $size *= 1024;
        case 'K':
            $size *= 1024;
    }

    return $size;
}
```

```php
$file = empty($_FILES['myfile']) ? null : $_FILES['myfile'];

$upload_max_size = convToByte(ini_get('upload_max_filesize')); // 2M->2097152
$post_max_size = convToByte(ini_get('post_max_size')); // 8M->8388608
```

```php
if ($file['size'] == 0)
{
    die('Пустой файл');
}

if ($file > $upload_max_size || $file > $post_max_size)
{
    die('Файл слишком большой');
}
```

4) Генерировать новые имена файлов и перенос (move_uploaded_file())

```php
$uploads_dir = '/uploads';
$file = empty($_FILES['myfile']) ? null : $_FILES['myfile'];
$ext = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
$new_name = uniqid() .'-'. time() .'.'. $ext;

move_uploaded_file($file['tmp_name'], "$uploads_dir/$new_name");
```