What's the best method for sanitizing user input with PHP?

KEEP YOUR CLOUD TO YOURSELF.
Private servers. Private clouds. Private network. Private. Private. Private.
SOFTLAYER an IBM Company
LEARN MORE >

Is there a catchall function somewhere that works well for sanitizing user input for sql injection and XSS attacks, while still allowing certain types of html tags?

php security xss sql-injection

edited Sep 24 '08 at 20:26
Till
15.8k 3 40 78

asked Sep 24 '08 at 20:20
UltimateBrent
6,823 7 31 44

19 Nowadays, to avoid sql injection, use PDO or MySQLi. – Francisco Presencia Apr 8 '13 at 16:00

21 Using PDO or MySQLi isn't enough. If you build your SQL statements with untrusted data, like `select * from users where name='$name'` , then it doesn't matter if you use PDO or MySQLi or MySQL. You are still in danger. You must use parametrized queries or, if you must, use escaping mechanisms on your data, but that is much less preferable. – Andy Lester Dec 20 '13 at 17:01

5 @AndyLester Are you implying that someone uses PDO without prepared statements? :) – user1537415 Mar 30 '14 at 14:20

18 I'm saying that "Use PDO or MySQLi" is not information enough to explain to novices on how to safely use them. You and I know that prepared statements matter, but I do not assume that everyone who reads this question will know it. That is why I added the explicit instructions. – Andy Lester Mar 30 '14 at 22:10

8 Andy's comment is entirely valid. I converted my mysql website to PDO recently thinking that I was now somehow safe from injection attacks. It was only during the process I realised that some of my sql statements were still built using user input. I then fixed that using prepared statements. To a complete novice, it's not fully clear that there is a distinction as many experts throw out the comment about using PDO but don't specify the need for prepared statements. The assumption being that this is obvious. But not to a novice. – GhostRider May 25 '14 at 8:15

11 Answers

It's a common misconception that user input can be filtered. PHP even has a (now deprecated) "feature", called magic-quotes, that builds on this idea. It's nonsense. Forget about filtering (Or cleaning, or whatever people call it).

What you should do, to avoid problems is quite simple: Whenever you embed a string within foreign code, you must escape it, according to the rules of that language. For example, if you embed a string in some SQL targeting MySql, you must escape the string with MySql's function for this purpose (`mysqli_real_escape_string`).

Another example is HTML: If you embed strings within HTML markup, you must escape it with `htmlspecialchars` . This means that every single `echo` or `print` statement should use `htmlspecialchars` .

A third example could be shell commands: If you are going to embed strings (Such as arguments) to external commands, and call them with `exec` , then you must use `escapeshellcmd` and `escapeshellarg` .

And so on and so forth ...

The *only* case where you need to actively filter data, is if you're accepting preformatted input. Eg. if you let your users post HTML markup, that you plan to display on the site. However, you should be wise to avoid this at all cost, since no matter how well you filter it, it will always be a potential security hole.

edited Jan 11 '14 at 23:40
peter.petrov
20.5k 3 14 35

answered Sep 24 '08 at 22:30
troelskn
58.6k 17 79 116

26 Thanks, that was a really clear explanation. A lot better than just "no!" :) – UltimateBrent Sep 24 '08 at 23:34

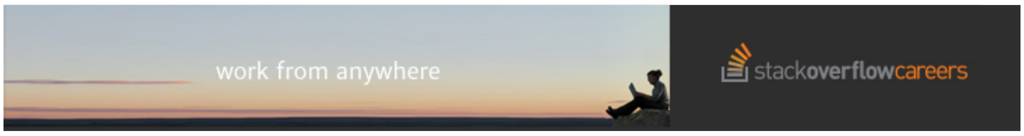
147 "This means that every single echo or print statement should use htmlspecialchars" - of course, you mean "every ... statement outputting user input"; htmlspecialchars()-ifying "echo 'Hello, world!';" would be

crazy ;) – Bobby Jack Oct 20 '08 at 13:32

9 Excellent concise answer! I cringe when I hear about sanitizing input with no regard to the context. – Cory House May 23 '09 at 3:59

6 There's one case where I think filtering is the right solution: UTF-8. You don't want invalid UTF-8 sequences all over your application (you might get different error recovery depending on code path), and UTF-8 can be filtered (or rejected) easily. – porneL Sep 9 '09 at 21:33

6 @jbyrd - no, LIKE uses a specialised regexp language. You will have to escape your input string twice - once for the regexp and once for the mysql string encoding. It's code within code within code. – troelskn Oct 29 '11 at 20:02



Do not try to prevent SQL injection by sanitizing input data.

Instead, **do not allow data to be used in creating your SQL code**. Use parameterized SQL that uses bound variables. It is the only way to be guaranteed against SQL injection.

Please see my website http://bobby-tables.com/ for more about preventing SQL injection.

edited Dec 20 '13 at 16:59 answered Oct 9 '08 at 6:28
Andy Lester
39.6k 10 55 107

2 Or visit the official documentation and learn PDO and prepared statements. Tiny learning curve, but if you know SQL pretty well, you'll have no trouble adapting. – a coder Nov 13 '14 at 2:49

No. You can't generically filter data without any context of what it's for. Sometimes you'd want to take a SQL query as input and sometimes you'd want to take HTML as input.

You need to filter input on a whitelist -- ensure that the data matches some specification of what you expect. Then you need to escape it before you use it, depending on the context in which you are using it.

The process of escaping data for SQL - to prevent SQL injection - is very different from the process of escaping data for (X)HTML, to prevent XSS.

edited Dec 23 '12 at 14:26 answered Sep 24 '08 at 20:24
Daniel Papasian
10.9k 5 19 26

11 +1 "You can't generically filter data without any context of what it's for." Should be repeated more often. – Steve May 8 '12 at 6:07

PHP has the new nice filter_input functions now, that for instance liberate you from finding 'the ultimate e-mail regex' now that there is a built-in FILTER_VALIDATE_EMAIL type

My own filter class (uses javascript to highlight faulty fields) can be initiated by either an ajax request or normal form post. (see the example below)

```
/**
 *  Pork.FormValidator
 *  Validates arrays or properties by setting up simple arrays.
 *  Note that some of the regexes are for dutch input!
 *  Example:
 *
 *  $validations = array('name' => 'anything','email' => 'email','alias' =>
'anything','pwd'=>'anything','gsm' => 'phone','birthdate' => 'date');
 *  $required = array('name', 'email', 'alias', 'pwd');
 *  $sanatize = array('alias');
 *
 *  $validator = new FormValidator($validations, $required, $sanatize);
 *
 *  if($validator->validate($_POST))
 *  {
 *      $_POST = $validator->sanatize($_POST);
 *      // now do your saving, $_POST has been sanatized.
 *      die($validator->getScript()."<script type='text/javascript'>alert('saved
changes');</script>");
 *  }
 *  else
 *  {
 *      die($validator->getScript());
 *  }
 *
```

```
  * To validate just one element:
  * $validated = new FormValidator()->validate('blah@bla.', 'email');
  *
  * To sanatize just one element:
  * $sanatized = new FormValidator()->sanatize('<b>blah</b>', 'string');
  *
  * @package pork
  * @author SchizoDuckie
  * @copyright SchizoDuckie 2008
  * @version 1.0
  * @access public
  */
class FormValidator
{
    public static $regexes = Array(
            'date' => "^[0-9]{1,2}[-/][0-9]{1,2}[-/][0-9]{4}\$",
            'amount' => "^[-]?[0-9]+\$",
            'number' => "^[-]?[0-9,]+\$",
            'alfanum' => "^[0-9a-zA-Z ,.-_\\s\?\!]+\$",
            'not_empty' => "[a-z0-9A-Z]+",
            'words' => "^[A-Za-z]+[A-Za-z \\s]*\$",
            'phone' => "^[0-9]{10,11}\$",
            'zipcode' => "^[1-9][0-9]{3}[a-zA-Z]{2}\$",
            'plate' => "^([0-9a-zA-Z]{2}[-]){2}[0-9a-zA-Z]{2}\$",
            'price' => "^[0-9.,]*(([.,][-])|([.,][0-9]{2}))?\$",
            '2digitopt' => "^\d+(\,\d{2})?\$",
            '2digitforce' => "^\d+\,\d\d\$",
            'anything' => "^[\d\D]{1,}\$"
    );
    private $validations, $sanatations, $mandatories, $errors, $corrects, $fields;


    public function __construct($validations=array(), $mandatories = array(), $sanatations
= array())
    {
        $this->validations = $validations;
        $this->sanatations = $sanatations;
        $this->mandatories = $mandatories;
        $this->errors = array();
        $this->corrects = array();
    }

    /**
     * Validates an array of items (if needed) and returns true or false
     *
     */
    public function validate($items)
    {
        $this->fields = $items;
        $havefailures = false;
        foreach($items as $key=>$val)
        {
            if((strlen($val) == 0 || array_search($key, $this->validations) === false) &&
array_search($key, $this->mandatories) === false)
            {
                $this->corrects[] = $key;
                continue;
            }
            $result = self::validateItem($val, $this->validations[$key]);
            if($result === false) {
                $havefailures = true;
                $this->addError($key, $this->validations[$key]);
            }
            else
            {
                $this->corrects[] = $key;
            }
        }

        return(!$havefailures);
    }

    /**
     *
     *  Adds unvalidated class to thos elements that are not validated. Removes them from
classes that are.
     */
    public function getScript() {
        if(!empty($this->errors))
        {
            $errors = array();
            foreach($this->errors as $key=>$val) { $errors[] = "'INPUT[name={$key}]'"; }

            $output = '$$('.implode(',', $errors).').addClass("unvalidated");';
            $output .= "new FormValidator().showMessage();";
        }
        if(!empty($this->corrects))
        {
            $corrects = array();
            foreach($this->corrects as $key) { $corrects[] = "'INPUT[name={$key}]'"; }
            $output .= '$$('.implode(',', $corrects).').removeClass("unvalidated");';
        }
        $output = "<script type='text/javascript'>{$output} </script>";
        return($output);
    }


    /**
     *
     * Sanatizes an array of items according to the $this->sanatations
     * sanatations will be standard of type string, but can also be specified.
     * For ease of use, this syntax is accepted:
     * $sanatations = array('fieldname', 'otherfieldname'=>'float');
     */
    public function sanatize($items)
    {
        foreach($items as $key=>$val)
        {
            if(array_search($key, $this->sanatations) === false && !array_key_exists($key,
$this->sanatations)) continue;
            $items[$key] = self::sanatizeItem($val, $this->validations[$key]);
        }
        return($items);
    }


    /**
     *
     * Adds an error to the errors array.
     */
    private function addError($field, $type='string')
    {
        $this->errors[$field] = $type;
    }

    /**
     *
     * Sanatize a single var according to $type.
     * Allows for static calling to allow simple sanatization
     */
    public static function sanatizeItem($var, $type)
    {
        $flags = NULL;
        switch($type)
        {
            case 'url':
                $filter = FILTER_SANITIZE_URL;
            break;
            case 'int':
                $filter = FILTER_SANITIZE_NUMBER_INT;
            break;
            case 'float':
                $filter = FILTER_SANITIZE_NUMBER_FLOAT;
                $flags = FILTER_FLAG_ALLOW_FRACTION | FILTER_FLAG_ALLOW_THOUSAND;
            break;
            case 'email':
                $var = substr($var, 0, 254);
                $filter = FILTER_SANITIZE_EMAIL;
            break;
            case 'string':
            default:
                $filter = FILTER_SANITIZE_STRING;
                $flags = FILTER_FLAG_NO_ENCODE_QUOTES;
            break;

        }
        $output = filter_var($var, $filter, $flags);
        return($output);
    }

    /**
     *
     * Validates a single var according to $type.
     * Allows for static calling to allow simple validation.
     *
     */
    public static function validateItem($var, $type)
    {
        if(array_key_exists($type, self::$regexes))
        {
            $returnval =  filter_var($var, FILTER_VALIDATE_REGEXP, array("options"=>
array("regexp"=>'!'.self::$regexes[$type].'!i'))) !== false;
            return($returnval);
        }
        $filter = false;
        switch($type)
        {
            case 'email':
                $var = substr($var, 0, 254);
                $filter = FILTER_VALIDATE_EMAIL;
            break;
            case 'int':
                $filter = FILTER_VALIDATE_INT;
            break;
            case 'boolean':
                $filter = FILTER_VALIDATE_BOOLEAN;
            break;
            case 'ip':
                $filter = FILTER_VALIDATE_IP;
            break;
            case 'url':
                $filter = FILTER_VALIDATE_URL;
            break;
        }
        return ($filter === false) ? false : filter_var($var, $filter) !== false ? true :
false;
    }
```

```
}
```

Of course, keep in mind that you need to do your sql query escaping too depending on what type of db your are using (mysql_real_escape_string() is useless for an sql server for instance). You probably want to handle this automatically at your appropriate application layer like an ORM. Also, as mentioned above: for outputting to html use the other php dedicated functions like htmlspecialchars ;)

For really allowing HTML input with like stripped classes and/or tags depend on one of the dedicated xss validation packages. DO NOT WRITE YOUR OWN REGEXES TO PARSE HTML!

edited Jul 1 '11 at 23:11
Wesley Murch
55.6k 17 95 146

answered Sep 24 '08 at 23:12
SchizoDuckie
6,902 3 19 34

11 This looks like it might be a handy script for validating inputs, but it is *completely* irrelevant to the question. – rjmunro Aug 1 '11 at 14:50

1 s/sanatize/sanitize/g; – Brock Hensley Mar 18 '13 at 13:24

2 s/sanitize/sanitise/g; – Christian Feb 19 '14 at 1:53

tea and crumpets ? sanitise : sanitize – ch1pn3ss Jan 28 at 19:46

No, there is not.

First of all, SQL injection is an input filtering problem, and XSS is an output escaping one - so you wouldn't even execute these two operations at the same time in the code lifecycle.

Basic rules of thumb

- For SQL query, bind parameters (as with PDO) or use a driver-native escaping function for query variables (such as `mysql_real_escape_string()`)
- Use `strip_tags()` to filter out unwanted HTML
- Escape all other output with `htmlspecialchars()` and be mindful of the 2nd and 3rd parameters here.

answered Sep 24 '08 at 20:30
Peter Bailey
69.9k 20 122 163

1 So you only use strip_tags() or htmlspecialchars() when you know that the input has HTML that you want to get rid of or escape respectively - you are not using it for any security purpose right? Also, when you do the bind, what does it do for stuff like Bobby Tables? "Robert'); DROP TABLE Students;--" Does it just escape the quotes? – Robert Mark Bram Oct 29 '12 at 1:16

1 If you have user data that will go into a database and later be displayed on web pages, isn't it usually read a lot more than it's written? To me, it makes more sense to filter it once (as input) before you store it, instead of having to filter it every time you display it. Am I missing something or did a bunch of people vote for needless performance overhead in this and the accepted answer? – jbo5112 Apr 30 '14 at 14:07

5 @jbo5112 You *can* do that, but then your database is storing HTML, not raw content. For some systems that makes sense, but for many others it doesn't. Think of the impacts that storing HTML has on the rest of your setup. Will you run into length limits or length calculation issues? How about field searching with LIKE clauses or FULLTEXT? Will you ever need this content as *not* HTML? Also, the "performance hit" of converting to HTML at display time is extremely minor, especially when considered against the drawbacks. Plus, much of the impact can be mitigated with sensible caching strategies. – Peter Bailey Apr 30 '14 at 15:45

Best answer for me. It's short and addresses the question well if you ask me. Is it possible to attack PHP somehow via $_POST or $_GET with some injection or is this impossible? – tastro Jul 14 '14 at 12:26

To address the XSS issue, take a look at HTML Purifier. It is fairly configurable and has a decent track record.

As for the SQL injection attacks, make sure you check the user input, and then run it though mysql_real_escape_string(). The function won't defeat all injection attacks, though, so it is important that you check the data before dumping it into your query string.

A better solution is to use prepared statements. The PDO library and mysqli extension support these.

answered Sep 24 '08 at 20:29
jasonbar
7,823 22 37

there is no "best way" to do something like sanitizing input.. Use some library, html purifier is good. These libraries have been pounded on many times. So it is much more bulletproof than anything ou can come up yourself – paan Sep 24 '08 at 22:29

See also bioinformatics.org/phplabware/internal_utilities/htmLawed . From my understanding WordPress uses an older version, core.trac.wordpress.org/browser/tags/2.9.2/wp-includes/kses.php – Steve Clay Jun 6 '10 at 18:09

One trick that can help in the specific circumstance where you have a page like `/mypage?id=53` and you use the id in a WHERE clause is to ensure that id definitely is an integer, like so:

```
if (isset($_GET['id'])) {
  $id = $_GET['id'];
  settype($id, 'integer');
  $result = mysql_query("SELECT * FROM mytable WHERE id = '$id'");
  # now use the result
}
```

But of course that only cuts out one specific attack, so read all the other answers. (And yes I know that the code above isn't great, but it shows the specific defence.)

answered Mar 8 '10 at 23:14
Hamish Downer
5,109 7 41 58

4 I use $id = intval($id) instead :) – silentbang Jul 22 '13 at 6:58

Casting integer is a good way to ensure only numerical data is inserted. – Dan J. Dec 22 '14 at 3:03

What you are describing here is two separate issues:

1. Sanitizing / filtering of user input data.
2. Escaping output.

1) User input should always be assumed to be bad.

Using prepared statements, or/and filtering with mysql_real_escape_string is definitely a must. PHP also has filter_input built in which is a good place to start.

2) This is a large topic, and it depends on the context of the data being output. For HTML there are solutions such as htmlpurifier out there. as a rule of thumb, always escape anything you output.

Both issues are far too big to go into in a single post, but there are lots of posts which go into more detail:

Methods PHP output

Safer PHP output

edited Feb 18 '13 at 7:40
Tony Stark
3,733 6 27 58

answered Jul 16 '12 at 10:44
Andrew
8,884 1 14 33

PHP 5.2 introduced the **filter_var** function.

It supports a great deal of SANITIZE, VALIDATE filters.

http://php.net/manual/en/function.filter-var.php

answered Oct 15 '12 at 8:40
dangel
574 5 7

There is the filter extension (howto-link, manual), which works pretty well with all GPC variables. It's not a magic-do-it-all thing though, you will still have to use it.

answered Sep 24 '08 at 20:26
Till
15.8k 3 40 78

Just wanted to add that on the subject of output escaping, if you use php DOMDocument to make your html output it will automatically escape in the right context. An attribute (value="") and the inner text of a <span> are not equal. To be safe against XSS read this: OWASP XSS Prevention Cheat Sheet answered Nov 17 '14 at 21:59

user138720
26 6

protected by Shankar Damodaran Jan 15 '14 at 4:28

Thank you for your interest in this question. Because it has attracted low-quality answers, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?