How can I sanitize user input with PHP?

Asked 16 years, 3 months ago Modified 4 months ago

Viewed 698k times Part of PHP Collective

1284

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 php security xss sql-injection user-input

Share

Improve this question

Follow

edited Jun 26, 2019 at 18:34

TylerH

21.2k • 76 • 79 • 110

asked Sep 24, 2008 at 20:20

Brent

23.7k • 10 • 47 • 49

16 Answers

Sorted by: Highest score (default)

1316

Answer recommended by PHP Collective

It's a common misconception that user input can be filtered. PHP even had a (now defunct) "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 piece of data within a foreign code, you must format it according to the rules of that code. But you must understand that such rules could be too complicated to try to follow them all manually. For example, in SQL, rules for strings, numbers and identifiers are all different. For your convenience, in most cases there is a dedicated tool for such embedding. For example, when some data has to be used in the SQL query, instead of adding a variable directly to SQL string, it has to be done though a parameter in the query, using prepared statement. And it will take care of all the proper formatting.

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`.

Also, a very compelling example is JSON. The rules are so numerous and complicated that you would never be able to follow them all manually. That's why you should never ever create a JSON string manually, but always

use a dedicated function, `json_encode()` that will correctly format every bit of data.

And so on and so forth ...

The *only* case where you need to actively filter data, is if you're accepting preformatted input. For example, 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.

Share Improve this answer

Follow

edited Apr 30, 2023 at 17:51

Your Common Sense

158k ● 42 ● 221 ● 362

answered Sep 24, 2008 at 22:30

troelskn

117k ● 27 ● 134 ● 156

276 "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, 2008 at 13:32

12 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. – Kornel Sep 9, 2009 at 21:33

7 @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, 2011 at 20:02

12 At this moment `mysql_real_escape_string` is deprecated. It's considered good practice nowadays to use prepared statements to prevent SQL injection. So switch to either MySQLi or PDO. – Marcel Korpel Jun 5, 2013 at 12:46

8 Because you limit the attack surface. If you sanitize early (when input), you have to be certain that there are no other holes in the application where bad data could enter through. Whereas if you do it late, then your output function doesn't have to "trust" that it is given safe data - it simply assumes that everything is unsafe. – troelskn Jul 15, 2014 at 17:33

256

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 Prepared Statements (i.e. using parameters in a template query) 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.

Share Improve this answer

Follow

edited Jul 1, 2015 at 17:59

scorpiodawg

5,752 ● 3 ● 44 ● 62

answered Oct 9, 2008 at 6:28

Andy Lester
93.5k • 15 • 104 • 159

23 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, 2014 at 2:49

5 For the specific case of SQL Injection, *this* is the correct answer! – Scott Arciszewski May 30, 2015 at 2:04

7 Note that prepared statements don't add any security, parameterised queries do. They just happen to be very easy to use together in PHP. – Basic Aug 16, 2015 at 3:01

Its not the only guaranteed way. Hex the input and unhex in query will prevent also. Also hex attacks are not possible if you use hexing right. – Ramon Bakker Feb 22, 2016 at 15:50

1 What if you're inputting something specialized, like email addresses or usernames? – Abraham Brookes Jan 9, 2017 at 8:34

84

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.

Share Improve this answer Follow

edited Dec 23, 2012 at 14:26

answered Sep 24, 2008 at 20:24

Daniel Papasian

16.4k ● 6 ● 31 ● 32

65

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 fields by regexes and can sanitize them. Uses PHP filter_var built-in functions and extra regexes * @package pork */

```
/**
 *  Pork.FormValidator
 *  Validates arrays or properties by setting up simpl
 *  Note that some of the regexes are for dutch input!
 *  Example:
 *
 *  $validations = array('name' => 'anything','email'
'anything','pwd'=>'anything','gsm' => 'phone','birthda
 *  $required = array('name', 'email', 'alias', 'pwd')
 *  $sanitize = array('alias');
 *
 *  $validator = new FormValidator($validations, $requ
```

```
 *
 *  if($validator->validate($_POST))
 *  {
 *      $_POST = $validator->sanitize($_POST);
 *      // now do your saving, $_POST has been sanitiz
 *      die($validator->getScript()."<script
type='text/javascript'>alert('saved changes');</script
 *  }
 *  else
 *  {
 *      die($validator->getScript());
 *  }
 *
 * To validate just one element:
 * $validated = new FormValidator()->validate('blah@bl
 *
 * To sanitize just one element:
 * $sanitized = new FormValidator()->sanitize('<b>blah
 *
 * @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
            '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-z
            'price' => "^[0-9.,]*(([.,][-])|([.,][0-9]
            '2digitopt' => "^\d+(\,\d{2})?\$",
            '2digitforce' => "^\d+\,\d\d\$",
            'anything' => "^[\d\D]{1,}\$"
    );
    private $validations, $sanatations, $mandatories,
$fields;
```

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

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

        return(!$havefailures);
    }

    /**
     *
```

```
     *  Adds unvalidated class to thos elements that a
them from classes that are.
     */
    public function getScript() {
        if(!empty($this->errors))
        {
            $errors = array();
            foreach($this->errors as $key=>$val) { $er
{$key}]'"; }

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


    /**
     *
     * Sanitizes an array of items according to the $t
     * sanitations will be standard of type string, bu
     * For ease of use, this syntax is accepted:
     * $sanitations = array('fieldname', 'otherfieldna
     */
    public function sanitize($items)
    {
        foreach($items as $key=>$val)
        {
            if(array_search($key, $this->sanitations)
!array_key_exists($key, $this->sanitations)) continue;
            $items[$key] = self::sanitizeItem($val, $t
        }
        return($items);
    }
```

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

    /**
     *
     * Sanitize a single var according to $type.
     * Allows for static calling to allow simple sanit
     */
    public static function sanitizeItem($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 $type.
     * Allows for static calling to allow simple valid
     *
     */
    public static function validateItem($var, $type)
    {
        if(array_key_exists($type, self::$regexes))
        {
            $returnval =  filter_var($var, FILTER_VALI
array("options"=> array("regexp"=>'!'.self::$regexes[$
            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_va
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!

Share Improve this answer

Follow

edited Jun 4, 2021 at 8:49

Liam

29.4k ● 28 ● 137 ● 199

answered Sep 24, 2008 at 23:12

SchizoDuckie

9,391 ● 6 ● 35 ● 40

26 This looks like it might be a handy script for validating inputs, but it is *completely* irrelevant to the question. – rjmunro Aug 1, 2011 at 14:50

1 I don't agree with using ORM , it's over engineering imo. – mercury Oct 12, 2021 at 1:40

@PHP >= 8.0 gives error `Parse error: syntax error, unexpected '->' (T_OBJECT_OPERATOR)` – Reham Fahmy Dec 6, 2021 at 10:33

2 @Reham Fahmy: This code is from 2008. It's 2022 now. Don't Use this. Use a framework. – SchizoDuckie Jul 6, 2022

at 10:10

53

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
- Use `strip_tags()` to filter out unwanted HTML
- Escape all other output with `htmlspecialchars()` and be mindful of the 2nd and 3rd parameters here.

Share Improve this answer

Follow

edited Mar 21, 2022 at 7:58

Your Common Sense

158k ● 42 ● 221 ● 362

answered Sep 24, 2008 at 20:30

Peter Bailey

106k ● 32 ● 185 ● 206

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, 2012 at 1:16

2 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, 2014 at 14:07

3 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? – Jo Smo Jul 14, 2014 at 12:26

oh yes, the $post and $get arrays accept all characters, but some of those characters can be used against you if the character is allowed to be enumerated in the posted php page. so if you don't escape encapsulating characters (like ", ' and ` ) it could open up an attack vector. the ` character is often missed, and can be used to form command line execution hacks. Sanitation will prevent user input hacking, but will not help you with web application firewall hacks. – drtechno Sep 18, 2019 at 16:43

27

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, the solution is to use prepared statements. The PDO library and mysqli extension support these.

Share Improve this answer

Follow

edited Mar 21, 2022 at 7:59

Your Common Sense

158k ● 42 ● 221 ● 362

answered Sep 24, 2008 at 20:29

jasonbar

13.5k ● 4 ● 40 ● 47

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, 2008 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, 2010 at 18:09

The problem with wordpress is that its not necessarily a php-sql injection attack that causes database breaches. Miss programmed plugins that store data that an xml query reveals secrets is more problematic. – drtechno Oct 1, 2019 at 17:44

22

There's no catchall function, because there are multiple concerns to be addressed.

1. SQL Injection

Today, generally, every PHP project should be using prepared statements via PHP Data Objects (PDO) as a best practice, *preventing an error from a stray quote as well as a full-featured solution against injection*. It's also the most flexible & secure way to access your database. Check out *(The only proper) PDO tutorial* for pretty much everything you need to know about PDO. (Sincere thanks

to top SO contributor, @YourCommonSense, for this great resource on the subject.)

2. XSS - Validate HTML input

HTML Purifier has been around a long time and is still actively updated. You can use it to only allow harmless HTML, so the resulting code can be used in your HTML pages. Works great with many WYSIWYG editors, but it might be heavy for some use cases.

3. XSS - Sanitize data on the way out

There is no catchall function as well. For different context you need different escaping

- When you output any data in HTML context, use `htmlspecialchars` unless the data was properly sanitized safe and is allowed to display HTML.
- `json_encode` is a safe way to provide values from PHP to Javascript

4. Calling shell commands

Do you call external shell commands using `exec()` or `system()` functions, or to the `backtick` operator?** If so, in addition to SQL Injection & XSS you might have an additional concern to address, *users running malicious commands on your server*. You need to use `escapeshellcmd` if you'd like to escape the entire command OR `escapeshellarg` to escape individual arguments.

Share Improve this answer

Follow

edited Jun 18 at 16:09

Your Common Sense

158k ● 42 ● 221 ● 362

answered Dec 17, 2017 at 18:47

webaholik

1,787 ● 1 ● 22 ● 34

could mb_encode_numericentity be used instead? Since it encodes everything? – drtechno Oct 1, 2019 at 17:33

@drtechno - `mb_encode_numericentity` is discussed in the `htmlspecialchars` link on #3 XSS – webaholik Oct 2, 2019 at 21:11

From what I know, XSS is an output concern, not an input one. – Abel LIFAEFI MBULA Oct 12, 2020 at 7:46

@bam - you are correct, just don't miss a spot! Luckily most frameworks will handle for us when used properly. – webaholik Oct 13, 2020 at 21:30

As FILTER_SANITIZE_STRING is deprecated I took the liberty of modifying the answer – theking2 Jun 18 at 11:59

21

PHP 5.2 introduced the `filter_var` function.

It supports a great deal of `SANITIZE`, `VALIDATE` filters.

Share Improve this answer

Follow

edited Apr 10, 2021 at 10:51

Mykola Semenov

802 ● 3 ● 13 ● 21

answered Oct 15, 2012 at 8:40

dangel
1,544 • 15 • 10

18

Methods for safe database interaction in PHP

Using modern versions of MySQL and PHP

1. Set charset explicitly:

[MySQLi]

```
$mysqli->set_charset("utf8mb");
```

[PDO]

```
$pdo = new PDO('mysql:host=localhost;dbname=testdb;cha
$password);
```

2. Use prepared statements

[MySQLi prepared statements:]

```
$stmt = $mysqli->prepare('SELECT * FROM test WHERE nam
"' OR 1=1 /*";<br>$stmt->bind_param('s', $param);
$stmt->execute();
```

PDO Prepared Statements:

Compared to MySQLi prepared statements, PDO supports more database drivers and named parameters:

```
$stmt = $pdo->prepare('SELECT * FROM test WHERE name =
$stmt->execute(["' OR 1=1 /*"]);
```

Share Improve this answer

Follow

edited Mar 6, 2023 at 10:45

Your Common Sense

158k ● 42 ● 221 ● 362

answered Feb 25, 2018 at 6:11

Mark Martin

372 ● 3 ● 8

14

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 i
  # 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.)

Share Improve this answer

Follow

answered Mar 8, 2010 at 23:14

Hamish Downer

17k ● 17 ● 92 ● 85

12 I use $id = intval($id) instead :) – Duc Tran Jul 22, 2013 at 6:58

Casting integer is a good way to ensure only numerical data is inserted. – test Dec 22, 2014 at 3:03

1 `$id = (int)$_GET['id']` and `$que = sprintf('SELECT ... WHERE id="%d"', $id)` is good too – vladkras Nov 21, 2016 at 14:40

perhaps `if (isset($_GET['id']) { if !( (int) $_GET['id'] === intval($_GET['id'] ) ) { throw new \InvalidArgumentException('Invalid page id format'); } /* use a prepared statement for insert here */ };` might suit you. I prefer to make no database call at all if I can identify that a parameter is definitely not valid based on known schema it is being handed to. – mopsyd Jan 4, 2021 at 2:31

9

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]

Share Improve this answer

Follow

edited Feb 18, 2013 at 7:40

Tony Stark

8,084 ● 8 ● 46 ● 63

answered Jul 16, 2012 at 10:44

Andrew

12.8k ● 1 ● 35 ● 48

8

If you're using PostgreSQL, the input from PHP can be escaped with `pg_escape_literal()`

```
$username = pg_escape_literal($_POST['username']);
```

From the documentation:

> `pg_escape_literal()` escapes a literal for querying the PostgreSQL database. It returns an escaped literal in the PostgreSQL format.

Share Improve this answer

Follow

edited Apr 10, 2021 at 11:05

Your Common Sense

158k ● 42 ● 221 ● 362

answered May 27, 2015 at 15:36

Alejandro Silva

9,098 ● 1 ● 37 ● 29

2 **pg_escape_literal()** is the recommended function to use for PostgreSQL. – kittycat May 30, 2015 at 8:24

7

You never sanitize input.

You always sanitize output.

The transforms you apply to data to make it safe for inclusion in an SQL statement are completely different from those you apply for inclusion in HTML are completely different from those you apply for inclusion in Javascript are completely different from those you apply for inclusion in LDIF are completely different from those you apply to inclusion in CSS are completely different from those you apply to inclusion in an Email....

By all means validate input - decide whether you should accept it for further processing or tell the user it is unacceptable. But don't apply any change to representation of the data until it is about to leave PHP land.

A long time ago someone tried to invent a one-size fits all mechanism for escaping data and we ended up with "magic_quotes" which didn't properly escape data for all output targets and resulted in different installation requiring different code to work.

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Feb 19, 2018 at 12:12

symcbean

48.3k ● 6 ● 62 ● 97

one problem with that is that its not always a database attack, and all user input should be protected from the system. not just one language type. So on your sites, when you enumerate your $_POST data, even with using binding, it could escape out enough to execute shell or even other php code. – drtechno Oct 1, 2019 at 17:29

"its not always a database attack" : "The transforms you apply to data to make it safe for inclusion in an SQL statement are completely different from those...." – symcbean Oct 2, 2019 at 10:55

"all user input should be protected from the system" : no the system should be protected from user input. – symcbean Oct 2, 2019 at 10:55

well I ran out of words, but yes the input needs to be prevented from effecting the system operation. to clarify this... – drtechno Oct 4, 2019 at 13:09

1 Both input and output should be sanitized. – JSowa May 6, 2020 at 12:56

4

Easiest way to avoid mistakes in sanitizing input and escaping data is using PHP framework like Symfony, Nette etc. or part of that framework (templating engine, database layer, ORM).

Templating engine like Twig or Latte has output escaping on by default - you don't have to solve manually if you have properly escaped your output depending on context (HTML or Javascript part of web page).

Framework is automatically sanitizing input and you should't use $_POST, $_GET or $_SESSION variables directly, but through mechanism like routing, session handling etc.

And for database (model) layer there are ORM frameworks like Doctrine or wrappers around PDO like Nette Database.

You can read more about it here - What is a software framework?

Share Improve this answer

Follow

edited Feb 23, 2018 at 17:29

Jon Winstanley

23.3k ● 22 ● 78 ● 119

answered Jul 17, 2015 at 17:13

Ondřej Šotek

1,812 ● 1 ● 17 ● 25

3

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 are not equal. To be safe against XSS read this: OWASP XSS Prevention Cheat Sheet

Share Improve this answer

Follow

answered Nov 17, 2014 at 21:59

user138720

151 • 8

> Speaking of OWASP cheat sheets, my personal opinion is that the best way to educate PHP developers in secure coding is to hire the services of an ethical hacker or to learn how to use and run Kali Linux / Metasploit, adn then demo various exploits. When developers see how an exploit actually works, the impact is much higher than a dozen SO Q&A's. Then interested developers begin to study file formats, protocols, query logging, demand Wireshark, etc. Till then, they follow Q&a's on hope and faith. Again, this is a personal opinion. YMMV. – site80443 Aug 11, 2023 at 12:11

-1

Several answers recommend **HTMLPurifier**, which I found effective but challenging to scale due to its extensive configuration requirements. No other PHP library provided a viable alternative, so I developed xssless. This library currently wraps the JS library DOMPurify, which is also recommended by OWASP for HTML sanitization.

> ... OWASP recommends **DOMPurify** for HTML Sanitization.

Share Improve this answer

Follow

answered Aug 10 at 18:40

medilies

2,178 1 14 38

Good suggestion but a complete off topic here. HTML sanitization is just a specific part of input validation. And therefore doesn't answer the actual question, "How to sanitize input" in PHP. Not to mention there is zero sense in cleaning input with deliberate XSS. In case you go at lengths to detect it, then it must be rejected right away
– Your Common Sense Aug 11 at 6:47

@YourCommonSense I agree, I am not giving a full solution to the question. I am offering an alternative to HTMLPurifier since it was recommended three times in previous answers. And for cleaning content from XSS instead of rejecting it, you will need that when allowing your users to use rich text editors, You don't want to make your users struggle to save their content. – medilies Aug 11 at 8:22