When is it best to sanitize user input?

Asked 16 years, 3 months ago Modified 7 months ago

Viewed 34k times

73

User equals untrustworthy. Never trust untrustworthy user's input. I get that. However, I am wondering when the best time to sanitize input is. For example, do you blindly store user input and then sanitize it whenever it is accessed/used, or do you sanitize the input immediately and then store this "cleaned" version? Maybe there are also some other approaches I haven't though of in addition to these. I am leaning more towards the first method, because any data that came from user input must still be approached cautiously, where the "cleaned" data might still unknowingly or accidentally be dangerous. Either way, what method do people think is best, and for what reasons?

xss sql-injection user-input sanitization

Share

Improve this question

Follow

edited Mar 25, 2019 at 13:39

Script47

14.5k ● 4 ● 47 ● 68

asked Aug 29, 2008 at 18:07

Aaron
23.8k 10 50 48

14 Answers

Sorted by: Highest score (default)

61

Unfortunately, almost no one of the participants ever clearly understands what are they talking about. Literally. Only Kibbee managed to make it straight.

This topic is all about sanitization. But the truth is, such a thing like wide-termed "general purpose sanitization" everyone is so eager to talk about is **just doesn't exist.**

There are **a zillion different mediums**, each require **it's own, distinct data formatting.** Moreover - even **single certain medium require different formatting for it's parts**. Say, HTML formatting is useless for javascript embedded in HTML page. Or, string formatting is useless for the numbers in SQL query.

As a matter of fact, such a "sanitization as early as possible", as suggested in most upvoted answers, is just **impossible**. As one just cannot tell in which certain medium or medium part the data will be used. Say, we are preparing to defend from "sql-injection", escaping everything that moves. But whoops! - some required fields weren't filled and we have to fill out data back into form instead of database... with all the slashes added.

On the other hand, we diligently escaped all the "user input"... but in the sql query we have no quotes around it,

as it is a number or identifier. And no "sanitization" ever helped us.

On the third hand - okay, we did our best in sanitizing the terrible, untrustworthy and disdained "user input"... but in some inner process we used this very data without any formatting (as we did our best already!) - and whoops! have got second order injection in all its glory.

So, from the real life usage point of view, the only proper way would be

- formatting, not whatever "sanitization"
- right before use
- according to the certain medium rules
- and even following sub-rules required for this medium's different parts.

edited Sep 16, 2021 at 9:11

answered Sep 2, 2013 at 12:49

Your Common Sense
158k ● 42 ● 221 ● 362

8 Reading through the responses, i was feeling quite an urge to post something just like this. – cHao Sep 2, 2013 at 12:58

1 It seems that what we really need instead of a "general-purpose sanitizer", is a well-designed, flexible framework for addressing your bullet points while still keeping code sane and maintainable. – alexw Feb 15, 2015 at 17:55

3 This is a much better answer than the accepted one... The facepalm avatar is perfect here :-) – RavenHursT Jul 14, 2016 at 4:33

16

It depends on what kind of sanitizing you are doing.

For protecting against SQL injection, don't do anything to the data itself. Just use prepared statements, and that way, you don't have to worry about messing with the data that the user entered, and having it negatively affect your logic. You have to sanitize a little bit, to ensure that numbers are numbers, and dates are dates, since everything is a string as it comes from the request, but don't try to do any checking to do things like block keywords or anything.

For protecting against XSS attacks, it would probably be easier to fix the data before it's stored. However, as others mentioned, sometimes it's nice to have a pristine copy of exactly what the user entered, because once you change it, it's lost forever. It's almost too bad there's not a fool proof way to ensure you application only puts out sanitized HTML the way you can ensure you don't get caught by SQL injection by using prepared queries.

Share Improve this answer answered Aug 30, 2008 at 16:22

I sanitize my user data much like Radu...

1. First client-side using both regex's and taking control over allowable characters input into given form fields using javascript or jQuery tied to events, such as onChange or OnBlur, which removes any disallowed input before it can even be submitted. Realize however, that this really only has the effect of letting those users in the know, that the data is going to be checked server-side as well. It's more a warning than any actual protection.
2. Second, and I rarely see this done these days anymore, that the first check being done server-side is to check the location of where the form is being submitted from. By only allowing form submission from a page that you have designated as a valid location, you can kill the script BEFORE you have even read in any data. Granted, that in itself is insufficient, as a good hacker with their own server can 'spoof' both the domain and the IP address to make it appear to your script that it is coming from a valid form location.
3. Next, and I shouldn't even have to say this, but always, and I mean **ALWAYS**, run your scripts in taint mode. This forces you to not get lazy, and to be diligent about step number 4.

4. Sanitize the user data as soon as possible using well-formed regexes appropriate to the data that is expected from any given field on the form. Don't take shortcuts like the infamous '*magic horn of the unicorn*' to blow through your taint checks... or you may as well just turn off taint checking in the first place for all the good it will do for your security. That's like giving a psychopath a sharp knife, bearing your throat, and saying 'You really won't hurt me with that will you".

   And here is where I differ than most others in this fourth step, as I only sanitize the user data that I am going to actually USE in a way that may present a security risk, such as any system calls, assignments to other variables, or any writing to store data. If I am only using the data input by a user to make a comparison to data I have stored on the system myself (therefore knowing that data of my own is safe), then I don't bother to sanitize the user data, as I am never going to us it a way that presents itself as a security problem. For instance, take a username input as an example. I use the username input by the user only to check it against a match in my database, and if true, after that I use the data from the database to perform all other functions I might call for it in the script, knowing it is safe, and never use the users data again after that.

5. Last, is to filter out all the attempted auto-submits by robots these days, with a 'human authentication' system, such as Captcha. This is important enough

these days that I took the time to write my own 'human authentication' schema that uses photos and an input for the 'human' to enter what they see in the picture. I did this because I've found that Captcha type systems really annoy users (you can tell by their squinted-up eyes from trying to decipher the distorted letters... usually over and over again). This is especially important for scripts that use either SendMail or SMTP for email, as these are favorites for your hungry spam-bots.

To wrap it up in a nutshell, I'll explain it as I do to my wife... your server is like a popular nightclub, and the more bouncers you have, the less trouble you are likely to have in the nightclub. I have two bouncers outside the door (client-side validation and human authentication), one bouncer right inside the door (checking for valid form submission location... 'Is that really you on this ID'), and several more bouncers in close proximity to the door (running taint mode and using good regexes to check the user data).

I know this is an older post, but I felt it important enough for anyone that may read it after my visit here to realize their is no '*magic bullet*' when it comes to security, and it takes all these working in conjuction with one another to make your user-provided data secure. Just using one or two of these methods alone is practically worthless, as their power only exists when they all team together.

Or in summary, as my Mum would often say... 'Better safe than sorry".

UPDATE:

One more thing I am doing these days, is Base64 encoding all my data, and then encrypting the Base64 data that will reside on my SQL Databases. It takes about a third more total bytes to store it this way, but the security benefits outweigh the extra size of the data in my opinion.

Share Improve this answer

Follow

edited Mar 25, 2019 at 14:07

Script47

14.5k ● 4 ● 47 ● 68

answered Aug 20, 2010 at 2:18

Epiphany

1,904 ● 1 ● 20 ● 14

thanks :) for sharing. I liked the 2 point. i.e. checking the source before form submission. – Gaurav Sharma Jun 22, 2011 at 9:48

8 Locking out blind users is not best practice. – TRiG Jun 28, 2011 at 13:43

1 Update just made it fabulous. – Your Common Sense Sep 2, 2013 at 12:52

So I'm guessing that "taint mode" is a joke, but could you elaborate on what that is, just so I understand? – Funktr0n Apr 17, 2014 at 8:33

1 Through I agree with your edit @CommonSenseCode, I rolled it back as per:

meta.stackoverflow.com/questions/381780/… – Script47 Mar 25, 2019 at 14:09

12

I like to sanitize it as early as possible, which means the sanitizing happens when the user tries to enter in invalid data. If there's a TextBox for their age, and they type in anything other that a number, I don't let the keypress for the letter go through.

Then, whatever is reading the data (often a server) I do a sanity check when I read in the data, just to make sure that nothing slips in due to a more determined user (such as hand-editing files, or even modifying packets!)

Edit: Overall, sanitize early and sanitize any time you've lost sight of the data for even a second (e.g. File Save -> File Open)

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:09

Daniel Jennings

6,490 ● 3 ● 33 ● 44

8 ... or even a user with disabled js :/ – Fluffy Jul 19, 2010 at 8:16

25 This is bad advice. You should sanitize your outputs, not your inputs. – csauve Mar 30, 2015 at 20:45

8 @csauve is correct. Not sure why this is the accepted answer when it's quite clear that the correct solution here is to sanitize outputs. Not try to "detect" "bad" or "malicious" inputs. Approaching the problem this way would end up being an attempt to implement an almost infinite number of

heuristic solutions. See here for more info: owasp.org/index.php/… – RavenHursT Jul 14, 2016 at 4:29

5 Upon reading the question and answer again, Daniel may be interpretting "Sanitize" to mean "Validate". If you want to validate that an input matches a criteria for a field (ie must be a positive integer), then do it on input. If you want to sanitize as in protect against malicious values then do it on output. – csauve Jul 14, 2016 at 18:58

6

The most important thing is to always be consistent in when you escape. Accidental double sanitizing is lame and not sanitizing is dangerous.

For SQL, just make sure your database access library supports bind variables which automatically escapes values. Anyone who manually concatenates user input onto SQL strings should know better.

For HTML, I prefer to escape at the last possible moment. If you destroy user input, you can never get it back, and if they make a mistake they can edit and fix later. If you destroy their original input, it's gone forever.

Share Improve this answer

Follow

answered Aug 29, 2008 at 19:42

cpm

1,543 ● 10 ● 16

3

Early is good, definitely before you try to parse it. Anything you're going to output later, or especially pass to

other components (i.e., shell, SQL, etc) must be sanitized.

But don't go overboard - for instance, passwords are hashed before you store them (right?). Hash functions can accept arbitrary binary data. And you'll never print out a password (right?). So don't parse passwords - and don't sanitize them.

Also, make sure that you're doing the sanitizing from a trusted process - JavaScript/anything client-side is worse than useless security/integrity-wise. (It might provide a better user experience to fail early, though - just do it both places.)

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:40

Peter Stone

3,786 ● 4 ● 24 ● 14

3

My opinion is to sanitize user input as soon as posible client side and server side, i'm doing it like this

1. (client side), allow the user to enter just specific keys in the field.
2. (client side), when user goes to the next field using onblur, test the input he entered against a regexp, and notice the user if something is not good.
3. (server side), test the input again, if field should be INTEGER check for that (in PHP you can use is_numeric()), if field has a well known format check

it against a regexp, all others (like text comments), just escape them. If anything is suspicious stop script execution and return a notice to the user that the data he enetered in invalid.

If something realy looks like a posible attack, the script send a mail and a SMS to me, so I can check and maibe prevent it as soon as posible, I just need to check the log where i'm loggin all user inputs, and the steps the script made before accepting the input or rejecting it.

Share Improve this answer

Follow

answered Jul 19, 2010 at 8:11

Radu Maris

5,748 ● 4 ● 43 ● 54

2

Perl has a taint option which considers all user input "tainted" until it's been checked with a regular expression. Tainted data can be used and passed around, but it taints any data that it comes in contact with until untainted. For instance, if user input is appended to another string, the new string is also tainted. Basically, any expression that contains tainted values will output a tainted result.

Tainted data can be thrown around at will (tainting data as it goes), but as soon as it is used by a command that has effect on the outside world, the perl script fails. So if I use tainted data to create a file, construct a shell command, change working directory, etc, Perl will fail with a security error.

I'm not aware of another language that has something like "taint", but using it has been very eye opening. It's amazing how quickly tainted data gets spread around if you don't untaint it right away. Things that natural and normal for a programmer, like setting a variable based on user data or opening a file, seem dangerous and risky with tainting turned on. So the best strategy for getting things done is to untaint as soon as you get some data from the outside.

And I suspect that's the best way in other languages as well: validate user data right away so that bugs and security holes can't propagate too far. Also, it ought to be easier to audit code for security holes if the potential holes are in one place. And you can never predict which data will be used for what purpose later.

Share Improve this answer

Follow

answered Aug 29, 2008 at 19:23

Jon Ericson

21.5k ● 12 ● 102 ● 151

3 "validate user data right away" = wrong. Your last sentence gets it right: "And you can never predict which data will be used for what purpose later." This is why you need to sanitize your data as you *use* is, not when it is created. – csauve Mar 30, 2015 at 20:49

1 @csauve: Oh, I don't think you shouldn't *also* sanity check data before you use it. But let me ask you: if you collect data from a user and it turns out to be unusable at some later date, how do you prompt the user to correct the problem? To be honest, the question is really a bit of a false dichotomy. – Jon Ericson Mar 30, 2015 at 21:04

1 You should check for *unusable* data according to your *business requirements*, but data that has special characters isn't necessarily unusable. You just need to encode it appropriately for the language it is being inserted into (ie url encode for urls, html encode for html). – csauve Apr 1, 2015 at 15:31

1

Clean the data before you store it. Generally you shouldn't be preforming **ANY** SQL actions without first cleaning up input. You don't want to subject yourself to a SQL injection attack.

I sort of follow these basic rules.

1. Only do modifying SQL actions, such as, INSERT, UPDATE, DELETE through POST. Never GET.
2. Escape everything.
3. If you are expecting user input to be something make sure you check that it is that something. For example, you are requesting an number, then make sure it is a number. Use validations.
4. Use filters. Clean up unwanted characters.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:10

mk.

26.2k ● 13 ● 39 ● 41

Users are evil!

1

Well perhaps not always, but my approach is to always sanatize immediately to ensure nothing risky goes anywhere near my backend.

The added benefit is that you can provide feed back to the user if you sanitize at point of input.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:10

Martin

40.3k ● 20 ● 100 ● 131

1 Martin, is it just me or is this full of innuendo? :) – Aaron Sep 15, 2008 at 22:54

When I wrote it I had no intention of that being the case, re-reading it I have to agree with you :) – Martin Nov 24, 2008 at 11:18

1

I sanitize my data right before I do any processing on it. I may need to take the First and Last name fields and concatenate them into a third field that gets inserted to the database. I'm going to sanitize the input before I even do the concatenation so I don't get any kind of processing or insertion errors. The sooner the better. Even using Javascript on the front end (in a web setup) is ideal because that will occur without any data going to the server to begin with.

The scary part is that you might even want to start sanitizing data coming out of your database as well. The recent surge of ASPRox SQL Injection attacks that have

been going around are doubly lethal because it will infect all database tables in a given database. If your database is hosted somewhere where there are multiple accounts being hosted in the same database, your data becomes corrupted because of somebody else's mistake, but now you've joined the ranks of hosting malware to your visitors due to no initial fault of your own.

Sure this makes for a whole lot of work up front, but if the data is critical, then it is a worthy investment.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:14

Dillie-O

29.7k ● 14 ● 102 ● 141

0

User input should always be treated as malicious before making it down into lower layers of your application. Always handle sanitizing input as soon as possible and should not for any reason be stored in your database before checking for malicious intent.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:09

Sean Chambers

8,670 ● 7 ● 42 ● 55

0

Assume all users are malicious. Sanitize all input as soon as possible. Full stop.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:13

BrianH
1,080 • 1 • 16 • 23

-2

I find that cleaning it immediately has two advantages. One, you can validate against it and provide feedback to the user. Two, you do not have to worry about consuming the data in other places.

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:09

Craig
11.9k • 13 • 45 • 62