

Forum:[Hack Forums](#)

Thread: [\[Polymath\] Securing your website against nasty hackers!](#)

Th3PonyWizard - August 8, 2014, 5:54 am

Securing Your Website Against Hackers

[img]http://i.minus.com/ivrx6GdS8dZ5O.gif[/img]

Contents:

- 1) SQL Injection prevention
- 2) XSS Injection prevention
- 3) LFI Injection prevention

Note: I'm no PHP pro; there are usually better ways of doing stuff here. I'm focussing on showing you the ideas rather than telling you what to copy and paste etc.

[img]http://i.minus.com/ivrx6GdS8dZ5O.gif[/img]

SQL Injection

Is the art of manipulating user inserted SQL queries into being received by the server through the http, all the way to the DMBS to being executed, and all the way back to the client to display the malfunctioning content the attacker wants the website to display. SQL Injection will usually be used for:

1. Giving unauthorized access to hidden website areas (Common)
2. Providing hidden database information such as client details, passwords, and information (Common)
3. Serve as a DoS point in cases where an injection doesn't lead to database access (Less common)
4. Being bent to a client side attack (XSS), specifically stealing cookies and such. Every SQLi is an XSS. (Less common)
5. Grant arbitrary file reading with load_file() due to incorrect current MySQL user privilege placement (Rare: MySQL)
6. Grant complete ADMINISTRATOR/SYSTEM netuser privileges through the combination of a skilled injector and a vulnerable, unpatched Win server. (xp_cmdshell) (Rare: MSSQL 2000/2005)
7. Abuse Stacked Queries based injection to include multiple types of queries to be able to have complete database control: pulling, inserting, deleting, creating information at will. (Rare: ASP/.NET)

You don't want that, do you?

I'm going to tell you how to prevent this form of attack on your website. We're using the PHP/MySQL platforms.

Prepared statements

This is the best way of doing it. It's not the easiest, but when doing creative queries beyond the basic select GET ID value, you'll find that it is the best solution. The basic documentation regarding how to use this can be viewed [here](#).

Basically:

```
mysqli_prepare ($connection variable{$con = mysqli_connect()}, string $query{SELECT priv FROM testUsers WHERE username=? AND password=?})
```

This code is pseudo. The red means stuff that is inside the variable, to make it more understandable. You then fetch the output of mysqli_prepare. Generally, you'd want to learn MySQLi (MySQL Improved) here, or PDO.

Limit Maximum Input Length

```
<?php
$get = $_GET['id'];
$len = 5;
if(strlen($get) > $len)
{
die("Illegal URL set of actions");
}
?>
```

If the input parameter is larger than five (every injection will be), the page will die.

Force Casting Input Variables

```
<?php
$get = (int)$_GET['id'];
?>
```

or

```
<?php
$get = intval($_GET['id']);
?>
```

The list goes on and on. There are a ton of ways to express what I'm saying here and how simple and effective this is. It's arguably my favorite method for SIMPLE stuff. You're going to face some issues when doing an INSERT query with legit English input ;)

Blacklisting

```
<?php
$get = $_GET['id'];
if(preg_match('/and|where|if|union|select|all|order|concat|group|schema/i', $get)) {
die("bad word detected :^");
}
?>
```

Yeah, this way is sort of awful. I'm just explaining it to show you that it exists - blacklisting popular injection words. Please don't use this one!

mysql_real_escape_string()

In my beginning year, I used to jump to this function. Now I force myself to forget it. How to:

<http://php.net/manual/en/function.mysql-real-escape-string.php>

function will later be deleted on (php 5.5).

http://www.iodigitalsec.com/mysql_real_escape_string-wont-magically-solve-your-sql-injection-problems/

Do NOT rely on mysql_real_escape_string(). However, if you do: set_charset('utf8'), and only then use it.

[img]http://i.minus.com/ivrx6GdS8dZ5O.gif[/img]

XSS Injection Prevention

XSS (Cross Site Scripting, C replaced the X because Cascade Style Sheets stole the CSS term. Dirtbags.) is the categorizing term for attacking the other end site of the Website formation = The Client. This means client sided Keyloggers, Cookie Stealers, XSSF shells and future shells (XSS shells), session fixation through URL parameters, ClickJacking, form grabbing, Phishing, JavaScript, BTC miners, browser crashers, browser exploit kits, victim becomes Proxy, XSS worms, DoS and DDoS (to client and server as one), AND MORE. Yeah, there's more.

To avoid this, you need to filter out what you let in. Remember: user input is the devil. You don't need any fancy statements or blacklists here.

htmlspecialchars() vs htmlentities()

Special characters will cover the most lethal of the characters, while entities will cover absolutely all of them. I'd recommend using the latter, but it doesn't block the ' sign by default, so if you use it, use it like this:

```
<?php
$get = $_GET['x'];
echo htmlentities($get, ENT_QUOTES);
?>
```

Popular Question:

When using a database, should you filter input BEFORE database insertion or AFTER insertion when you echo it?

Answer: This is subjective; both have issues. You might suffer some ugly character twists (double filtering, if you use gfc_magic_quotes etc), but it's okay to handle (compared to having to filter the same input OVER AND OVER again as it's unfiltered inside the database).

So yeah, my answer would be filter it before database insertion.

[img]http://i.minus.com/ivrx6GdS8dZ5O.gif[/img]

LFI Prevention

LFI (Local File Inclusion), as well as RFI (Remote File Inclusion)- or even RCE (Remote Code Execution), at times- all have a possibility of being executed from a single mistake: including user input. What are the possible consequences? Server ownage. Nothing more, nothing less; it's this dangerous.

No stress, though. It's very easy to prevent. If you're working with a bunch of PHP pages being displayed through a built-in formation that links around stuff and includes what is linked, all you have to do is to specify exactly the pages that can be included inside an array, and then later check whether the inclusion request actually has anything from that list.

```
<?php
$list = array('one.php', 'help.php', 'etc.php');
if(!in_array($_GET['include'], $list)
{
die("requested page doesn't exist");
}
else {
include($_GET['include']);
}
?>
```

Another way is to use `open_basedir` and basically limit the ability of PHP to include files from other directories, but that isn't the best method (if the attacker includes an admin session file from the same dir, for example). You can review the process over [here](#).

These are basically the basics. Of course, there is much more to learn and comprehend around website security. If there will be a demand, I'll release other tutorials regarding this subject.

- Written by Th3PonyWizard/Sonder

Aura - August 8, 2014, 5:56 am

Awww yisss, finally a Website Security-related tutorial here ?
Good job ponyfuk, when are you opening the Help Desk ?

Th3PonyWizard - August 8, 2014, 5:56 am

(Aug 8, 2014 05:56 AM)Aur? Wrote: [\[CITE\]](#) Awww yisss, finally a Website Security-related tutorial here ?

Good job ponyfuk, when are you opening the Help Desk ?Doing it as we speak. I don't really know the amount of people needing help over that subject here

Aura - August 8, 2014, 5:59 am

(Aug 8, 2014 05:56 AM)Sonder Wrote: [\[CITE\]](#) Doing it as we speak. I don't really know the amount of people needing help over that subject here

Well, it's better to create a Help Desk, even if no one asks questions in it rather than me sending you a message on Skype telling you "You're up!" when someone ask a Web Security related question.

Meow - August 8, 2014, 6:00 am

Very HQ tutorial! Thanks OP,It will help a lot of people! ?

TheNakedEmperor - August 8, 2014, 7:05 am

Very nice!
thanks for sharing op I'm sure a lot of us find it useful

Doxicity - August 8, 2014, 7:05 am

Very nice tutorial Pony, I'm sure it'll help many.
I didn't recognize you with that new name. :)

Skorp - August 8, 2014, 7:11 am

You forgot RFI lol jk, nice guide here my son. :)

Th3PonyWizard - August 8, 2014, 7:36 am

(Aug 8, 2014 07:11 AM)Skorp Wrote: [\[CITE\]](#) You forgot RFI lol jk, nice guide here my son. :)updated
php = no rfi
and I can't even bother adding it

Hidden™ - August 8, 2014, 7:38 am

Thanks for the share, I am opening my clothing lines website soon and this will really help.

Helios. - August 8, 2014, 9:51 am

Great tutorial, however I found 1 tiny mistake.

This code:

PHP Code

```
<?php
$get = $_GET['id'];
if(!is_int($get))
{
die("URL parameter isn't integer");
}
?>
```

Will always say that the parameter is not integer, even if you actually enter a number.
Because \$_GET, \$_POST etc.. Are actually strings.

Check this (read the note): <http://php.net/manual/en/function.is-int.php>

EDIT:

Also to secure your web application from SQL Injections just use prepared statements (MySQLi or PDO).

It's not necessary for checks and "die" the web application.

Anyways, again great tutorial.

The purpose of this reply wasn't to act like a smart-ass or to trash your thread. ?

Th3PonyWizard - August 8, 2014, 10:19 am

(Aug 8, 2014 09:51 AM)Helios. Wrote: [\[CITE\]](#) Great tutorial, however I found 1 tiny mistake.

This code:

PHP Code

```
<?php
$get = $_GET['id'];
if(!is_int($get))
{
die("URL parameter isn't integer");
}
?>
```

Will always say that the parameter is not integer, even if you actually enter a number.
Because \$_GET, \$_POST etc.. Are actually strings.

Check this (read the note): <http://php.net/manual/en/function.is-int.php>

EDIT:

Also to secure your web application from SQL Injections just use prepared statements (MySQLi or PDO).

It's not necessary for checks and "die" the web application.

Anyways, again great tutorial.

The purpose of this reply wasn't to act like a smart-ass or to trash your thread. ?Correct, my bad. It was a recent addition I've added while writing and now I see why I've never actually used it.

Using Prepared statements is the best solution like I've said, but from my experience people would still avoid using them at any price. So showing them the cheap yet somewhat effective methods are important :)