The War on SQLi, or What Happened in 4.8.2 and 4.8.3

The War on SQLi, or What Happened in 4.8.2 and 4.8.3 - Author not stated, Making WordPress Secure.

- Published: 2017-11-13
- Original: <https://make.wordpress.org/security/2017/11/13/the-war-on-sqli-or-what-happened-in-4-8-2-and-4-8-3/>
- Preserved from: https://make.wordpress.org/security/2017/11/13/the-war-on-sqli-or-what-happened-in-4-8-2-and-4-8-3/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

WordPress is complex software used by a large percentage of the internet in millions of unique ways. So when there is an issue that touches deep in the internals of the WordPress codebase, we need to be careful and deliberate. Even more so if the issue has security implications. An incorrect or incomplete fix could break millions of sites or, possibly worse yet, leave them vulnerable.

The Original Issues

Yes, “issues” plural. There were two of them that were somewhat worse when linked together. Both were first discovered by one of our WordPress security team members.

Array Handling in prepare()

The first was that `wpdb::prepare()` accepted an array of replacements as the second parameter, or any number of mixed placeholders. In other words, both of these were valid and functionally identical:

```
$wpdb->prepare( '%s %d %s', array( 'testing', 1, '2 3' ) );
$wpdb->prepare( '%s %d %s', 'testing', 1, '2 3' );
```

However, if the first argument was array, then the following arguments were ignored. So this would give a potentially unexpected and possibly unsafe returned value:

```
$wpdb->prepare( '%s %d %s', array( 'testing', 0, 'NO!' ), 1, '2 3' ) );
```

Poor Defense Against Unsupported Placeholders

The second issue was that `wpdb::prepare()` worked with both the `%c` as well as numbered placeholders, even though it was documented as not supporting those. We realized this could be potentially exploitable. For example, if a user could control both the `page_title` and the `post_type` of a `get_page_by_title()` call:

```
get_page_by_title( '39', OBJECT, array( '%1$c) OR 1 = 1 -- -' ) )

// as sprintf( '%1$c', '39' ) == "'", the following happens to the query:
//
// $sql = $wpdb->prepare( "SELECT ID FROM $wpdb->posts WHERE post_title = %s AND post_type IN ( '%1$c) OR 1= 1 -- -' )", "3
// $sql = "SELECT ID FROM $wpdb->posts WHERE post_title = '39' AND post_type IN ( '' ) OR 1 = 1 -- - ' )"
```

As is often the case, the further we got into researching it, the more we found. Eventually coming up with a proof of concept that followed a commonly used coding pattern and showed that it was exploitable:

```
$user = $wpdb->prepare( "user_login = %s", "%1$' OR [SQLi PAYLOAD] -- -" );
$sql  = $wpdb->prepare( "SELECT * FROM wp_users WHERE ID != %d AND " . $user, 1337 );
//SELECT * FROM wp_users WHERE ID != 1337 AND user_login = '' OR [SQLi PAYLOAD] -- -'
```

The problem was coming up with a fix that would close the vulnerability without breaking too many plugins and sites.

The Outside Report

It was around this time, on October 28th of last year, that we received a report from Slavco via our security E-Mail address. Five days later, on November 3rd, an invitation to HackerOne was sent and Slavco opened a report on the WordPress HackerOne program. Both issues were noted in the report. The array issue:

```
//Issue number 1
//no matter how many input arguments have prepare method, if first one is an array will use it and avoid any of the another
$in1 = array("a","b","c");//only sql escaped variable...
$in2 = 2;//example: cleaned variable
$in3 = 3;//example: user id

echo $demo->prepare("Test %s %s %s", $in1, $in2, $in3)."\n";
```

And the “double prepare” issue as we were now calling it:

```
//Issue number 2
//If you use wpdb::prepare on string that is already prepared
//could be combined with issue 1 or with sprintf argument swapping e.g. %1$%s

$str = $demo->prepare("where a = %s ","'%s'");
echo $demo->prepare("select * from t $str and q = %s or g = %s", array(" and 1=1 #", "a", "b"), 1 )."\n";

$str = 'valid sql \'%1$\'%s\' test@gmail.com #\'';
echo $demo->prepare("select * from t $str and q = %d ", 1 )."\n";
```

The Fix(es)

We had potential fixes for both of these issues at the time the report was opened and they were combined into a single patch and added to the report on November 9th, eleven days after the first E-Mail. However, we were not yet comfortable with the backwards compatibility of the fix. It changed the inner workings of `wpdb::prepare()` in two ways and while it brought it more in line with the documentation for that function, we were uneasy about how many plugins or sites might be affected.

It took a lot of time, but we continued to test the fixes that we had developed, try alternatives, etc. We had some back and forth on the HackerOne report, ran some checks against the plugin A plugin is a piece of software containing a group of functions that can be added to a WordPress website. They can extend functionality or add new features to your WordPress websites. WordPress plugins are written in the PHP programming language and integrate seamlessly with WordPress. These can be free in the WordPress.org Plugin Directory https://wordpress.org/plugins/ or can be cost-based plugin from a third-party. directory, and had a select group of plugin developers test part of the fix. In hindsight, I should have had them test the entire patch – more on this in a bit – but by September we were ready for a release.

4.8.2

The fix was packaged with eight other security fixes (oh yeah, this wasn’t the only thing we’d been working on) and WordPress 4.8.2 was released on September 19th.

The day after release a ticket was opened wanting to add the support for numbered placeholders back to `wpdb::prepare()`. We expected this. We knew that taking away functionality is never popular. Finding the right balance between security and functionality is one of the hardest things we do as a team.

Other than the expected push to reintroduce the functionality we had limited, the release seemed to go pretty well. There were no abnormal spikes in issues on the support forums, no hosts complained of unexpected increases in support requests (and I talked to several), and the actual update numbers looked relatively normal.

Unfortunately, we now know that we broke more plugins than we needed to. If we had given the full patch to our plugin test group, we could have caught some of the potential breakage. That was my fault. I'm sorry, and lesson learned.

One of the plugins that broke was one running on a site belonging to Anthony Ferrara. When he dug into why, he realized that it was due to how we blocked numbered placeholders, and he saw a problem. He opened a HackerOne report on September 20th, the day after 4.8.2 was released.

The New Problem

The new problem, as specified in the report, was that there were some cases where queries using numbered or padded placeholders could have been secure in 4.8.1 and vulnerable in 4.8.2. While I do think that we fixed far more than we broke with 4.8.2, the issue was still valid. A proof of concept was included:

```
$wpdb->prepare("SELECT * FROM foo WHERE name= '%4s' AND user_id = %d", $_GET['name'], get_current_user_id());
```

Because our fix for 4.8.2 didn't allow for `%4s`, this query would no longer try to select the current user with a user-specified name, but instead try to select a user with the name `%%4s` and the user-specified "name" used as the `user_id`.

There was some back and forth on the ticket where we tried to give background on the original issue and why we fixed it the way we did. What we had dubbed "double prepares" *were* in fact an issue, but our fix had left some potential edge cases worse off than before. Obviously not ideal.

We dug in again.

Security is Complicated

The technical issue was relatively easy to agree on: User supplied content – that could resemble a placeholder – ending up in the first parameter of `wpdb::prepare()`. The correct solution though, was much harder to find and agree upon.

One solution that was offered was to disallow all double prepares, making them throw a hard error. Unfortunately, looking through the plugin directory told us that this would likely break a lot of plugins. It was an "ideal" solution from a security standpoint, but realistically it wasn't an option for us.

The solution we came up with, on the other hand, was very complex and convoluted. It tried to keep existing functionality while still mitigating the vulnerability, but the rabbit hole seemed to be extremely deep. The patch became more complex as we discovered unexpected idiosyncrasies with the internal workings of certain PHP PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. https://www.php.net/manual/en/index.php functions.

Communication is Important and Hard

Working with people can often be complicated, and working with our security team on a complex WordPress issue is no exception. Communication broke down some during the process and things got difficult. Luckily both sides kept working at it.

Ultimately, working back and forth through the issues we were able to continue to improve our approach. Implementing ideas from both sides allowed us to simplify the code and come up with a solution that was not prohibitively complex, addressed the issues, and avoided widespread plugin breakage.

There *was* some changed behavior in `esc_sql()`, that mostly affects its use when **not** used in conjunction with `wpdb::query()`.

Success

The resulting fix was released in WordPress 4.8.3 on October 31, 42 days after 4.8.2. Updates rolled out smoothly and the release seems to be working well for people. Thanks to all involved for their hard work on this one!