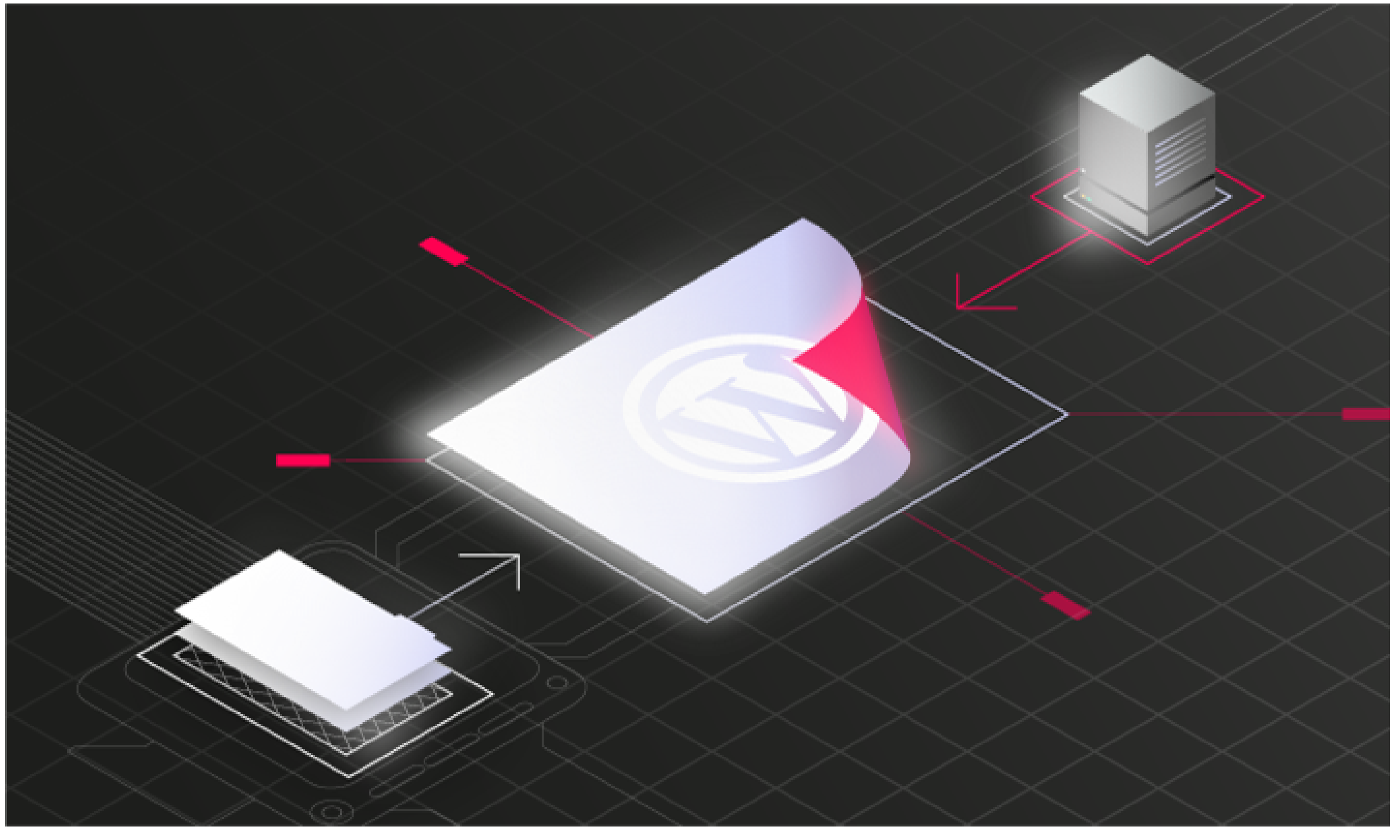
WordPress 5.0.0 Remote Code Execution

15 min read — 19 Feb 2019 by Simon Scannell

This blog post details how a combination of a Path Traversal and Local File Inclusion vulnerability lead to Remote Code Execution in the WordPress core (CVE-2019-8943). The vulnerability remained uncovered in the WordPress core for over **6 years**.



PHP WORDPRESS REMOTE CODE EXECUTION FILE INCLUSION PATH TRAVERSAL VULNERABILITY CMS TOP5

Impact

An attacker who gains access to an account with at least `author` privileges on a target WordPress site can execute arbitrary PHP code on the underlying server, leading to a full remote takeover. We sent the WordPress security team details about another vulnerability in the WordPress core that can give attackers exactly such access to any WordPress site, which is currently unfixed.

Who is affected?

The vulnerability explained in this post was rendered non-exploitable by another security patch in versions **4.9.9** and **5.0.1**. However, the Path Traversal is still possible and currently unpatched. Any WordPress site with a plugin installed that incorrectly handles *Post Meta* entries can make exploitation still possible. We have seen plugins with millions of active installations do this mistake in the past during the preparations for our WordPress security month.

According to the download page of WordPress, the software is used by over 33%[1] of all websites on the internet. Considering that plugins might reintroduce the issue and taking in factors such as outdated sites, the number of affected installations is still in the millions.

Technical Analysis

Both the Path Traversal and Local File Inclusion vulnerability was automatically detected by our leading SAST solution RIPS within 3 minutes scan time with a click of a button. However, at first sight the bugs looked not exploitable. It turned out that the exploitation of the vulnerabilities is much more complex but possible.

SEE RIPS SCAN REPORT

0:00

Background - WordPress Image Management

When an image is uploaded to a WordPress installation, it is first moved to the uploads directory (`wp-content/uploads`). WordPress will also create an internal reference to the image in the database, to keep track of meta information such as the owner of the image or the time of the upload.

This meta information is stored as *Post Meta* entries in the database. Each of these entries are a key / value pair, assigned to a certain ID.

```
Example Post Meta reference to an uploaded image ‘evil.jpg’
```

```
MariaDB [wordpress]> SELECT * FROM wp_postmeta WHERE post_ID = 50;
+---------+-------------------------+---------------------------+
| post_id | meta_key                | meta_value                |
+---------+-------------------------+---------------------------+
|      50 | _wp_attached_file       | evil.jpg                  |
|      50 | _wp_attachment_metadata | a:5:{s:5:"width";i:450 ... |
...
+---------+-------------------------+---------------------------+
```

In this example, the image has been assigned the `post_ID` 50. If the user wants to use or edit the image with said *ID* in the future, WordPress will look up the matching `_wp_attached_file` meta entry and use it's value in order to find the file in the `wp-content/uploads` directory.

Core issue - Post Meta entries can be overwritten

The issue with these *Post Meta* entries prior to WordPress **4.9.9** and **5.0.1** is that it was possible to modify any entries and set them to arbitrary values.

When an image is updated (e.g. it's description is changed), the `edit_post()` function is called. This function directly acts on the `$_POST` array.

Arbitrary Post Meta values can be updated.

```
function edit_post( $post_data = null ) {

    if ( empty($postarr) )
        $postarr = &$_POST;
    ;
    if ( ! empty( $postarr['meta_input'] ) ) {
        foreach ( $postarr['meta_input'] as $field => $value ) {
            update_post_meta( $post_ID, $field, $value );
        }
    }
```

As can be seen, it is possible to inject arbitrary *Post Meta* entries. Since no check is made on which entries are modified, an attacker can update the `_wp_attached_file` meta entry and set it to **any** value. This does not rename the file in any way, it just changes the file WordPress will look for when trying to edit the image. This will lead to a Path Traversal later.

Path Traversal via Modified Post Meta

The Path Traversal takes place in the `wp_crop_image()` function which gets called when a user crops an image.

The function takes the ID of an image to crop (`$attachment_id`) and fetches the corresponding `_wp_attached_file` *Post Meta* entry from the database.

Remember that due to the flaw in `edit_post()`, `$src_file` can be set to anything.

Simplified wp_crop_image() function. The actual code is located in wp-admin/includes/image.php

```
function wp_crop_image( $attachment_id, $src_x, ...) {

```

```
3       $src_file = $file = get_post_meta( $attachment_id, '_wp_attached_file'
4       ;
```

In the next step, WordPress has to make sure the image actually exists and load it. WordPress has two ways of loading the given image. The first is to simply look for the filename provided by the `_wp_attached_file` *Post Meta* entry in the `wp-content/uploads` directory (line 2 of the next code snippet).

If that method fails, WordPress will try to download the image from it's own server as a fallback. To do so it will generate a download URL consisting of the URL of the `wp-content/uploads` directory and the filename stored in the `_wp_attached_file` *Post Meta* entry (line 6).

To give a concrete example: If the value stored in the `_wp_attached_file` *Post Meta* entry was `evil.jpg`, then WordPress would first try to check if the file `wp-content/uploads/evil.jpg` exists. If not, it would try to download the file from the following URL: `https://targetserver.com/wp-content/uploads/evil.jpg`.

The reason for trying to download the image instead of looking for it locally is for the case that some plugin generates the image on the fly when the URL is visited.

Take note here that no sanitization whatsoever is performed here. WordPress will simply concatenate the upload directory and the URL with the `$src_file` user input.

Once WordPress has successfully loaded a valid image via `wp_get_image_editor()`, it will crop the image.

```
1       ;
2       if ( ! file_exists( "wp-content/uploads/" . $src_file ) ) {
 3              // If the file doesn't exist, attempt a URL fopen on the src li
 4              // This can occur with certain file replication plugins.
 5              $uploads = wp_get_upload_dir();
 6              $src = $uploads['baseurl'] . "/" . $src_file;
 7          } else {
 8              $src = "wp-content/uploads/" . $src_file;
 9          }
10
11      $editor = wp_get_image_editor( $src );
12      ⋮
```

The cropped image is then saved back to the filesystem (regardless of whether it was downloaded or not). The resulting filename is going to be the `$src_file` returned by `get_post_meta()`, which is under control of an attacker. The only modification made to the resulting filename string is that the basename of the file is prepended by `cropped-` (line 4 of the next code snippet.) To follow the example of the `evil.jpg`, the resulting filename would be `cropped-evil.jpg`.

WordPress then creates any directories in the resulting path that do not exist yet via `wp_mkdir_p()` (line 6).

It is then finally written to the filesystem using the `save()` method of the image editor object. The `save()` method also performs no Path Traversal checks on the given file name.

```
1    ⋮
2    $src = $editor->crop( $src_x, $src_y, $src_w, $src_h, $dst_w, $dst_h, $s
3
4    $dst_file = str_replace( basename( $src_file ), 'cropped-' . basename( $
5
6    wp_mkdir_p( dirname( $dst_file ) );
```

```
    $result = $editor->save( $dst_file );
```

The idea

So far, we have discussed that it is possible to determine which file gets loaded into the image editor, since no sanitization checks are performed. However, the image editor will throw an exception if the file is not a valid image. The first assumption might be, that it is only possible to crop images outside the uploads directory then.

However, the circumstance that WordPress tries to download the image if it is not found leads to a Remote Code Execution vulnerability.

	Local File	**HTTP Download**
Uploaded file	evil.jpg	evil.jpg
_wp_attached_file	evil.jpg?shell.php	evil.jpg?shell.php
Resulting file that will be loaded	wp-content/uploads/evil.jpg?shell.php	https://targetserver.com/wp-content/uploads/evil.jpg?shell.php
Actual location	wp-content/uploads/evil.jpg	https://targetserver.com/wp-content/uploads/evil.jpg

	Local File	HTTP Download
Resulting filename	None - image loading fails	evil.jpg?cropped-shell.php

The idea is to set `_wp_attached_file` to `evil.jpg?shell.php`, which would lead to a HTTP request being made to the following URL: `https://targetserver.com/wp-content/uploads/evil.jpg?shell.php`. This request would return a valid image file, since everything after the `?` is ignored in this context. The resulting filename would be `evil.jpg?shell.php`.

However, although the `save()` method of the image editor does not check against Path Traversal attacks, it will append the extension of the mime type of the image being loaded to the resulting filename. In this case, the resulting filename would be `evil.jpg?cropped-shell.php.jpg`. This renders the newly created file harmless again.

However, it is still possible to plant the resulting image into any directory by using a payload such as `evil.jpg?/../../evil.jpg`.

Exploiting the Path Traversal - LFI in Theme directory

Each WordPress theme is simply a directory located in the `wp-content/themes` directory of WordPress and provides template files for different cases. For example, if a visitor of a blog wants to view a blog post, WordPress looks for a `post.php` file in the directory of the currently active theme. If it finds the template it will `include()` it.

In order to add an extra layer of customization, it is possible to select a custom template for certain posts. To do so, a user has to set the `_wp_page_template` *Post Meta* entry in the database to such a custom filename. The only limitation here is that the file to be `include()`‘ed must be located in the directory of the currently active theme.

Usually, this directory cannot be accessed and no files can be uploaded. However, by abusing the above described Path Traversal, it is possible to plant a maliciously crafted image into the directory of the currently used theme. The attacker can then create a new post and abuse the same bug that enabled him to update the `_wp_attached_file` *Post Meta* entry in order to `include()` the image. By injecting PHP code into the image, the attacker then gains arbitrary Remote Code Execution.

Crafting a malicious image - GD vs Imagick

WordPress supports two image editing extensions for PHP: GD and Imagick. The difference between them is that Imagick does *not* strip exif metadata of the image, in which PHP code can be stored. GD compresses each image it edits and strips all exif metadata. This is a result of how GD processes images.

However, exploitation is still possible by crafting an image that contains crafted pixels that will be flipped in a way that results in PHP code execution once GD is done cropping the image. During our efforts to research the internal structures of PHP’s GD extension, an exploitable memory corruption flaw was discovered in libgd. (CVE-2019-6977[2]).

Time Line

Date	What
2018/10/16	Vulnerability reported to the WordPress security team on Hackerone.
2018/10/18	A WordPress Security Team member acknowledges the report and says they will come back once the report is verified.
2018/10/19	Another WordPress Security Team member asks for more information.
2018/10/22	We provide WordPress with more information and provide a complete, 270 line exploit script to help verify the vulnerability,
2018/11/15	WordPress triages the vulnerability and says they were able to replicate it.
2018/12/06	WordPress 5.0 is released, without a patch for the vulnerability.
2018/12/12	WordPress 5.0.1 is released and is a security update. One of the patches makes the vulnerabilities non exploitable by preventing attackers to set arbitrary post meta entries. However, the Path Traversal is still possible and can be exploited if plugins are installed that incorrectly handle Post Meta entries. WordPress 5.0.1 does not address either the Path Traversal or Local File Inclusion vulnerability.
2018/12/19	WordPress 5.0.2 is released. without a patch for the vulnerability.

Date	What
2019/01/09	WordPress 5.0.3 is released, without a patch for the vulnerability.
2019/01/28	We ask WordPress for an ETA of the next security release so we can coordinate our blog post schedule and release the blog post after the release.
2019/02/14	WordPress proposes a patch.
2019/02/14	We provide feedback on the patch and verify that it prevents exploitation.

Summary

This blog post detailed a Remote Code Execution in the WordPress core that was present for over **6 years**. It became non-exploitable with a patch for another vulnerability reported by RIPS in versions **5.0.1** and **4.9.9**. However, the Path Traversal is still possible and can be exploited if a plugin is installed that still allows overwriting of arbitrary Post Data. Since certain authentication to a target WordPress site is needed for exploitation, we decided to make the vulnerability public after 4 months of initially reporting the vulnerabilities.

We would like to thank the volunteers of the WordPress security team which have been very friendly and acted professionally when working with us on this issue.

Update: Pwnie Award Nomination

The Pwnie Awards is an annual awards ceremony celebrating outstanding research in different security categories. It is a great honour that our research detailed in this blog post was nominated for the *Best Server-Side Bug* award by a panel of respected security researchers. Once a year this award goes to the researchers who discovered or exploited the most technically sophisticated and interesting server-side bug.

---

1. https://wordpress.org/download/ [return]
2. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-6977 [return]

Related Posts

- Wormable Stored XSS on WordPress.org
- WordPress Privilege Escalation through Post Types
- WordPress Design Flaw Leads to WooCommerce RCE
- WARNING: WordPress File Delete to Code Execution
- osClass 3.6.1: Remote Code Execution via Image File

Simon Scannell

Security Researcher

Simon is a self taught security researcher at RIPS Technologies and is passionate about web application security and coming up with new ways to find and exploit vulnerabilities. He currently focuses on the analysis of popular content management systems and their security architecture.