HOW TO:
FIND WORDPRESS PLUGIN VULNS

BY WPSCAN.COM

Version 1.0
Released June 8th, 2021

WPScan

WPScan is a WordPress security company based in France. WPScan started over ten years ago as a tool for penetration testers to test the security of WordPress websites. Today WPScan manages the WordPress vulnerability database for WordPress core, plugin and theme vulnerabilities. The WordPress vulnerability database is manually updated daily with new vulnerabilities. The vulnerability database contains over 22,835 vulnerabilities recorded since 2014.

wpscan.com

1. INTRODUCTION 4
2. TESTING ENVIRONMENT 6
3. TESTING METHODOLOGY 10
4. CROSS-SITE SCRIPTING (XSS) 13
5. SQL INJECTION 15
6. CROSS-SITE REQUEST FORGERY (CSRF) 18
7. AUTHORISATION 20
8. REMOTE COMMAND EXECUTION (RCE) 23
9. REMOTE PHP CODE EXECUTION (RCE) 25
10. PHP OBJECT INJECTION 27
10. OTHER THINGS TO TEST FOR 29

1. INTRODUCTION

WordPress is by far the most popular Content Management System (CMS) in existence today. W3Techs.com estimates WordPress's market share, compared to other CMSs, to be almost 65% in June 2021. And that WordPress is run on 41.6% of all websites on the web. This upwards trend doesn't seem to be slowing down either, with WordPress's market share continually growing and hitting new market share milestones.

One of WordPress's key strengths is its ability to install third-party plugins. The official WordPress plugin repository has almost 60,000 Open Source plugins written in PHP, which amounts to over 157 million lines of code. And there are also many other plugin repositories, such as Envato, that host and sell many more. At WPScan, we track over 90,000 plugins in our database. That should give you a good idea of the scope of the attack surface, it's huge!

WordPress themes can also be a source of vulnerabilities, and the methods discussed in this book can also be applied to them.

It shouldn't be surprising that the majority of vulnerabilities found in the WordPress ecosystem today are within the plugins. According to our vulnerability database statistics, 88% of our vulnerabilities affect plugins, while only 5% affect WordPress core.

If you want to find vulnerabilities in a WordPress website, a good place to look is in the plugins. And in this book we are going to show you how to find them, whether you are a penetration tester, a bug bounty hunter, a security researcher, or want to test your own WordPress plugin for security issues.

If you do find some vulnerabilities after reading this book, please submit them to our vulnerability database. We can help you in the vulnerability disclosure process by contacting the author of the plugin, provide you with a CVE number, give you credit, ensure the vulnerability information is spread as far and wide as possible, and you can also win prizes, such as OSCP courses, swag, Amazon vouchers and more.

You can submit new vulnerabilities to us via our submission page:

https://wpscan.com/submit

"SUBMIT VULNERABILITIES TO OUR DATABASE TO BE IN WITH A CHANCE TO WIN MONTHLY GIVEAWAYS!"

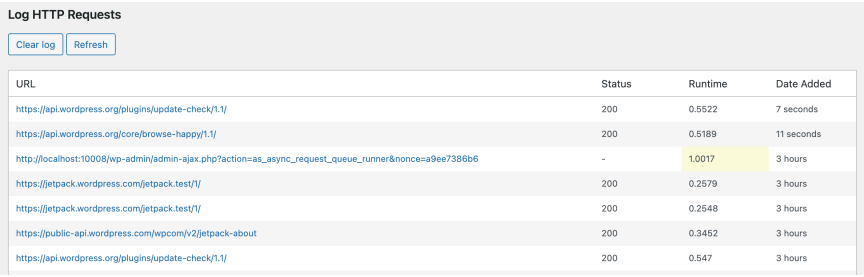
2. TESTING ENVIRONMENT

One of the first steps to finding WordPress plugin vulnerabilities is by setting up a testing environment. But first! You are going to need some tools to help you with your testing. We highly recommend Burp Suite Professional or OWASP ZAP Proxy. These tools will act as a proxy between your web browser and the WordPress application, allowing you to view all the raw HTTP requests and responses.

Once you have installed one of the above testing tools, it's time to install your WordPress testing website. To make this step as easy and quick as possible we would recommend Local by Flywheel or DevKinsta.

Once you have your WordPress testing website up and running, let's install some plugins to help us find vulnerabilities and make some configuration changes.

The Log HTTP Requests WordPress plugin by FacetWP, LLC, will help you view any HTTP requests that are made from the WordPress website. This can be helpful if the plugin that you are testing calls out to an API on the web.



You can also configure WordPress to send all HTTP requests through a proxy, but this can be tricky when you are working with virtual machines, which Local and DevKinsta use. Nevertheless, here is the configuration needed in the wp-config.php file to proxy all of the WordPress websites server-side HTTP requests through your web proxy, Burp or ZAP:

```
define('WP_PROXY_HOST', '127.0.0.1');
define('WP_PROXY_PORT', '8080');
define('WP_PROXY_BYPASS_HOSTS', 'localhost,
www.example.com, *.wordpress.org');
```

The most important WordPress configuration change we are going to make to the wp-config.php file is to prevent WordPress Administrator and Editor user roles from being able to inject HTML, which in WordPress those user roles are allowed to do by default. This will decrease the chances of Cross-Site Scripting (XSS) false positives. *This trips a lot of people new to WordPress security testing up!*

To disable unfiltered html add the following configuration to the wp-config.php file:

```
define( 'DISALLOW_UNFILTERED_HTML', true );
```

For further details about the UNFILTERED_HTML user capability and others, check out the official documentation here;

https://wordpress.org/support/article/roles-and-capabilities/

The second configuration change we are going to make to WordPress is to enable debug logging, this will help you both identify potential vulnerabilities as well as debug them.

To enable debug logging add the following configuration to the wp-config.php file:

```
define( 'WP_DEBUG', true );
define( 'WP_DEBUG_LOG', true );
```

All PHP errors will now be logged to the /wp-content/debug.log file. To create your own logs for researching potential vulnerabilities, you can add the following code to the plugin that you are testing:

```
error_log( 'This is a log' );
```

Another popular tool used for debugging vulnerabilities is Xdebug, which allows you to step through the PHP code while it is executing.

3. TESTING METHODOLOGY

There are two main ways to find vulnerabilities in web applications such as WordPress and its plugins, and they are automated and manually. And these can both be split into dynamic and static testing.

We can use automated web vulnerability scanning tools, like the ones included in Burp Suite and OWASP ZAP. These tools will crawl the site's pages, find forms and other inputs, and try to find vulnerabilities within them. These tools are not perfect, so if the scanner finds a vulnerability then you have to verify that it is not a false positive.

Another automated way of finding vulnerabilities is through static code analysis. The best way to do this is to use WordPress's official coding standard CodeSniffer rules, which contain some security checks.

Getting PHP CodeSniffer setup with the WordPress rules can be cumbersome, but once you have it installed and setup, you can run the following command to just run the security checks against the plugin you are testing:

```
./vendor/bin/phpcs --standard=WordPress --
sniffs=WordPress.CSRF.NonceVerification,WordPress.DB.
PreparedSQL,WordPress.DB.PreparedSQLPlaceholders,Word
Press.DB.RestrictedClasses,WordPress.DB.RestrictedFun
ctions,WordPress.Security.NonceVerification,WordPress
.Security.PluginMenuSlug,WordPress.Security.SafeRedir
ect,WordPress.Security.ValidatedSanitizedInput,WordPr
ess.Security.EscapeOutputSniff,WordPress.WP.PreparedS
QL,WordPress.XSS.EscapeOutput -p -d memory_limit=256M
--colors /path/to/plugin/
```

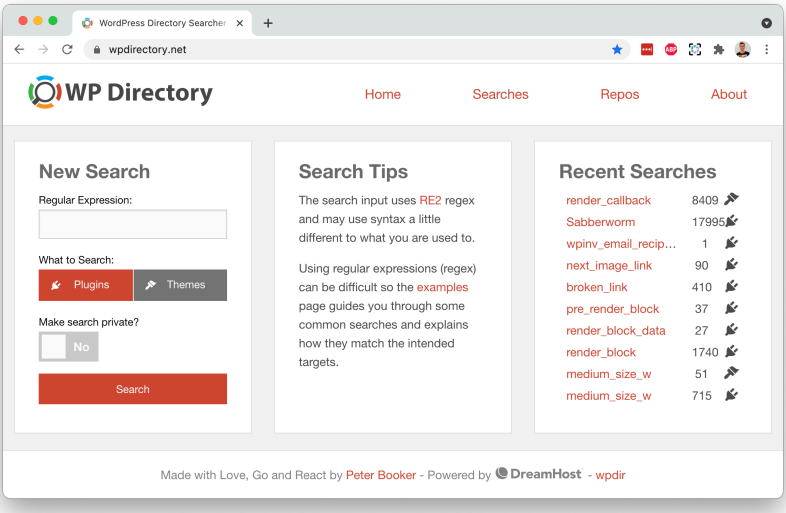
Running the above command against a WordPress plugin is likely to produce a lot of false positives. You will have to go through each finding and manually verify it before submitting the vulnerability to our database.

Finally, another static analysis technique to find vulnerabilities in WordPress plugins is to manually read through the plugin's code to try to spot vulnerabilities. You can use Linux command line tools such as Grep to help, as well as a development environment such as Sublime Text, or others.

Another great way of finding vulnerabilities in WordPress plugins is by manually tampering with the HTTP requests that are intercepted by your testing proxy of choice; Burp or ZAP. This technique is great for finding Cross-Site Request Forgery (CSRF) vulnerabilities, authorisation vulnerabilities, Insecure Direct Object Reference (IDOR) vulnerabilities and business logic vulnerabilities.

Some techniques are better than others at finding certain types of vulnerabilities. To increase your chances of finding WordPress plugin vulnerabilities you should use a combination of all of the techniques discussed above.

You can also use the WP Directory Searcher online tool by Peter Booker to grep for vulnerabilities across the entire WordPress plugin repository. This can be great if you find a vulnerability in one plugin and want to check if others are affected by the same issue.



4. CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting (XSS) is when an attacker can inject malicious JavaScript into the WordPress plugin, which could then potentially be executed in another user's web browser. This is easily the most widespread vulnerability affecting web applications.

Here are some common inputs to get you started that can lead to XSS vulnerabilities that you can grep for:

- $_GET
- $_POST
- $_REQUEST
- $_SERVER['REQUEST_URI']
- $_SERVER['PHP_SELF']
- $_SERVER['HTTP_REFERER']
- $_COOKIE

There's also two WordPress functions that are commonly used that can lead to XSS vulnerabilities, those are:

- add_query_arg()
- remove_query_arg()

These functions add or remove query arguments from URLs. They are insecure because they use $_SERVER['PHP_SELF'] if the supplied second or third parameter is not a URL.

There's one *"XSS gotcha!"* that trips up a lot of new security researchers that are not familiar with the WordPress security ecosystem. And that is that WordPress by default allows Administrator and Editor users to inject arbitrary JavaScript into pages, posts, comments and widgets. This leads to a lot of false positive submissions to our database, which ends up with disappointed security researchers.

The user capability responsible for allowing users to inject arbitrary JavaScript is called unfiltered_html.

To reduce the chances of a Cross-Site Scripting (XSS) vulnerability false positive, you can disable the unfiltered_html capability for all users, as discussed in the Testing Environment chapter. Another way to check if your finding is a false positive is to test the XSS vulnerability with a non Administrator or Editor user, such as an Author user.

Some example XSS vulnerabilities:

- Photo Gallery < 1.5.69 - Multiple Reflected Cross-Site Scripting (XSS)
- WPBakery Page Builder Clipboard < 4.5.6 - Subscriber+ Stored Cross-Site Scripting (XSS)
- Patreon WordPress < 1.7.2 - Reflected XSS on Login Form

5. SQL INJECTION

SQL Injection is when user input is mixed within a SQL query, which allows the attacker to modify the SQL query. This can lead to the WordPress user password hashes being leaked, the site's content being modified, or even complete server compromise.

WordPress and its plugins protect against SQL Injection vulnerabilities by using the $wpdb->prepare() function. If you find a SQL query that is not passed through this function, unless it is within a safe API method listed below, and it has unvalidated user input within it, then it is likely vulnerable to SQL Injection.

The WordPress functions below need to be passed through the $wpdb->prepare() function, otherwise they are potentially vulnerable to SQL Injection:

- $wpdb->query()
- $wpdb->get_var()
- $wpdb->get_row()
- $wpdb->get_col()
- $wpdb->get_results()
- $wpdb->replace()

The following functions are considered as being safe from SQL Injection vulnerabilities:

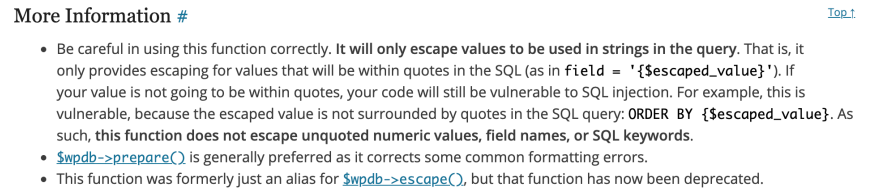
- $wpdb->insert()
- $wpdb->update()
- $wpdb->delete()

Sometimes, even though the plugin developer did not use the $wpdb->prepare() function, you will notice that the plugin developer had attempted to secure their SQL query by using one of the functions below.

- esc_sql()
- escape()
- esc_like()
- like_escape()

However, don't let the function names fool you! They do not adequately protect against SQL Injection, because they will only escape values to be used in strings in the query. If the user input is not surrounded by quotes, then it will be vulnerable to SQL Injection.

The WordPress documentation for the esc_sql() function gives a great explanation of this:



More Information #

Top ↑

- Be careful in using this function correctly. **It will only escape values to be used in strings in the query**. That is, it only provides escaping for values that will be within quotes in the SQL (as in `field = '{$escaped_value}'`). If your value is not going to be within quotes, your code will still be vulnerable to SQL injection. For example, this is vulnerable, because the escaped value is not surrounded by quotes in the SQL query: `ORDER BY {$escaped_value}`. As such, **this function does not escape unquoted numeric values, field names, or SQL keywords**.
- `$wpdb->prepare()` is generally preferred as it corrects some common formatting errors.
- This function was formerly just an alias for `$wpdb->escape()`, but that function has now been deprecated.

The best tool for exploiting SQL Injection vulnerabilities is called sqlmap by Bernardo Damele and Miroslav Stampar. It can automate the exploitation and make it much easier.

Some example SQL Injection vulnerabilities:

- WP Statistics < 13.0.8 - Unauthenticated SQL Injection
- wpDataTables < 3.4.2 - Blind SQL Injection
- RSVPMaker < 7.8.2 - Unauthenticated SQL Injection

6. CROSS-SITE REQUEST FORGERY (CSRF)

Cross-Site Request Forgery (CSRF) is a vulnerability that can allow an attacker to force an authenticated user into taking some kind of action. That action could be to add a new user, for example. To successfully exploit the vulnerability the attacker needs for an authenticated user to click a link, or visit a web page that they control. A CSRF attack works by forcing the authenticated user into submitting a HTTP request.

There are a few different ways to prevent CSRF attacks, but WordPress uses the cryptographic nonce method (a number that can only be used once). CSRF vulnerabilities can affect forms and URLs; POST or GET methods. The following WordPress functions add or verify the nonce value:

- wp_nonce_field()
- wp_nonce_url()
- wp_nonce_field()
- wp_verify_nonce()
- check_admin_referer()
- check_ajax_referer()

The best way to test a WordPress plugin for CSRF vulnerabilities is to use a web proxy such as Burp or ZAP,

submit a form or click a link, then check if the HTTP request has a CSRF nonce parameter when submitted. If the HTTP request does not have a CSRF nonce then it is likely to be vulnerable to a CSRF attack.

Even if the HTTP request does have a CSRF nonce, we often find that the nonce is not checked by the plugin, which defeats the whole purpose of having a cryptographic nonce in the first place. If the HTTP request does have a CSRF nonce, you can try removing the CSRF nonce value, removing the parameter altogether, or just changing the CSRF nonce value. If WordPress does not throw an error when doing one of those, then the plugin is likely to be vulnerable to CSRF attacks.

By default, the WordPress nonce parameters are named _wpnonce or _ajax_nonce, but these can be changed by the WordPress plugin author.

Another thing to note is that WordPress CSRF nonces are user specific, so one user cannot use the nonce of another.

For further details about CSRF and WordPress nonces we recommend the following resources:

- https://codex.wordpress.org/WordPress_Nonces
- https://css-tricks.com/wordpress-front-end-security-csrf-and-nonces/

Some example CSRF vulnerabilities:

- Database Backups <= 1.2.2.6 - CSRF to Backup Download
- NextGen Gallery < 3.5.0 - CSRF allows File Upload
- Ninja Forms < 3.4.24.2 - CSRF to Stored XSS

7. AUTHORISATION

Authentication is handled by WordPress core so it is generally not something that we have to test or even worry about. However, authorisation is a different matter. Authorisation vulnerabilities are when one user can access plugin functionality that they are not supposed to have access to. This can be either two users of the same role, such as changing another user's avatar for example. Or, two users of different roles, allowing an Author user to edit the plugin's settings for example, which should only be available to Administrator users.

One common pitfall for WordPress plugin developers is the is_admin() and is_user_admin() functions. If you thought that these functions checked that your user was an admin then you'd be wrong! These functions merely check that the request was made from a WordPress admin page, which other non-admin WordPress users have access to.

In WordPress, the current_user_can() function should be used to properly check a user's roles and capabilities.

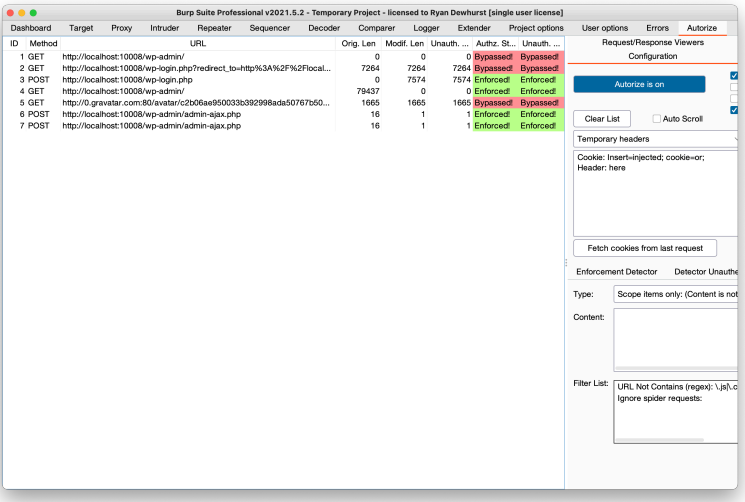
Another common mistake that we see often is the do_action("wp_ajax_nopriv_{$action}") function being applied to sensitive functionality. This action allows unauthenticated AJAX requests to the functionality in question. Sometimes developers accidentally use this

function without realising this. Or, they may think that attackers won't find out that it is used; security through obscurity.

A quick grep for wp_ajax_nopriv_ in the plugin's code is always worthwhile. If you do find some uses of this action, see if any of the functionality is sensitive enough to warrant only being exposed to authenticated users.

Something we often see is authorisation vulnerabilities allowing for other vulnerabilities to be exploited. For example, perhaps the plugin developer thought that they did not need to secure that piece of code because they assumed it could only be accessed by Administrator users.

A good way to test for authorisation vulnerabilities is to use the Autorize Burp Suite extension. You can configure it to swap your Administrator user cookie for another users' and easily see the results.



Some example authorisation vulnerabilities:

- MStore API < 3.2.0 - Authentication Bypass With Sign In With Apple
- GiveWp < 2.5.5 - Authentication Bypass
- WP Database Reset < 3.15 - Unauthenticated Database Reset

8. REMOTE COMMAND EXECUTION (RCE)

Remote Command Execution (RCE) is probably one of the most dangerous vulnerabilities that can affect a WordPress plugin as it allows attackers to run commands on the Operating System (OS).

The PHP functions below all allow arbitrary commands to be run on the OS. If a WordPress plugin allows user input to be passed to one of these, without first being sanitised then it can lead to RCE.

- system()
- exec()
- passthru()
- shell_exec()

These functions can be grepped for within the plugin's code. They are generally not so commonly used as plugin developers know of their dangers. But they do come across from time to time!

Some example Remote Command Execution vulnerabilities:

- Plainview Activity Monitor <= 20161228 - Remote Command Execution (RCE)
- Cool Video Gallery <= 1.9 - Authenticated Command Injection
- Form Manager <= 1.7.2 - Authenticated Remote Command Execution (RCE)

9. REMOTE PHP CODE EXECUTION (RCE)

Remote PHP Code Execution (RCE) is again probably one of the most dangerous vulnerabilities that can affect a WordPress plugin as it allows attackers to arbitrary PHP code on the web server.

The PHP functions below all allow arbitrary PHP to be run on the web server. If a WordPress plugin allows user input to be passed to one of these, without first being sanitised then it can lead to RCE.

- eval()
- assert()
- preg_replace()
- php://input
- call_user_func()

These functions can be grepped for within the plugin's code. They are generally not so commonly used as plugin developers know of their dangers. But they do come across from time to time!

Some example Remote Code Execution vulnerabilities:

- Social Warfare <= 3.5.2 - Unauthenticated Remote Code Execution (RCE)
- WP Super Cache < 1.7.2 - Authenticated Remote Code Execution (RCE)
- Secure File Manager < 2.8.2 - Authenticated Remote Code Execution

10. PHP OBJECT INJECTION

PHP Objection Injection is arguably one of the more difficult vulnerabilities to exploit in the real world. But they are still relatively easy to discover with the right tools and methodologies. Over the past few years the vulnerability has gained in popularity, with many WordPress plugins, and even WordPress itself, being affected.

The two functions to look out for in WordPress that can cause an Object Injection vulnerability are:

- unserialize()
- maybe_unserialize()

If user supplied input is passed to either of these functions without any prior validation, then it is likely that you have found a PHP Object Injection vulnerability.

The other way of finding Object Injection vulnerabilities within WordPress is to use this Burp Suite extension along with this WordPress plugin.

Once the Burp Suite Extension and the WordPress plugin is installed, you can use Burp's automated Scanner to scan the

plugin. If a PHP Object Injection is found it will appear in Burp's findings.

There's a great YouTube video linked below by IppSec titled "Intro to PHP Deserialization / Object Injection" if you want to learn more about how the vulnerability works and how to exploit it.

https://www.youtube.com/watch?v=HaW15aMzBUM

Some example PHP Object Injection vulnerabilities:

- WordPress 3.7 to 5.7.1 - Object Injection in PHPMailer
- Ultimate Reviews < 2.1.33 - Unauthenticated PHP Object Injection
- GDPR CCPA Compliance Support < 2.4 - Unauthenticated PHP Object Injection

10. OTHER THINGS TO TEST FOR

Privilege Escalation

User supplied input passed to either of these WordPress functions can lead to privilege escalation vulnerabilities.

- update_option()
- do_action()

There's an example vulnerability that was identified in the WP GDPR Compliance Plugin by Wordfence linked below:

https://www.wordfence.com/blog/2018/11/privilege-escalation-flaw-in-wp-gdpr-compliance-plugin-exploited-in-the-wild/

File Inclusion

PHP allows multiple ways to include files, these could be vulnerable to Local or Remote File Inclusion vulnerabilities if unvalidated user supplied input is passed to any of the following functions:

- include()
- require()

- include_once()
- require_once()
- fread()

File Upload

The sanitize_file_name() WordPress function can help in bypassing filename validation in some circumstances. For example, passing “test.(php)” would return the valid filename “test.php”.

- sanitize_file_name()

File Download

The following functions can lead to arbitrary file download if user supplied input is passed to them without proper validation:

- file()
- readfile()
- file_get_contents()

Open Redirects

The wp_redirect() WordPress function does not validate the host and therefore can be vulnerable to Open Redirects. It is recommended that the wp_safe_redirect() function be used instead, which does validate hosts.

- wp_redirect()

SSL/TLS Issues

It is important to check if any communications made by the plugin are using secure transport standards.

- CURLOPT_SSL_VERIFYHOST if set to 0 then does not check name in the host certificate.

- CURLOPT_SSL_VERIFYPEER if set to FALSE then does not check if the certificate (inc chain), is trusted. A Man-in-the-Middle (MitM) attacker could use a self-signed certificate.

Also, check if plaintext HTTP is used to communicate with backend servers or APIs. A grep for "http://" should be sufficient.

WPScan

We hope that you have enjoyed this book and that it has helped you find WordPress plugin vulnerabilities, either in your own code, or as part of a bug bounty or penetration test.

This e-book will be constantly updated with subsequent versions as new research and findings around WordPress security come to light. So keep an eye out for new versions!

 https://twitter.com/_wpscan_

 https://www.facebook.com/WPScan

 https://www.linkedin.com/company/wpscan/