PHP Mussel

Documentation for phpMussel (English)

Max Base

Last Updated: 26 May 2019 (2019.05.26).
MAX BASE, ASREZ TEAM, PHPMUSSEL TEAM
ASREZ.COM, PHPMUSSEL.GITHUB.IO
Writing by Caleb Mazalevskis, Max Base
https://phpmussel.github.io/

First release, Jun 2019

Contents

1. PHP Mussel

1.1 Introduction

1.1.1 What is phpMussel?

An ideal solution for shared hosting environments, where it's often not possible to utilise or install conventional anti-virus protection solutions, phpMussel is a PHP script designed to **detect trojans, viruses, malware and other threats** within files uploaded to your system wherever the script is hooked, based on the signatures of ClamAV and others. For information regarding *HOW TO INSTALL* {2A+2B} and *HOW TO USE* {3A+3B} phpMussel, please refer either to the Wiki or to the documentation (direct links to that documentation included under the "Documentation" header below this paragraph).

Features:

- Licensed as GNU General Public License version 2.0 (GPLv2).
- Easy to install, easy to customise, easy to use.
- Works for any system with PHP+PCRE installed, regardless of OS (PHP+PCRE required).
- Fully configurable based on your needs.
- Ideal solution for shared hosting services.
- Ideal solution for forum systems in need of file upload protection.
- Does NOT require shell access.
- Does NOT require administrative privileges.
- CLI mode available.
- Good, strong, stable support base.

1.2 Preamble

Thank you for using phpMussel, a PHP script designed to detect trojans, viruses, malware, and other threats within files uploaded to your system wherever the script is hooked, based on the signatures of ClamAV and others.

Special thanks to ClamAV both for project inspiration and for the signatures that this script utilises, without which, the script would likely not exist, or at best, would have very limited value.

Special thanks to SourceForge, Bitbucket, and GitHub for hosting the project files, and to the additional sources of a number of the signatures utilised by phpMussel: PhishTank, NLNetLabs, Malware.Expert and others, and special thanks to all those supporting the project, to anyone else that I may have otherwise forgotten to mention, and to you, for using the script.

This document and its associated package can be downloaded for free from: - GitHub. - Bitbucket. - SourceForge.

1.3 How to install

1.3.1 Installing manually (for web servers)

1) By your reading this, I'm assuming you've already downloaded an archived copy of the script, decompressed its contents and have it sitting somewhere on your local machine. From here, you'll want to work out where on your host or CMS you want to place those contents. A directory such as `/public_html/phpmussel/` or similar (though, it doesn't matter which you choose, so long as it's something secure and something you're happy with) will suffice. *Before you begin uploading, read on..*
2) Rename `config.ini.RenameMe` to `config.ini` (located inside `vault`), and optionally (strongly recommended for advanced users, but not recommended for beginners or for the inexperienced), open it (this file contains all the directives available for phpMussel; above each option should be a brief comment describing what it does and what it's for). Adjust these directives as you see fit, as per whatever is appropriate for your particular setup. Save file, close.
3) Upload the contents (phpMussel and its files) to the directory you'd decided on earlier (you don't need to include the `*.txt`/`*.md` files, but mostly, you should upload everything).
4) CHMOD the `vault` directory to "755" (if there are problems, you can try "777"; this is less secure, though). The main directory storing the contents (the one you chose earlier), usually, can be left alone, but CHMOD status should be checked if you've had permissions issues in the past on your system (by default, should be something like "755"). In short: For the package to work properly, PHP needs to be able to read and write files inside the `vault` directory. Many things (updating, logging, etc) won't be possible, if PHP can't write to the `vault` directory, and the package won't work at all if PHP can't read from the `vault` directory. However, for optimal security, the `vault` directory must NOT be publicly accessible (sensitive information, such as the information contained by `config.ini` or `frontend.dat`, could be exposed to potential attackers if the `vault` directory is publicly accessible).
5) Install any signatures that you'll need. *See: INSTALLING SIGNATURES.*

6) Next, you'll need to "hook" phpMussel to your system or CMS. There are several different ways you can "hook" scripts such as phpMussel to your system or CMS, but the easiest is to simply include the script at the beginning of a core file of your system or CMS (one that'll generally always be loaded when someone accesses any page across your website) using a `require` or `include` statement. Usually, this'll be something stored in a directory such as `/includes`, `/assets` or `/functions`, and will often be named something like `init.php`, `common_functions.php`, `functions.php` or similar. You'll have to work out which file this is for your situation; If you encounter difficulties in determining this for yourself, visit the phpMussel issues page at GitHub or the phpMussel support forums for assistance; It's possible that either myself or another user may have experience with the CMS that you're using (you'll need to let us know which CMS you're using), and thus, may be able to provide some assistance in this area. To do this [to use `require` or `include`], insert the following line of code to the very beginning of that core file, replacing the string contained inside the quotation marks with the exact address of the `loader.php` file (local address, not the HTTP address; it'll look similar to the vault address mentioned earlier).

```
<?php require '/user_name/public_html/phpmussel/loader.php'; ?>
```

Save file, close, reupload.

– OR ALTERNATIVELY –

If you're using an Apache webserver and if you have access to `php.ini`, you can use the `auto_prepend_file` directive to prepend phpMussel whenever any PHP request is made. Something like:

```
auto_prepend_file = "/user_name/public_html/phpmussel/loader.php"
```

Or this in the `.htaccess` file:

```
php_value auto_prepend_file "/user_name/public_html/phpmussel/loader.php"
```

7) At this point, you're done! However, you should probably test it out to make sure it's working properly. To test out file upload protections, attempt to upload the testing files included in the package under `_testfiles` to your website via your usual browser-based upload methods. (Make sure you've included the `phpmussel*.*db` signature files in your `Active` setting for the test files to trigger). If everything is working, a message should appear from phpMussel confirming that the upload was successfully blocked. If nothing appears, something isn't working correctly. If you're using any advanced features or if you're using the other types of scanning possible with the tool, I'd suggest trying it out with those to make sure it works as expected, too.

1.3.2 Installing manually (for CLI)

1) By your reading this, I'm assuming you've already downloaded an archived copy of the script, decompressed its contents and have it sitting somewhere on your local machine. When you've determined that you're happy with the location chosen for phpMussel, continue.
2) phpMussel requires PHP to be installed on the host machine in order to execute. If you don't have PHP installed on your machine, please install PHP on your machine, following any instructions supplied by the PHP installer.
3) Optionally (strongly recommended for advanced users, but not recommended for beginners or for the inexperienced), open `config.ini` (located inside `vault`) – This file contains all the directives available for phpMussel. Above each option should be a brief comment describing what it does and what it's for. Adjust these options as you see fit, as per whatever is

appropriate for your particular setup. Save file, close.

4) Optionally, you can make using phpMussel in CLI mode easier for yourself by creating a batch file to automatically load PHP and phpMussel. To do this, open a plain text editor such as Notepad or Notepad++, type the complete path to the `php.exe` file in the directory of your PHP installation, followed by a space, followed by the complete path to the `loader.php` file in the directory of your phpMussel installation, save the file with a `.bat` extension somewhere that you'll find it easily, and double-click on that file to run phpMussel in the future.
5) Install any signatures that you'll need. *See: INSTALLING SIGNATURES.*
6) At this point, you're done! However, you should probably test it out to make sure it's working properly. To test phpMussel, run phpMussel and try scanning the `_testfiles` directory provided with the package.

1.3.3 Installing with composer

phpMussel is registered with Packagist, and so, if you're familiar with Composer, you can use Composer to install phpMussel (you'll still need to prepare the configuration, permissions, signatures and hooks though; see "installing manually (for web servers)" steps 2, 4, 5, and 6).

```
composer require phpmussel/phpmussel
```

1.3.4 Installing signatures

Since v1.0.0, signatures aren't included in the phpMussel package. Signatures are required by phpMussel for detecting specific threats. There are 3 main methods to install signatures:

1. Install automatically using the front-end updates page.
2. Generate signatures using "SigTool" and install manually.
3. Download signatures from "phpMussel/Signatures" and install manually.

2.3.1 Install automatically using the front-end updates page. Firstly, you'll need to make sure that the front-end is enabled. *See: FRONT-END MANAGEMENT.*

Then, all you'll need to do is go to the front-end updates page, find the necessary signature files, and using the options provided on the page, install them, and activate them.

2.3.2 Generate signatures using "SigTool" and install manually. *See: SigTool documentation.*

2.3.3 Download signatures from "phpMussel/Signatures" and install manually. Firstly, go to phpMussel/Signatures. The repository contains various GZ-compressed signature files. Download the files that you need, decompress them, and copy the decompressed files to the `/vault/signatures` directory to install them. List the names of the copied files to the `Active` directive in your phpMussel configuration to activate them.

1.4 How to use

1.4.1 How to use (for web servers)

phpMussel should be able to operate correctly with minimal requirements on your part: After installing it, it should work immediately and be immediately usable.

Results	Description
-4	Indicates that data couldn't be scanned due to encryption.
-3	Indicates that problems were encountered with the phpMussel signatures files.
-2	Indicates that corrupt data was detected during the scan and thus the scan failed to complete.
-1	Indicates that extensions or addons required by PHP to execute the scan were missing and thus the scan faile
0	Indicates that the scan target doesn't exist and thus there was nothing to scan.
1	Indicates that the target was successfully scanned and no problems were detected.
2	Indicates that the target was successfully scanned and problems were detected.

File upload scanning is automated and enabled by default, so nothing is required on your behalf for this particular functionality.

However, you're also able to instruct phpMussel to scan specific files, directories and/or archives. To do this, firstly, you'll need to ensure that the appropriate configuration is set in the `config.ini` file (`cleanup` must be disabled), and when done, in a PHP file that's hooked to phpMussel, use the following closure in your code:

```
$phpMussel['Scan']($what_to_scan, $output_type, $output_flatness);
```

- `$what_to_scan` can be a string, an array, or an array of arrays, and indicates which file, files, directory and/or directories to scan.
- `$output_type` is a boolean, indicating the format for the scan results to be returned as. `false` instructs the function to return results as an integer. `true` instructs the function to return results as human readable text. Additionally, in either case, the results can be accessed via global variables after scanning has completed. This variable is optional, defaulting to `false`. The following describes the integer results:
- `$output_flatness` is a boolean, indicating to the function whether to return the results of scanning (when there are multiple scan targets) as an array or a string. `false` will return the results as an array. `true` will return the results as a string. This variable is optional, defaulting to `false`.

Examples:

```
$results = $phpMussel['Scan']('/user_name/public_html/my_file.html', true, true)
echo $results;
```

Returns something like this (as a string):

```
Wed, 16 Sep 2013 02:49:46 +0000 Started.
> Checking '/user_name/public_html/my_file.html':
-> No problems found.
Wed, 16 Sep 2013 02:49:47 +0000 Finished.
```

For a full break-down of what sort of signatures phpMussel uses during its scans and how it handles these signatures, refer to the SIGNATURE FORMAT section of this README file.

If you encounter any false positives, if you encounter something new that you think should be blocked, or for anything else regarding signatures, please contact me about it so that I may make the necessary changes, which, if you do not contact me, I may not necessarily be aware of. *(See: What is a "false positive"?).*

To disable specific signatures included with phpMussel (such as, if you're experiencing a false positive specific to your purposes that shouldn't normally be removed from mainline), add the names of the specific signature to be disabled to the signatures greylist file (`/vault/greylist.csv`), separated by commas.

See also: How to access specific details about files when they are scanned?

1.4.2 How to use (for CLI)

Please refer to the "INSTALLING MANUALLY (FOR CLI)" section of this README file.

Also be aware that phpMussel is an *on-demand* scanner; It is *NOT* an *on-access* scanner (other than for file uploads, at the time of upload), and unlike conventional anti-virus suites, doesn't monitor active memory! It'll only detect viruses contained by file uploads, and by those specific files that you explicitly tell it to scan.

1.5 Front-End Management

1.5.1 What is the Front-End.

The front-end provides a convenient and easy way to maintain, manage, and update your phpMussel installation. You can view, share, and download logfiles via the logs page, you can modify configuration via the configuration page, you can install and uninstall components via the updates page, and you can upload, download, and modify files in your vault via the file manager.

The front-end is disabled by default in order to prevent unauthorised access (unauthorised access could have significant consequences for your website and its security). Instructions for enabling it are included below this paragraph.

1.5.2 How to enable the Front-End.

1) Locate the `disable_frontend` directive inside `config.ini`, and set it to `false` (it will be `true` by default).
2) Access `loader.php` from your browser (e.g., `http://localhost/phpmussel/loader.php`).
3) Log in with the default username and password (admin/password).

Note: After you've logged in for the first time, in order to prevent unauthorised access to the front-end, you should immediately change your username and password! This is very important, because it's possible to upload arbitrary PHP code to your website via the front-end.

Also, for optimal security, enabling "two-factor authentication" for all front-end accounts is strongly recommended (instructions provided below).

1.5.3 How to use the Front-End.

Instructions are provided on each page of the front-end, to explain the correct way to use it and its intended purpose. If you need further explanation or any special assistance, please contact support. Alternatively, there are some videos available on YouTube which could help by way of demonstration.

1.5.4 Two-factor authentication

It's possible to make the front-end more secure by enabling two-factor authentication ("2FA"). When logging into a 2FA-enabled account, an email is sent to the email address associated with that account. This email contains a "2FA code", which the user must then enter, in addition to the

username and password, in order to be able to log in using that account. This means that obtaining an account password would not be enough for any hacker or potential attacker to be able to log into that account, as they would also need to already have access to the email address associated with that account in order to be able to receive and utilise the 2FA code associated with the session, thus making the front-end more secure.

Firstly, to enable two-factor authentication, using the front-end updates page, install the PHPMailer component. phpMussel utilises PHPMailer for sending emails. It should be noted that although phpMussel, by itself, is compatible with PHP >= 5.4.0, PHPMailer requires PHP >= 5.5.0, therefore meaning that enabling two-factor authentication for the phpMussel front-end won't be possible for PHP 5.4 users.

After you've installed PHPMailer, you'll need to populate the configuration directives for PHPMailer via the phpMussel configuration page or configuration file. More information about these configuration directives is included in the configuration section of this document. After you've populated the PHPMailer configuration directives, set `enable_two_factor` to `true`. Two-factor authentication should now be enabled.

Next, you'll need to associate an email address with an account, so that phpMussel knows where to send 2FA codes when logging in with that account. To do this, use the email address as the username for the account (like `foo@bar.tld`), or include the email address as part of the username in the same way that you would when sending an email normally (like `Foo Bar <foo@bar.tld>`).

Note: Protecting your vault against unauthorised access (e.g., by hardening your server's security and public access permissions), is particularly important here, due to that unauthorised access to your configuration file (which is stored in your vault), could risk exposing your outbound SMTP settings (including SMTP username and password). You should ensure that your vault is properly secured before enabling two-factor authentication. If you're unable to do this, then at least, you should create a new email account, dedicated for this purpose, as such to reduce the risks associated with exposed SMTP settings.

1.6 CLI (command line interface)

phpMussel can be run as an interactive file scanner in CLI mode under Windows-based systems. Refer to the "HOW TO INSTALL (FOR CLI)" section of this README file for more details.

For a list of available CLI commands, at the CLI prompt, type 'c', and press Enter.

Additionally, for those interested, a video tutorial for how to use phpMussel in CLI mode is available here: - `https://youtu.be/H-Pa740-utc`

1.7 Files included in this package

The following is a list of all of the files that should have been included in the archived copy of the package when you downloaded it, any files that may be potentially created as a result of your using it, along with a short description of what all these files are for.

This information applies to the most recent v2 release, and mightn't be so relevant for other versions or releases.

Filename may differ based on configuration stipulations (in `config.ini`).

File
/*testfiles/* | *Test files directory (contains various files). All contained files are test files for testing if phpMussel was corr*
/*testfiles/coex*testfile.rtf
/*testfiles/exe*standard_testfile.exe
/*testfiles/general*standard_testfile.txt
/*testfiles/graphics*standard_testfile.gif
/*testfiles/html*standard_testfile.html
/*testfiles/md5*testfile.txt
/*testfiles/ole*testfile.ole
/*testfiles/pdf* standard_testfile.pdf
/*testfiles/pe*sectional_testfile.exe
/*testfiles/swf* standard_testfile.swf
/vault/
/vault/cache/
/vault/cache/.htaccess
/vault/classes/
/vault/classes/Maikuolan/
/vault/classes/Maikuolan/Cache.php
/vault/classes/Maikuolan/ComplexStringHandler.php
/vault/classes/Maikuolan/L10N.php
/vault/classes/Maikuolan/YAML.php
/vault/classes/.htaccess
/vault/classes/ArchiveHandler.php
/vault/classes/CompressionHandler.php
/vault/classes/TemporaryFileHandler.php
/vault/fe_assets/
/vault/fe_assets/.htaccess
/vault/fe_assets/2*fa.html* | *An HTML template used when asking the user for a 2FA code. /vault/fe*assets/*accounts.html*
/vault/fe_assets/*cache.html* | *An HTML template for the front-end cache data page. /vault/fe*assets/*config.html* | *An HT*
/vault/fe_assets/*files.html* | *An HTML template for the file manager. /vault/fe*assets/*files*edit.html
/vault/fe_assets/*files*rename.html
/vault/fe_assets/*files*row.html
/vault/fe_assets/*home.html* | *An HTML template for the front-end homepage. /vault/fe*assets/*login.html* | *An HTML tem*
/vault/fe_assets/*nav*logs_access_only.html
/vault/fe_assets/*quarantine.html* | *An HTML template for the front-end quarantine page. /vault/fe*assets/*quarantine*row
/vault/fe_assets/*siginfo.html* | *An HTML template for the front-end signature information page. /vault/fe*assets/*siginfo*
/vault/fe_assets/*statistics.html* | *An HTML template for the front-end statistics page. /vault/fe*assets/*updates.html* | *An H*
/vault/fe_assets/*upload*test.html
/vault/fe_assets/frontend.css
/vault/fe_assets/frontend.dat
/vault/fe_assets/frontend.dat.safety
/vault/fe_assets/frontend.html
/vault/fe_assets/icons.php
/vault/fe_assets/pips.php
/vault/fe_assets/scripts.js
/vault/lang/
/vault/lang/.htaccess
/vault/lang/lang.ar.fe.php
/vault/lang/lang.ar.php
/vault/lang/lang.bn.fe.php
/vault/lang/lang.bn.php
/vault/lang/lang.de.fe.php

general
cleanupscan_logscan_log_serializedscan_killstruncatelog_rotation_limitlog_rotation_actiontimezonetime_offsettime_
compatibility
ignore_upload_errorsonly_allow_images
legal
pseudonymise_ip_addressesprivacy_policy

1.8 Configuration options

The following is a list of variables found in the `config.ini` configuration file of phpMussel, along with a description of their purpose and function.

"general" (Category)
General phpMussel configuration.

"cleanup"
- Unset variables and cache used by the script after the initial upload scanning? False = No; True = Yes [Default]. If you *aren't* using the script beyond the initial scanning of uploads, you should set this to `true` (yes), to minimize memory usage. If you *are* using the script beyond the initial scanning of uploads, should set to `false` (no), to avoid unnecessarily reloading duplicate data into memory. In general practice, it should usually be set to `true`, but, if you do this, you won't be able to use the script for anything other than the initial file upload scanning.
- Has no influence in CLI mode.

"scan_log"
- Filename of file to log all scanning results to. Specify a filename, or leave blank to disable.

"scan_log_serialized"
- Filename of file to log all scanning results to (using a serialised format). Specify a filename, or leave blank to disable.

"scan_kills"
- Filename of file to log all records of blocked or killed uploads to. Specify a filename, or leave blank to disable.

Useful tip: If you want, you can append date/time information to the names of your logfiles by including these in the name: `{yyyy}` for complete year, `{yy}` for abbreviated year, `{mm}` for month, `{dd}` for day, `{hh}` for hour.

Examples: - `scan_log='scan_log.{yyyy}-{mm}-{dd}-{hh}.txt'` - `scan_log_serialized='scan_log_se` - `scan_kills='scan_kills.{yyyy}-{mm}-{dd}-{hh}.txt'`

"truncate"
- Truncate logfiles when they reach a certain size? Value is the maximum size in B/KB/MB/GB/TB that a logfile may grow to before being truncated. The default value of 0KB disables truncation (logfiles can grow indefinitely). Note: Applies to individual logfiles! The size of logfiles is not considered collectively.

“log_rotation_limit”

- Log rotation limits the number of logfiles that should exist at any one time. When new logfiles are created, if the total number of logfiles exceeds the specified limit, the specified action will be performed. You can specify the desired limit here. A value of 0 will disable log rotation.

“log_rotation_action”

- Log rotation limits the number of logfiles that should exist at any one time. When new logfiles are created, if the total number of logfiles exceeds the specified limit, the specified action will be performed. You can specify the desired action here. Delete = Delete the oldest logfiles, until the limit is no longer exceeded. Archive = Firstly archive, and then delete the oldest logfiles, until the limit is no longer exceeded.

Technical clarification: In this context, “oldest” means least recently modified.

“timezone”

- This is used to specify which timezone phpMussel should use for date/time operations. If you don’t need it, ignore it. Possible values are determined by PHP. It’s generally recommended instead to adjust the timezone directive in your `php.ini` file, but sometimes (such as when working with limited shared hosting providers) this isn’t always possible to do, and so, this option is provided here.

“time_offset”

- *v1: “timeOffset”*
- If your server time doesn’t match your local time, you can specify an offset here to adjust the date/time information generated by phpMussel according to your needs. It’s generally recommended instead to adjust the timezone directive in your `php.ini` file, but sometimes (such as when working with limited shared hosting providers) this isn’t always possible to do, and so, this option is provided here. Offset is in minutes.
- Example (to add one hour): `time_offset=60`

“time_format”

- *v1: “timeFormat”*
- The date/time notation format used by phpMussel. Default = `{Day}, {dd} {Mon} {yyyy} {hh}:{ii}:{ss} {tz}`.

“ipaddr”

- Where to find the IP address of connecting requests? (Useful for services such as Cloudflare and the likes) Default = REMOTE_ADDR. WARNING: Don’t change this unless you know what you’re doing!

Recommended values for “ipaddr”:

“enable_plugins”

- Enable support for phpMussel plugins? False = No; True = Yes [Default].

Value	Using
`HTTP_INCAP_CLIENT_IP`	Incapsula reverse proxy.
`HTTP_CF_CONNECTING_IP`	Cloudflare reverse proxy.
`CF-Connecting-IP`	Cloudflare reverse proxy (alternative; if the above doesn't work).
`HTTP_X_FORWARDED_FOR`	Cloudbric reverse proxy.
`X-Forwarded-For`	Squid reverse proxy.
Defined by server configuration.	Nginx reverse proxy.
`REMOTE_ADDR`	No reverse proxy (default value).

"forbid_on_block"

- Should phpMussel send 403 headers with the file upload blocked message, or stick with the usual 200 OK? False = No (200); True = Yes (403) [Default].

"delete_on_sight"

- Enabling this directive will instruct the script to attempt to immediately delete any scanned attempted file upload matching any detection criteria, whether via signatures or otherwise. Files determined to be "clean" won't be touched. In the case of archives, the entire archive will be deleted, regardless of whether or not the offending file is only one of several files contained within the archive. For the case of file upload scanning, usually, it isn't necessary to enable this directive, because usually, PHP will automatically purge the contents of its cache when execution has finished, meaning it'll usually delete any files uploaded through it to the server unless they've been moved, copied or deleted already. This directive is added here as an extra measure of security for those whose copies of PHP mightn't always behave in the manner expected. False = After scanning, leave the file alone [Default]; True = After scanning, if not clean, delete immediately.

"lang"

- Specify the default language for phpMussel.

"numbers"

- Specifies how to display numbers.

Currently supported values:

Note: These values aren't standardised anywhere, and probably won't be relevant beyond the package. Also, supported values may change in the future.

"quarantine_key"

- phpMussel is able to quarantine flagged attempted file uploads in isolation within the phpMussel vault, if this is something you want it to do. Casual users of phpMussel that simply wish to protect their websites or hosting environment without having any interest in deeply analysing any flagged attempted file uploads should leave this functionality disabled, but any users interested in further analysis of flagged attempted file uploads for malware research or for similar such things should enable this functionality. Quarantining of flagged attempted file uploads can sometimes also assist in debugging false positives, if this is something that frequently occurs for you. To disable quarantine functionality, simply leave the `quarantine_key` directive empty, or erase the contents of that directive if it isn't already empty. To enable

Value	Produces	Description
NoSep-1	1234567.89	
NoSep-2	1234567,89	
Latin-1	1,234,567.89	Default value.
Latin-2	1234567.89	
Latin-3	1.234.567,89	
Latin-4	1234567,89	
Latin-5	1,234,567ů89	
China-1	123,4567.89	
India-1	12,34,567.89	
India-2	,,.	
Bengali-1	,,.	
Arabic-1		
Arabic-2		
Thai-1	,,.	

quarantine functionality, enter some value into the directive. The `quarantine_key` is an important security feature of the quarantine functionality required as a means of preventing the quarantine functionality from being exploited by potential attackers and as a means of preventing any potential execution of data stored within the quarantine. The `quarantine_key` should be treated in the same manner as your passwords: The longer the better, and guard it tightly. For best effect, use in conjunction with `delete_on_sight`.

"quarantine_max_filesize"

- The maximum filesize allowed for files to be quarantined. Files larger than the value specified will NOT be quarantined. This directive is important as a means of making it more difficult for any potential attackers to flood your quarantine with unwanted data potentially causing run-away data usage on your hosting service. Default = 2MB.

"quarantine_max_usage"

- The maximum memory usage allowed for the quarantine. If the total memory used by the quarantine reaches this value, the oldest quarantined files will be deleted until the total memory used no longer reaches this value. This directive is important as a means of making it more difficult for any potential attackers to flood your quarantine with unwanted data potentially causing run-away data usage on your hosting service. Default = 64MB.

"quarantine_max_files"

- The maximum number of files that can exist in the quarantine. When new files are added to the quarantine, if this number is exceeded, old files will be deleted until the remainder no longer exceeds this number. Default = 100.

"honeypot_mode"

- When honeypot mode is enabled, phpMussel will attempt to quarantine every single file upload that it encounters, regardless of whether or not the file being uploaded matches any included signatures, and no actual scanning or analysis of those attempted file uploads will

actually occur. This functionality should be useful for those that wish to use phpMussel for the purposes of virus/malware research, but it's neither recommended to enable this functionality if the intended use of phpMussel by the user is for actual file upload scanning, nor recommended to use the honeypot functionality for purposes other than honeypotting. By default, this option is disabled. False = Disabled [Default]; True = Enabled.

"scan_cache_expiry"
- For how long should phpMussel cache the results of scanning? Value is the number of seconds to cache the results of scanning for. Default is 21600 seconds (6 hours); A value of 0 will disable caching the results of scanning.

"disable_cli"
- Disable CLI mode? CLI mode is enabled by default, but can sometimes interfere with certain testing tools (such as PHPUnit, for example) and other CLI-based applications. If you don't need to disable CLI mode, you should ignore this directive. False = Enable CLI mode [Default]; True = Disable CLI mode.

"disable_frontend"
- Disable front-end access? Front-end access can make phpMussel more manageable, but can also be a potential security risk, too. It's recommended to manage phpMussel via the back-end whenever possible, but front-end access is provided for when it isn't possible. Keep it disabled unless you need it. False = Enable front-end access; True = Disable front-end access [Default].

"max_login_attempts"
- Maximum number of login attempts (front-end). Default = 5.

"frontend_log"
- *v1: "FrontEndLog"*
- File for logging front-end login attempts. Specify a filename, or leave blank to disable.

"disable_webfonts"
- Disable webfonts? True = Yes [Default]; False = No.

"maintenance_mode"
- Enable maintenance mode? True = Yes; False = No [Default]. Disables everything other than the front-end. Sometimes useful for when updating your CMS, frameworks, etc.

"default_algo"
- Defines which algorithm to use for all future passwords and sessions. Options: PASSWORD_DEFAULT (default), PASSWORD_BCRYPT, PASSWORD_ARGON2I (requires PHP >= 7.2.0).

"statistics"
- Track phpMussel usage statistics? True = Yes; False = No [Default].

"signatures" (Category)
Signatures configuration.

"Active"
- A list of the active signature files, delimited by commas.

Note: - Signature files must firstly be installed, before you can activate them. - For the test files to work correctly, the signature files must be installed and activated. - The value of this directive is cached. After changing it, for changes to take effect, you may need to delete the cache.

"fail_silently"
- Should phpMussel report when signatures files are missing or corrupted? If `fail_silently` is disabled, missing and corrupted files will be reported on scanning, and if `fail_silently` is enabled, missing and corrupted files will be ignored, with scanning reporting for those files that there aren't any problems. This should generally be left alone unless you're experiencing crashes or similar problems. False = Disabled; True = Enabled [Default].

"fail_extensions_silently"
- Should phpMussel report when extensions are missing? If `fail_extensions_silently` is disabled, missing extensions will be reported on scanning, and if `fail_extensions_silently` is enabled, missing extensions will be ignored, with scanning reporting for those files that there aren't any problems. Disabling this directive may potentially increase your security, but may also lead to an increase of false positives. False = Disabled; True = Enabled [Default].

"detect_adware"
- Should phpMussel parse signatures for detecting adware? False = No; True = Yes [Default].

"detect_joke_hoax"
- Should phpMussel parse signatures for detecting joke/hoax malware/viruses? False = No; True = Yes [Default].

"detect_pua_pup"
- Should phpMussel parse signatures for detecting PUAs/PUPs? False = No; True = Yes [Default].

"detect_packer_packed"
- Should phpMussel parse signatures for detecting packers and packed data? False = No; True = Yes [Default].

"detect_shell"
- Should phpMussel parse signatures for detecting shell scripts? False = No; True = Yes [Default].

"detect_deface"
- Should phpMussel parse signatures for detecting defacements and defacers? False = No; True = Yes [Default].

"detect_encryption"
- Should phpMussel detect and block encrypted files? False = No; True = Yes [Default].

"files" (Category)
File handling configuration.

"max_uploads"
- Maximum number of files that phpMussel is allowed to scan when scanning uploads before aborting the scan and informing the user they are uploading too much at once! Provides protection against a theoretical attack whereby an attacker attempts to DDoS your system or CMS by overloading phpMussel to slow down the PHP process to a grinding halt. Recommended: 10. You may wish to raise or lower this number depending on the speed of your hardware. Note that this number doesn't account for or include the contents of archives.

"filesize_limit"
- Filesize limit in KB. 65536 = 64MB [Default]; 0 = No limit (always greylisted). Any (positive) numeric value accepted. This can be useful when your PHP configuration limits the amount of memory a process can hold or if your PHP configuration limits filesize of uploads.

"filesize_response"
- What to do with files that exceed the filesize limit (if one exists). False = Whitelist; True = Blacklist [Default].

"filetype_whitelist", "filetype_blacklist", "filetype_greylist"
- If your system only allows specific types of files to be uploaded, or if your system explicitly denies certain types of files, specifying those filetypes in whitelists, blacklists and greylists can increase the speed at which scanning is performed by allowing the script to skip over certain filetypes. Format is CSV (comma separated values). If you want to scan everything, rather than whitelist, blacklist or greylist, leave the variable(/s) blank; Doing so will disable whitelist/blacklist/greylist.
- Logical order of processing is:
- If the filetype is whitelisted, don't scan and don't block the file, and don't check the file against the blacklist or the greylist.
- If the filetype is blacklisted, don't scan the file but block it anyway, and don't check the file against the greylist.
- If the greylist is empty or if the greylist is not empty and the filetype is greylisted, scan the file as per normal and determine whether to block it based on the results of the scan, but if the greylist is not empty and the filetype is not greylisted, treat the file as blacklisted, therefore not scanning it but blocking it anyway.

"check_archives"
- Attempt to check the contents of archives? False = Don't check; True = Check [Default].

If anyone is able and willing to help implement support for reading other archive formats, such help would be welcomed.

Format	Can read	Can read recursively	Can detect encryption	Notes
Zip	True	True	True	Requires libzip (normally bundled with PHP any
Tar	True	True	False	No special requirements. Format doesn't support
Rar	True	True	True	Requires the rar extension (when this extension i
7zip	False	False	False	Still currently investigating how to read 7zip file
Phar	False	False	False	Support for reading phar files was removed in v1

"filesize_archives"
- Carry over filesize blacklisting/whitelisting to the contents of archives? False = No (just greylist everything); True = Yes [Default].

"filetype_archives"
- Carry over filetype blacklisting/whitelisting to the contents of archives? False = No (just greylist everything) [Default]; True = Yes.

"max_recursion"
- Maximum recursion depth limit for archives. Default = 3.

"block_encrypted_archives"
- Detect and block encrypted archives? Because phpMussel isn't able to scan the contents of encrypted archives, it's possible that archive encryption may be employed by an attacker as a means of attempting to bypass phpMussel, anti-virus scanners and other such protections. Instructing phpMussel to block any archives that it discovers to be encrypted could potentially help reduce any risk associated with these such possibilities. False = No; True = Yes [Default].

"max_files_in_archives"
- Maximum number of files to scan from within archives before aborting the scan. Default = 0 (no maximum).

"attack_specific" (Category)

Attack-specific directives.

Chameleon attack detection: False = Off; True = On.

"chameleon_from_php"
- Search for PHP header in files that are neither PHP files nor recognised archives.

"can_contain_php_file_extensions"
- A list of file extensions allowed to contain PHP code, separated by commas. If PHP chameleon attack detection is enabled, files that contain PHP code, which have extensions that aren't on this list, will be detected as PHP chameleon attacks.

"chameleon_from_exe"
- Search for executable headers in files that are neither executables nor recognised archives and for executables whose headers are incorrect.

"chameleon_to_archive"

- Detect incorrect headers in archives and compressed files. Supported: BZ/BZIP2, GZ/GZIP, LZF, RAR, ZIP.

"chameleon_to_doc"

- Search for office documents whose headers are incorrect (Supported: DOC, DOT, PPS, PPT, XLA, XLS, WIZ).

"chameleon_to_img"

- Search for images whose headers are incorrect (Supported: BMP, DIB, PNG, GIF, JPEG, JPG, XCF, PSD, PDD, WEBP).

"chameleon_to_pdf"

- Search for PDF files whose headers are incorrect.

"archive_file_extensions"

- Recognised archive file extensions (format is CSV; should only add or remove when problems occur; unnecessarily removing may cause false positives to appear for archive files, whereas unnecessarily adding will essentially whitelist what you're adding from attack specific detection; modify with caution; also note that this has no effect on what archives can and can't be analysed at content-level). The list, as is at default, lists those formats used most commonly across the majority of systems and CMS, but intentionally isn't necessarily comprehensive.

"block_control_characters"

- Block any files containing any control characters (other than newlines)? (`[\x00-\x08\x0b\x0c\x0e\x1f\x7f]`) If you're ***ONLY*** uploading plain-text, then you can turn this option on to provide some additional protection to your system. However, if you upload anything other than plain-text, turning this on may result in false positives. False = Don't block [Default]; True = Block.

"corrupted_exe"

- Corrupted files and parse errors. False = Ignore; True = Block [Default]. Detect and block potentially corrupted PE (Portable Executable) files? Often (but not always), when certain aspects of a PE file are corrupted or can't be parsed correctly, it can be indicative of a viral infection. The processes used by most anti-virus programs to detect viruses in PE files require parsing those files in certain ways, which, if the programmer of a virus is aware of, will specifically try to prevent, in order to allow their virus to remain undetected.

"decode_threshold"

- Threshold for the length of raw data within which decode commands should be detected (in case there are any noticeable performance issues while scanning). Default = 512KB. Zero or null disables the threshold (removing any such limitation based on filesize).

"scannable_threshold"

- Threshold to the length of raw data that phpMussel is permitted to read and scan (in case there are any noticeable performance issues while scanning). Default = 32MB. Zero or null value disables the threshold. Generally, this value shouldn't be less than the average filesize of file uploads that you want and expect to receive to your server or website, shouldn't be more than the filesize_limit directive, and shouldn't be more than roughly one fifth of the total allowable memory allocation granted to PHP via the `php.ini` configuration file. This directive exists to try to prevent phpMussel from using up too much memory (that'd prevent it from being able to successfully scan files above a certain filesize).

"allow_leading_trailing_dots"

- Allow leading and trailing dots in filenames? This can sometimes be used to hide files, or to trick some systems into allowing directory traversal. False = Don't allow [Default]. True = Allow.

"block_macros"

- Try to block any files containing macros? Some types of documents and spreadsheets may contain executable macros, thus providing a dangerous potential malware vector. False = Don't block [Default]; True = Block.

"compatibility" (Category)

Compatibility directives for phpMussel.

"ignore_upload_errors"

- This directive should generally be disabled unless it's required for correct functionality of phpMussel on your specific system. Normally, when disabled, when phpMussel detects the presence of elements in the `$_FILES` array(), it'll attempt to initiate a scan of the files that those elements represent, and, if those elements are blank or empty, phpMussel will return an error message. This is proper behaviour for phpMussel. However, for some CMS, empty elements in `$_FILES` can occur as a result of the natural behaviour of those CMS, or errors may be reported when there aren't any, in which case, the normal behaviour for phpMussel will be interfering with the normal behaviour of those CMS. If such a situation occurs for you, enabling this option will instruct phpMussel to not attempt to initiate scans for such empty elements, ignore them when found and to not return any related error messages, thus allowing continuation of the page request. False = OFF; True = ON.

"only_allow_images"

- If you only expect or only intend to allow images to be uploaded to your system or CMS, and if you absolutely don't require any files other than images to be uploaded to your system or CMS, this directive should be enabled, but should otherwise be disabled. If this directive is enabled, it'll instruct phpMussel to indiscriminately block any uploads identified as non-image files, without scanning them. This may reduce processing time and memory usage for attempted uploads of non-image files. False = OFF; True = ON.

"heuristic" (Category)

Heuristic directives.

"threshold"

- Some phpMussel signatures are intended to identify suspicious and potentially malicious indicators in files, without in themselves identifying those files as being malicious directly. This "threshold" value tells phpMussel the maximum weight allowed before flagging those files as malicious. The definition of weight in this context is the total number of suspicious and potentially malicious indicators identified. By default, this value will be set to 3. A lower value generally will result in a higher occurrence of false positives but a higher number of malicious files being flagged, whereas a higher value generally will result in a lower occurrence of false positives but a lower number of malicious files being flagged. It's generally best to leave this value at its default unless you're experiencing problems related to it.

"virustotal" (Category)

VirusTotal.com directives.

"vt_public_api_key"

- Optionally, phpMussel is able to scan files using the Virus Total API as a way to provide a greatly enhanced level of protection against viruses, trojans, malware and other threats. By default, scanning files using the Virus Total API is disabled. To enable it, an API key from Virus Total is required. Due to the significant benefit that this could provide to you, it's something that I highly recommend enabling. Please be aware, however, that to use the Virus Total API, you ***MUST*** agree to their Terms of Service and you ***MUST*** adhere to all guidelines as per described by the Virus Total documentation! You are NOT permitted to use this integration feature UNLESS:
- You have read and agree to the Terms of Service of Virus Total and its API. The Terms of Service of Virus Total and its API can be found Here.
- You have read and you understand, at a minimum, the preamble of the Virus Total Public API documentation (everything after "VirusTotal Public API v2.0" but before "Contents"). The Virus Total Public API documentation can be found Here.

Note: If scanning files using the Virus Total API is disabled, you won't need to review any of the directives in this category (`virustotal`), because none of them will do anything if this is disabled. To acquire a Virus Total API key, from anywhere on their website, click the "Join our Community" link located towards the top-right of the page, enter in the information requested, and click "Sign up" when done. Follow all instructions supplied, and when you've got your public API key, copy/paste that public API key to the `vt_public_api_key` directive of the `config.ini` configuration file.

"vt_suspicion_level"

- By default, phpMussel will restrict which files it scans using the Virus Total API to those files that it considers "suspicious". You can optionally adjust this restriction by changing the value of the `vt_suspicion_level` directive.
- `0`: Files are only considered suspicious if, upon being scanned by phpMussel using its own signatures, they are deemed to carry a heuristic weight. This would effectively mean that use of the Virus Total API would be for a second opinion for when phpMussel suspects that a file may potentially be malicious, but can't entirely rule out that it may also potentially be benign (non-malicious) and therefore would otherwise normally not block it or flag it as being malicious.

- 1: Files are considered suspicious if, upon being scanned by phpMussel using its own signatures, they are deemed to carry a heuristic weight, if they're known to be executable (PE files, Mach-O files, ELF/Linux files, etc), or if they're known to be of a format that could potentially contain executable data (such as executable macros, DOC/DOCX files, archive files such as RARs, ZIPS and etc). This is the default and recommended suspicion level to apply, effectively meaning that use of the Virus Total API would be for a second opinion for when phpMussel doesn't initially find anything malicious or wrong with a file that it considers to be suspicious and therefore would otherwise normally not block it or flag it as being malicious.
- 2: All files are considered suspicious and should be scanned using the Virus Total API. I don't generally recommend applying this suspicion level, due to the risk of reaching your API quota much quicker than would otherwise be the case, but there are certain circumstances (such as when the webmaster or hostmaster has very little faith or trust whatsoever in any of the uploaded content of their users) where this suspicion level could be appropriate. With this suspicion level, all files not normally blocked or flagged as being malicious would be scanned using the Virus Total API. Note, however, that phpMussel will cease using the Virus Total API when your API quota has been reached (regardless of suspicion level), and that your quota will likely be reached much faster when using this suspicion level.

Note: Regardless of suspicion level, any files that are either blacklisted or whitelisted by phpMussel won't be scanned using the Virus Total API, because those such files would've already been declared as either malicious or benign by phpMussel by the time that they would've otherwise been scanned by the Virus Total API, and therefore, additional scanning wouldn't be required. The ability of phpMussel to scan files using the Virus Total API is intended to build further confidence for whether a file is malicious or benign in those circumstances where phpMussel itself isn't entirely certain as to whether a file is malicious or benign.

"vt_weighting"

- Should phpMussel apply the results of scanning using the Virus Total API as detections or as detection weighting? This directive exists, because, although scanning a file using multiple engines (as Virus Total does) should result in an increased detection rate (and therefore in a higher number of malicious files being caught), it can also result in a higher number of false positives, and therefore, in some circumstances, the results of scanning may be better utilised as a confidence score rather than as a definitive conclusion. If a value of 0 is used, the results of scanning using the Virus Total API will be applied as detections, and therefore, if any engine used by Virus Total flags the file being scanned as being malicious, phpMussel will consider the file to be malicious. If any other value is used, the results of scanning using the Virus Total API will be applied as detection weighting, and therefore, the number of engines used by Virus Total that flag the file being scanned as being malicious will serve as a confidence score (or detection weighting) for whether or not the file being scanned should be considered malicious by phpMussel (the value used will represent the minimum confidence score or weight required in order to be considered malicious). A value of 0 is used by default.

"vt_quota_rate" and "vt_quota_time"

- According to the Virus Total API documentation, "it is limited to at most 4 requests of any nature in any given 1 minute time frame. If you run a honeyclient, honeypot or any other automation that is going to provide resources to VirusTotal and not only retrieve reports you

are entitled to a higher request rate quota". By default, phpMussel will strictly adhere to these limitations, but due to the possibility of these rate quotas being increased, these two directives are provided as a means for you to instruct phpMussel as to what limit it should adhere to. Unless you've been instructed to do so, it's not recommended for you to increase these values, but, if you've encountered problems relating to reaching your rate quota, decreasing these values ***MAY*** sometimes help you in dealing with these problems. Your rate limit is determined as `vt_quota_rate` requests of any nature in any given `vt_quota_time` minute time frame.

"urlscanner" (Category)

A URL scanner in included with phpMussel, capable of detecting malicious URLs from within any data or files scanned.

Note: If the URL scanner is disabled, you won't need to review any of the directives in this category (`urlscanner`), because none of them will do anything if this is disabled.

URL scanner API lookup configuration.

"lookup_hphosts"

- Enables API lookups to the hpHosts API when set to true. hpHosts doesn't require an API key for performing API lookups.

"google_api_key"

- Enables API lookups to the Google Safe Browsing API when the necessary API key is defined. Google Safe Browsing API lookups requires an API key, which can be obtained from Here.
- Note: The cURL extension is required in order to use this feature.

"maximum_api_lookups"

- Maximum allowable number of API lookups to perform per individual scan iteration. Because each additional API lookup will add to the total time required to complete each scan iteration, you may wish to stipulate a limitation in order to expedite the overall scan process. When set to 0, no such maximum allowable number will be applied. Set to 10 by default.

"maximum_api_lookups_response"

- What to do if the maximum allowable number of API lookups is exceeded? False = Do nothing (continue processing) [Default]; True = Flag/block the file.

"cache_time"

- How long (in seconds) should the results of API lookups be cached for? Default is 3600 seconds (1 hour).

"legal" (Category)

Configuration relating to legal requirements.

For more information about legal requirements and how this could affect your configuration requirements, please refer to the "LEGAL INFORMATION" section of the documentation.

"pseudonymise_ip_addresses"
- Pseudonymise IP addresses when logging? True = Yes [Default]; False = No.

"privacy_policy"
- The address of a relevant privacy policy to be displayed in the footer of any generated pages. Specify a URL, or leave blank to disable.

"template_data" (Category)

Directives/Variables for templates and themes.

Template data relates to the HTML output used to generate the "Upload Denied" message displayed to users upon a file upload being blocked. If you're using custom themes for phpMussel, HTML output is sourced from the `template_custom.html` file, and otherwise, HTML output is sourced from the `template.html` file. Variables written to this section of the configuration file are parsed to the HTML output by way of replacing any variable names circumfixed by curly brackets found within the HTML output with the corresponding variable data. For example, where `foo="bar"`, any instance of `<p>{foo}</p>` found within the HTML output will become `<p>bar</p>`.

"theme"
- Default theme to use for phpMussel.

"Magnification"
- Font magnification. Default = 1.

"css_url"
- The template file for custom themes utilises external CSS properties, whereas the template file for the default theme utilises internal CSS properties. To instruct phpMussel to use the template file for custom themes, specify the public HTTP address of your custom theme's CSS files using the `css_url` variable. If you leave this variable blank, phpMussel will use the template file for the default theme.

"PHPMailer" (Category)

PHPMailer configuration.

Currently, phpMussel uses PHPMailer only for front-end two-factor authentication. If you don't use the front-end, or if you don't use two-factor authentication for the front-end, you can ignore these directives.

"event_log"
- *v1: "EventLog"*
- A file for logging all events in relation to PHPMailer. Specify a filename, or leave blank to disable.

"skip_auth_process"
- *v1: "SkipAuthProcess"*

- Setting this directive to `true` instructs PHPMailer to skip the normal authentication process that normally occurs when sending email via SMTP. This should be avoided, because skipping this process may expose outbound email to MITM attacks, but may be necessary in cases where this process prevents PHPMailer from connecting to an SMTP server.

“enable_two_factor”
- *v1: “Enable2FA”*
- This directive determines whether to use 2FA for front-end accounts.

“host”
- *v1: “Host”*
- The SMTP host to use for outbound email.

“port”
- *v1: “Port”*
- The port number to use for outbound email. Default = 587.

“smtp_secure”
- *v1: “SMTPSecure”*
- The protocol to use when sending email via SMTP (TLS or SSL).

“smtp_auth”
- *v1: “SMTPAuth”*
- This directive determines whether to authenticate SMTP sessions (should usually be left alone).

“username”
- *v1: “Username”*
- The username to use when sending email via SMTP.

“password”
- *v1: “Password”*
- The password to use when sending email via SMTP.

“set_from_address”
- *v1: “setFromAddress”*
- The sender address to cite when sending email via SMTP.

“set_from_name”
- *v1: “setFromName”*
- The sender name to cite when sending email via SMTP.

“add_reply_to_address”
- *v1: “addReplyToAddress”*
- The reply address to cite when sending email via SMTP.

“add_reply_to_name”
- *v1: “addReplyToName”*
- The reply name to cite when sending email via SMTP.

“supplementary_cache_options” (Category)
Supplementary cache options.

Currently, this is extremely experimental, and mightn’t behave as expected! For the moment, I recommend ignoring it.

“enable_apcu”
- Specifies whether to try using APCu for caching. Default = False.

“enable_memcached”
- Specifies whether to try using Memcached for caching. Default = False.

“enable_redis”
- Specifies whether to try using Redis for caching. Default = False.

“enable_pdo”
- Specifies whether to try using PDO for caching. Default = False.

“memcached_host”
- Memcached host value. Default = “localhost”.

“memcached_port”
- Memcached port value. Default = “11211”.

“redis_host”
- Redis host value. Default = “localhost”.

“redis_port”
- Redis port value. Default = “6379”.

“redis_timeout”
- Redis timeout value. Default = “2.5”.

“pdo_dsn”
- PDO DSN value. Default = “`mysql:dbname=phpmussel;host=localhost;port=3306`”.

“pdo_username”
- PDO username.

Type	Byte	Description
General_Command_Detections	0?	For CSV (comma separated values) signature files. Values (signatures) are
Filename	1?	For filename signatures.
Hash	2?	For hash signatures.
Standard	3?	For signature files that work directly with file content.
Standard_RegEx	4?	For signature files that work directly with file content. Signatures can cont
Normalised	5?	For signature files that work with ANSI-normalised file content.
Normalised_RegEx	6?	For signature files that work with ANSI-normalised file content. Signature
HTML	7?	For signature files that work with HTML-normalised file content.
HTML_RegEx	8?	For signature files that work with HTML-normalised file content. Signatu
PE_Extended	9?	For signature files that work with PE metadata (other than PE sectional m
PE_Sectional	A?	For signature files that work with PE sectional metadata.
Complex_Extended	B?	For signature files that work with various rules based on extended metadat
URL_Scanner	C?	For signature files that work with URLs.

"pdo_password"

- PDO password.

1.9 Signature format

See also: - What is a "signature"?

The first 9 bytes [x0-x8] of a phpMussel signature file are phpMussel, and act as a "magic number", to identify them as signature files (this helps to prevent phpMussel from accidentally attempting to use files that aren't signature files). The next byte [x9] identifies the type of signature file, which phpMussel must know in order to be able to correctly interpret the signature file. The following types of signature files are recognised:

The next byte [x10] is a newline [0A], and concludes the phpMussel signature file header.

Each non-empty line thereafter is a signature or rule. Each signature or rule occupies one line. The signature formats supported are described below.

FILENAME SIGNATURES

All filename signatures follow the format:

NAME:FNRX

Where NAME is the name to cite for that signature and FNRX is the regex pattern to match filenames (unencoded) against.

HASH SIGNATURES

All hash signatures follow the format:

HASH:FILESIZE:NAME

Where HASH is the hash (usually MD5) of an entire file, FILESIZE is the total size of that file and NAME is the name to cite for that signature.

PE SECTIONAL SIGNATURES

All PE Sectional signatures follow the format:

SIZE:HASH:NAME

Where HASH is the MD5 hash of a section of a PE file, SIZE is the total size of that section and NAME is the name to cite for that signature.

PE EXTENDED SIGNATURES

All PE extended signatures follow the format:

```
$VAR:HASH:SIZE:NAME
```

Where $VAR is the name of the PE variable to match against, HASH is the MD5 hash of that variable, SIZE is the total size of that variable and NAME is the name to cite for that signature.

COMPLEX EXTENDED SIGNATURES

Complex Extended signatures are rather different to the other types of signatures possible with phpMussel, in that what they are matching against is specified by the signatures themselves and they can match against multiple criteria. The match criteria are delimited by ";" and the match type and match data of each match criteria is delimited by ":" as so that format for these signatures tends to look a bit like:

```
$variable1:SOMEDATA;$variable2:SOMEDATA;SignatureName
```

EVERYTHING ELSE

All other signatures follow the format:

```
NAME:HEX:FROM:TO
```

Where NAME is the name to cite for that signature and HEX is a hexadecimal-encoded segment of the file intended to be matched by the given signature. FROM and TO are optional parameters, indicating from which and to which positions in the source data to check against.

REGEX (REGULAR EXPRESSIONS)

Any form of regex understood and correctly processed by PHP should also be correctly understood and processed by phpMussel and its signatures. However, I'd suggest taking extreme caution when writing new regex based signatures, because, if you're not entirely sure what you're doing, there can be highly irregular and/or unexpected results. Take a look at the phpMussel source-code if you're not entirely sure about the context in which regex statements are parsed. Also, remember that all patterns (with exception to filename, archive metadata and MD5 patterns) must be hexadecimally encoded (foregoing pattern syntax, of course)!

1.10 Known compatibility problems

PHP and PCRE

- phpMussel requires PHP and PCRE to execute and function correctly. Without PHP, or without the PCRE extension of PHP, phpMussel won't execute or function correctly. Should make sure your system has both PHP and PCRE installed and available prior to downloading and installing phpMussel.

ANTI-VIRUS SOFTWARE COMPATIBILITY

For the most part, phpMussel should be fairly compatible with most other virus scanning software. However, conflicts have been reported by a number of users in the past. This information below is from VirusTotal.com, and it describes a number of false positives reported by various anti-virus programs against phpMussel. Although this information isn't an absolute guarantee of whether or not you will encounter compatibility problems between phpMussel and your anti-virus software, if your anti-virus software is noted as flagging against phpMussel, you should either consider disabling it

Scanner	Results
Bkav	Reports “VEX.Webshell”

prior to working with phpMussel or should consider alternative options to either your anti-virus software or phpMussel.

This information was last updated 2018.10.09 and is current for all phpMussel releases of the two most recent minor versions (v1.5.0-v1.6.0) at the time of writing this.

This information only applies to the main package. Results may vary based on installed signature files, plugins, and other peripheral components.

1.11 Frequently asked questions (FAQ)

- What is a “signature”?
- What is a “false positive”?
- How frequently are signatures updated?
- I’ve encountered a problem while using phpMussel and I don’t know what to do about it! Please help!
- I want to use phpMussel (prior to v2) with a PHP version older than 5.4.0; Can you help?
- I want to use phpMussel (v2) with a PHP version older than 7.2.0; Can you help?
- Can I use a single phpMussel installation to protect multiple domains?
- I don’t want to mess around with installing this and getting it to work with my website; Can I just pay you to do it all for me?
- Can I hire you or any of the developers of this project for private work?
- I need specialist modifications, customisations, etc; Can you help?
- I’m a developer, website designer, or programmer. Can I accept or offer work relating to this project?
- I want to contribute to the project; Can I do this?
- How to access specific details about files when they are scanned?
- Can I use cron to update automatically?
- Can phpMussel scan files with non-ANSI names?
- Blacklists – Whitelists – Greylists – What are they, and how do I use them?
- When I activate or deactivate signature files via the updates page, it sorts them alphanumerically in the configuration. Can I change the way that they get sorted?

What is a “signature”?

In the context of phpMussel, a “signature” refers to data that acts as an indicator/identifier for something specific that we’re looking for, usually in the form of some very small, distinct, innocuous segment of something larger and otherwise harmful, like a virus or trojan, or in the form of a file checksum, hash, or other similarly identifying indicator, and usually includes a label, and some other data to help provide additional context that can be used by phpMussel to determine the best way to proceed when it encounters what we’re looking for.

What is a “false positive”?

The term “false positive” (*alternatively: “false positive error”; “false alarm”*), described very simply, and in a generalised context, is used when testing for a condition, to refer to the results of

	phpMussel should *NOT* block a file	phpMussel *SHOULD* block a file
phpMussel does *NOT* block a file	True negative (correct inference)	Missed detection (analogous to false negati
phpMussel *DOES* block a file	**False positive**	True positive (correct inference)

that test, when the results are positive (i.e., the condition is determined to be "positive", or "true"), but are expected to be (or should have been) negative (i.e., the condition, in reality, is "negative", or "false"). A "false positive" could be considered analogous to "crying wolf" (wherein the condition being tested is whether there's a wolf near the herd, the condition is "false" in that there's no wolf near the herd, and the condition is reported as "positive" by the shepherd by way of calling "wolf, wolf"), or analogous to situations in medical testing wherein a patient is diagnosed as having some illness or disease, when in reality, they have no such illness or disease.

Related outcomes when testing for a condition can be described using the terms "true positive", "true negative" and "false negative". A "true positive" refers to when the results of the test and the actual state of the condition are both true (or "positive"), and a "true negative" refers to when the results of the test and the actual state of the condition are both false (or "negative"); A "true positive" or a "true negative" is considered to be a "correct inference". The antithesis of a "false positive" is a "false negative"; A "false negative" refers to when the results of the test are negative (i.e., the condition is determined to be "negative", or "false"), but are expected to be (or should have been) positive (i.e., the condition, in reality, is "positive", or "true").

In the context of phpMussel, these terms refer to the signatures of phpMussel and the files that they block. When phpMussel blocks a file due to bad, outdated or incorrect signatures, but shouldn't have done so, or when it does so for the wrong reasons, we refer to this event as a "false positive". When phpMussel fails to block a file that should have been blocked, due to unforeseen threats, missing signatures or shortfalls in its signatures, we refer to this event as a "missed detection" (which is analogous to a "false negative").

This can be summarised by the table below:

How frequently are signatures updated?

Update frequency varies depending on the signature files in question. All maintainers for phpMussel signature files generally try to keep their signatures as up-to-date as is possible, but as all of us have various other commitments, our lives outside the project, and as none of us are financially compensated (i.e., paid) for our efforts on the project, a precise update schedule can't be guaranteed. Generally, signatures are updated whenever there's enough time to update them. Assistance is always appreciated if you're willing to offer any.

I've encountered a problem while using phpMussel and I don't know what to do about it! Please help!

- Are you using the latest version of the software? Are you using the latest versions of your signature files? If the answer to either of these two questions is no, try to update everything first, and check whether the problem persists. If it persists, continue reading.
- Have you checked through all the documentation? If not, please do so. If the problem can't be solved using the documentation, continue reading.
- Have you checked the **issues page**, to see whether the problem has been mentioned before? If it's been mentioned before, check whether any suggestions, ideas, and/or solutions were provided, and follow as per necessary to try to resolve the problem.
- If the problem still persists, please seek help about it by creating a new issue on the issues

page.

I want to use phpMussel (prior to v2) with a PHP version older than 5.4.0; Can you help?

No. PHP >= 5.4.0 is a minimum requirement for phpMussel < v2.

I want to use phpMussel (v2) with a PHP version older than 7.2.0; Can you help?

No. PHP >= 7.2.0 is a minimum requirement for phpMussel v2.

See also: Compatibility Charts.

Can I use a single phpMussel installation to protect multiple domains?

Yes. phpMussel installations are not naturally locked to specific domains, and can therefore be used to protect multiple domains. Generally, we refer to phpMussel installations protecting only one domain as "single-domain installations", and we refer to phpMussel installations protecting multiple domains and/or sub-domains as "multi-domain installations". If you operate a multi-domain installation and need to use different sets of signature files for different domains, or need phpMussel to be configured differently for different domains, it's possible to do this. After loading the configuration file (`config.ini`), phpMussel will check for the existence of a "configuration overrides file" specific to the domain (or sub-domain) being requested (`the-domain-being-requested.tld.config.ini`), and if found, any configuration values defined by the configuration overrides file will be used for the execution instance instead of the configuration values defined by the configuration file. Configuration overrides files are identical to the configuration file, and at your discretion, may contain either the entirety of all configuration directives available to phpMussel, or whichever small subsection required which differs from the values normally defined by the configuration file. Configuration overrides files are named according to the domain that they are intended for (so, for example, if you need a configuration overrides file for the domain, `http://www.some-domain.tld/`, its configuration overrides file should be named as `some-domain.tld.config.ini`, and should be placed within the vault alongside the configuration file, `config.ini`). The domain name for the execution instance is derived from the `HTTP_HOST` header of the request; "www" is ignored.

I don't want to mess around with installing this and getting it to work with my website; Can I just pay you to do it all for me?

Maybe. This is considered on a case-by-case basis. Let us know what you need, what you're offering, and we'll let you know whether we can help.

Can I hire you or any of the developers of this project for private work?

See above.

I need specialist modifications, customisations, etc; Can you help?

See above.

I'm a developer, website designer, or programmer. Can I accept or offer work relating to this project?

Yes. Our license does not prohibit this.

I want to contribute to the project; Can I do this?

Yes. Contributions to the project are very welcome. Please see "CONTRIBUTING.md" for more information.

How to access specific details about files when they are scanned?

You can access specific details about files when they are scanned by assigning an array to use for this purpose prior to instructing phpMussel to scan them.

In the example below, `$Foo` is assigned for this purpose. After scanning `/file/path/...`, detailed information about the files contained by `/file/path/...` will be contained by `$Foo`.

```
<?php
require 'phpmussel/loader.php';
$phpMussel['Set-Scan-Debug-Array']($Foo);
$Results = $phpMussel['Scan']('/file/path/...');
var_dump($Foo);
```

The array is a multidimensional array consisting of elements representing each file being scanned and sub-elements representing the details about these files. These sub-elements are as follows:

- Filename (`string`)
- FromCache (`bool`)
- Depth (`int`)
- Size (`int`)
- MD5 (`string`)
- SHA1 (`string`)
- SHA256 (`string`)
- CRC32B (`string`)
- 2CC (`string`)
- 4CC (`string`)
- ScanPhase (`string`)
- Container (`string`)
- FileSwitch (`string`)
- Is_ELF (`bool`)
- Is_Graphics (`bool`)
- Is_HTML (`bool`)
- Is_Email (`bool`)
- Is_MachO (`bool`)
- Is_PDF (`bool`)
- Is_SWF (`bool`)
- Is_PE (`bool`)
- Is_Not_HTML (`bool`)
- Is_Not_PHP (`bool`)
- NumOfSections (`int`)
- PEFileDescription (`string`)
- PEFileVersion (`string`)
- PEProductName (`string`)
- PEProductVersion (`string`)
- PECopyright (`string`)
- PEOriginalFilename (`string`)

- PECompanyName (`string`)
- Results (`int`)
- Output (`string`)

- Not provided with cached results (only provided for new scan results).
- Only provided when scanning PE files.

Optionally, this array can be destroyed by using the following:

```
$phpMussel['Destroy-Scan-Debug-Array']($Foo);
```

Can I use cron to update automatically?

Yes. An API is built into the front-end for interacting with the updates page via external scripts. A separate script, "Cronable", is available, and can be used by your cron manager or cron scheduler to update this and other supported packages automatically (this script provides its own documentation).

Can phpMussel scan files with non-ANSI names?

Let's say there's a directory you want to scan. In this directory, you have some files with non-ANSI names. - `.txt` - `.txt` - `.txt`

Let's assume that you're either using CLI mode or the phpMussel API to scan.

When using PHP < 7.1.0, on some systems, phpMussel won't see these files when attempting to scan the directory, and so, won't be able to scan these files. You'll likely see the same results as if you were to scan an empty directory:

```
Sun, 01 Apr 2018 22:27:41 +0800 Started.
Sun, 01 Apr 2018 22:27:41 +0800 Finished.
```

Additionally, when using PHP < 7.1.0, scanning the files individually produces results like these:

```
Sun, 01 Apr 2018 22:27:41 +0800 Started.
> Checking 'X:/directory/.txt' (FN: b831eb8f):
-> Invalid file!
Sun, 01 Apr 2018 22:27:41 +0800 Finished.
```

Or these:

```
Sun, 01 Apr 2018 22:27:41 +0800 Started.
> X:/directory/??????.txt is not a file or directory.
Sun, 01 Apr 2018 22:27:41 +0800 Finished.
```

This is because of the way that PHP handled non-ANSI filenames prior to PHP 7.1.0. If you experience this problem, the solution is to update your PHP installation to 7.1.0 or newer. In PHP >= 7.1.0, non-ANSI filenames are handled better, and phpMussel should be able to scan the files properly.

For comparison, the results when attempting to scan the directory using PHP >= 7.1.0:

```
Sun, 01 Apr 2018 22:27:41 +0800 Started.
-> Checking '\.txt' (FN: b2ce2d31; FD: 27cbe813):
--> No problems found.
-> Checking '\.txt' (FN: 50debed5; FD: 27cbe813):
--> No problems found.
-> Checking '\.txt' (FN: ee20a2ae; FD: 27cbe813):
--> No problems found.
Sun, 01 Apr 2018 22:27:41 +0800 Finished.
```

And attempting to scan the files individually:

```
Sun, 01 Apr 2018 22:27:41 +0800 Started.
> Checking 'X:/directory/.txt' (FN: b831eb8f; FD: 27cbe813):
-> No problems found.
Sun, 01 Apr 2018 22:27:41 +0800 Finished.
```

Blacklists – Whitelists – Greylists – What are they, and how do I use them?

The terms convey different meanings in different contexts. In phpMussel, there are three contexts where these terms are used: Filesize response, filetype response, and the signature greylist.

In order to achieve a desired outcome at minimal cost to processing, there are some simple things that phpMussel can check prior to actually scanning files, such as a file's size, name, and extension. For example; If a file is too large, or if its extension indicates a type of file that we don't want to allow onto our websites anyway, we can flag the file immediately, and don't need to scan it.

Filesize response is the way that phpMussel responds when a file exceeds a specified limit. Though no actual lists are involved, a file may be considered effectively blacklisted, whitelisted, or greylisted, based on its size. Two separate, optional configuration directives exist to specify a limit and desired response respectively.

Filetype response is the way that phpMussel responds to file's extension. Three separate, optional configuration directives exist to explicitly specify which extensions should be blacklisted, whitelisted, or greylisted. A file may be considered effectively blacklisted, whitelisted, or greylisted if its extension matches any of the specified extensions respectively.

In these two contexts, being whitelisted means that it shouldn't be scanned or flagged; being blacklisted means that it should be flagged (and therefore don't need to scan it); and being greylisted means further analysis is required to determine whether we should flag it (i.e., it should be scanned).

The signature greylist is a list of signatures that should essentially be ignored (this is briefly mentioned earlier in the documentation). When a signature on the signature greylist is triggered, phpMussel continues working through its signatures and takes no particular action in regards to the greylisted signature. There's no signature blacklist, because the implied behaviour is normal behaviour for triggered signatures anyway, and there's no signature whitelist, because the implied behaviour wouldn't really make sense in consideration of how phpMussel normal works and the capabilities it already has.

The signature greylist is useful if you need to resolve problems caused by a particular signature without disabling or uninstalling the entire signature file.

When I activate or deactivate signature files via the updates page, it sorts them alphanumerically in the configuration. Can I change the way that they get sorted?

Yes. If you need to force some files to execute in a specific order, you can add some arbitrary data before their names in the configuration directive where they're listed, separated by a colon. When

the updates page subsequently sorts the files again, this added arbitrary data will affect the sort order, causing them consequently to execute in the order that you want, without needing to rename any of them.

For example, assuming a configuration directive with files listed as follows:

`file1.php,file2.php,file3.php,file4.php,file5.php`

If you wanted `file3.php` to execute first, you could add something like `aaa:` before the name of the file:

`file1.php,file2.php,aaa:file3.php,file4.php,file5.php`

Then, if a new file, `file6.php`, is activated, when the updates page resorts them all, it should end up like this:

`aaa:file3.php,file1.php,file2.php,file4.php,file5.php,file6.php`

Same situation when a file is deactivated. Conversely, if you wanted the file to execute last, you could add something like `zzz:` before the name of the file. In any case, you won't need to rename the file in question.

1.12 Legal information

1.12.1 Section preamble

This section of the documentation is intended to describe possible legal considerations regarding the use and implementation of the package, and to provide some basic related information. This may be important for some users as a means to ensure compliancy with any legal requirements that may exist in the countries that they operate in, and some users may need to adjust their website policies in accordance with this information.

First and foremost, please realise that I (the package author) am not a lawyer, nor a qualified legal professional of any kind. Therefore, I am not legally qualified to provide legal advice. Also, in some cases, exact legal requirements may vary between different countries and jurisdictions, and these varying legal requirements may sometimes conflict (such as, for example, in the case of countries that favour privacy rights and the right to be forgotten, versus countries that favour extended data retention). Consider also that access to the package is not restricted to specific countries or jurisdictions, and therefore, the package userbase is likely to the geographically diverse. These points considered, I'm not in a position to state what it means to be "legally compliant" for all users, in all regards. However, I hope that the information herein will help you to come to a decision yourself regarding what you must do in order to remain legally compliant in the context of the package. If you have any doubts or concerns regarding the information herein, or if you need additional help and advice from a legal perspective, I would recommend consulting a qualified legal professional.

1.12.2 Liability and responsibility

As per already stated by the package license, the package is provided without any warranty. This includes (but is not limited to) all scope of liability. The package is provided to you for your convenience, in the hope that it will be useful, and that it will provide some benefit for you. However, whether you use or implement the package, is your own choice. You are not forced to use or implement the package, but when you do so, you are responsible for that decision. Neither I, nor any other contributors to the package, are legally responsible for the consequences of the decisions that you make, regardless of whether direct, indirect, implied, or otherwise.

1.12.3 Third parties

Depending on its exact configuration and implementation, the package may communicate and share information with third parties in some cases. This information may be defined as "personally identifiable information" (PII) in some contexts, by some jurisdictions.

How this information may be used by these third parties, is subject to the various policies set forth by these third parties, and is outside the scope of this documentation. However, in all such cases, sharing of information with these third parties can be disabled. In all such cases, if you choose to enable it, it is your responsibility to research any concerns that you may have regarding the privacy, security, and usage of PII by these third parties. If any doubts exist, or if you're unsatisfied with the conduct of these third parties in regards to PII, it may be best to disable all sharing of information with these third parties.

For the purpose of transparency, the type of information shared, and with whom, is described below.

11.2.0 WEBFONTS Some custom themes, as well as the the standard UI ("user interface") for the phpMussel front-end and the "Upload Denied" page, may use webfonts for aesthetic reasons. Webfonts are disabled by default, but when enabled, direct communication between the user's browser and the service hosting the webfonts occurs. This may potentially involve communicating information such as the user's IP address, user agent, operating system, and other details available to the request. Most of these webfonts are hosted by the Google Fonts service.

Relevant configuration directives: - `general` -> `disable_webfonts`

11.2.1 URL SCANNER URLs found within file uploads may be shared with the hpHosts API or the Google Safe Browsing API, depending on how the package is configured. In the case of the hpHosts API, this behaviour is enabled by default. The Google Safe Browsing API requires API keys in order to work correctly, and is therefore disabled by default.

Relevant configuration directives: - `urlscanner` -> `lookup_hphosts` - `urlscanner` -> `google_api_key`

11.2.2 VIRUS TOTAL When phpMussel scans a file upload, the hashes of those files may be shared with the Virus Total API, depending on how the package is configured. There are plans to be able to share entire files at some point in the future too, but this feature isn't supported by the package at this time. The Virus Total API requires an API key in order to work correctly, and is therefore disabled by default.

Information (including files and related file metadata) shared with Virus Total, may also be shared with their partners, affiliates, and various others for research purposes. This is described in more detail by their privacy policy.

See: Privacy Policy – VirusTotal.

Relevant configuration directives: - `virustotal` -> `vt_public_api_key`

1.12.4 Logging

Logging is an important part of phpMussel for a number of reasons. Without logging, it may be difficult to diagnose false positives, to ascertain exactly how performant phpMussel is in any particular context, and to determine where its shortfalls may be, and what changes may be required to its configuration or signatures accordingly, in order for it to continue functioning as intended.

Regardless, logging mightn't be desirable for all users, and remains entirely optional. In phpMussel, logging is disabled by default. To enable it, phpMussel must be configured accordingly.

Additionally, whether logging is legally permissible, and to the extent that it is legally permissible (e.g., the types of information that may be logged, for how long, and under what circumstances), may vary, depending on jurisdiction and on the context where phpMussel is implemented (e.g., whether you're operating as an individual, as a corporate entity, and whether on a commercial or non-commercial basis). It may therefore be useful for you to read through this section carefully.

There are multiple types of logging that phpMussel can perform. Different types of logging involves different types of information, for different reasons.

11.3.0 SCAN LOGS When enabled in the package configuration, phpMussel keeps logs of the files it scans. This type of logging is available in two different formats: - Human readable logfiles. - Serialised logfiles.

Entries to a human readable logfile typically look something like this (as an example):

```
Mon, 21 May 2018 00:47:58 +0800 Started.
> Checking 'ascii_standard_testfile.txt' (FN: ce76ae7a; FD: 7b9bfed5):
-> Detected phpMussel-Testfile.ASCII.Standard!
Mon, 21 May 2018 00:48:04 +0800 Finished.
```

A scan log entry typically includes the following information: - The date and time that the file was scanned. - The name of the file scanned. - CRC32b hashes of the name and contents of the file. - What was detected in the file (if anything was detected).

Relevant configuration directives: - `general -> scan_log` - `general -> scan_log_serialized`

When these directives are left empty, this type of logging will remain disabled.

11.3.1 SCAN KILLS When enabled in the package configuration, phpMussel keeps logs of the uploads that have been blocked.

Entries to a "scan kills" logfile typically look something like this (as an example):

```
Date: Mon, 21 May 2018 00:47:56 +0800
IP address: 127.0.0.1
== Scan results (why flagged) ==
Detected phpMussel-Testfile.ASCII.Standard (ascii_standard_testfile.txt)!
== Hash signatures reconstruction ==
3ed8a00c6c498a96a44d56533806153c:666:ascii_standard_testfile.txt
Quarantined as "/vault/quarantine/0000000000-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.qfu".
```

A "scan kills" entry typically includes the following information: - The date and time that the upload was blocked. - The IP address where the upload originated from. - The reason why the file was blocked (what was detected). - The name of the file blocked. - An MD5 and the size of the file blocked. - Whether the file was quarantined, and under what internal name.

Relevant configuration directives: - `general -> scan_kills`

11.3.2 FRONT-END LOGGING This type of logging relates front-end login attempts, and occurs only when a user attempts to log into the front-end (assuming front-end access is enabled).

A front-end log entry contains the IP address of the user attempting to log in, the date and time that the attempt occurred, and the results of the attempt (successfully logged in, or failed to log in). A front-end log entry typically looks something like this (as an example):

```
x.x.x.x - Day, dd Mon 20xx hh:ii:ss +0000 - "admin" - Logged in.
```

Relevant configuration directives: - `general` -> `frontend_log`

11.3.3 LOG ROTATION You may want to purge logs after a period of time, or may be required to do so by law (i.e., the amount of time that it's legally permissible for you to retain logs may be limited by law). You can achieve this by including date/time markers in the names of your logfiles as per specified by your package configuration (e.g., `{yyyy}-{mm}-{dd}.log`), and then enabling log rotation (log rotation allows you to perform some action on logfiles when specified limits are exceeded).

For example: If I was legally required to delete logs after 30 days, I could specify `{dd}.log` in the names of my logfiles (`{dd}` represents days), set the value of `log_rotation_limit` to 30, and set the value of `log_rotation_action` to `Delete`.

Conversely, if you're required to retain logs for an extended period of time, you could either not use log rotation at all, or you could set the value of `log_rotation_action` to `Archive`, to compress logfiles, thereby reducing the total amount of disk space that they occupy.

Relevant configuration directives: - `general` -> `log_rotation_limit` - `general` -> `log_rotation_action`

11.3.4 LOG TRUNCATION It's also possible to truncate individual logfiles when they exceed a certain size, if this is something you might need or want to do.

Relevant configuration directives: - `general` -> `truncate`

11.3.5 IP ADDRESS PSEUDONYMISATION Firstly, if you're not familiar with the term, "pseudonymisation" refers to the processing of personal data as such that it can't be identified to any specific data subject anymore without supplementary information, and provided that such supplementary information is maintained separately and subject to technical and organisational measures to ensure that personal data can't be identified to any natural person.

The following resources can help to explain it in more detail: - [trust-hub.com] What is pseudonymisation? - [dataprotection.ie] Anonymisation and pseudonymisation - [Wikipedia] Pseudonymization

In some circumstances, you may be legally required to anonymise or pseudonymise any PII collected, processed, or stored. Although this concept has existed for quite some time now, GDPR/DSGVO notably mentions, and specifically encourages "pseudonymisation".

phpMussel is able to pseudonymise IP addresses when logging them, if this is something you might need or want to do. When phpMussel pseudonymises IP addresses, when logged, the final octet of IPv4 addresses, and everything after the second part of IPv6 addresses is represented by an "x" (effectively rounding IPv4 addresses to the initial address of the 24th subnet they factor into, and IPv6 addresses to the initial address of the 32nd subnet they factor into).

Relevant configuration directives: - `legal` -> `pseudonymise_ip_addresses`

11.3.6 STATISTICS phpMussel is optionally able to track statistics such as the total number of files scanned and blocked since some particular point in time. This feature is disabled by default, but can be enabled via the package configuration. The type of information tracked shouldn't be regarded as PII.

Relevant configuration directives: - `general -> statistics`

11.3.7 ENCRYPTION phpMussel doesn't encrypt its cache or any log information. Cache and log encryption may be introduced in the future, but there aren't any specific plans for it currently. If you're concerned about unauthorised third parties gaining access to parts of phpMussel that may contain PII or sensitive information such as its cache or logs, I would recommend that phpMussel not be installed at a publicly accessible location (e.g., install phpMussel outside the standard `public_html` directory or equivalent thereof available to most standard webservers) and that appropriately restrictive permissions be enforced for the directory where it resides (in particular, for the vault directory). If that isn't sufficient to address your concerns, then configure phpMussel as such that the types of information causing your concerns won't be collected or logged in the first place (such as, by disabling logging).

1.12.5 Cookies

When a user successfully logs into the front-end, phpMussel sets a cookie in order to be able to remember the user for subsequent requests (i.e., cookies are used to authenticate the user to a login session). On the login page, a cookie warning is displayed prominently, warning the user that a cookie will be set if they engage in the relevant action. Cookies aren't set at any other points in the codebase.

Relevant configuration directives: - `general -> disable_frontend`

1.12.6 Marketing and advertising

phpMussel doesn't collect or process any information for marketing or advertising purposes, and neither sells nor profits from any collected or logged information. phpMussel is not a commercial enterprise, nor is related to any commercial interests, so doing these things wouldn't make any sense. This has been the case since the beginning of the project, and continues to be the case today. Additionally, doing these things would be counter-productive to the spirit and intended purpose of the project as a whole, and for as long as I continue to maintain the project, will never happen.

1.12.7 Privacy policy

In some circumstances, you may be legally required to clearly display a link to your privacy policy on all pages and sections of your website. This may be important as a means to ensure that users are well-informed of your exact privacy practices, the types of PII you collect, and how you intend to use it. In order to be able to include such a link on phpMussel's "Upload Denied" page, a configuration directive is provided to specify the URL to your privacy policy.

Relevant configuration directives: - `legal -> privacy_policy`

1.12.8 GDPR/DSGVO

The General Data Protection Regulation (GDPR) is a regulation of the European Union, which comes into effect as of May 25, 2018. The primary goal of the regulation is to give control to

EU citizens and residents regarding their own personal data, and to unify regulation within the EU concerning privacy and personal data.

The regulation contains specific provisions pertaining to the processing of "personally identifiable information" (PII) of any "data subjects" (any identified or identifiable natural person) either from or within the EU. To be compliant with the regulation, "enterprises" (as per defined by the regulation), and any relevant systems and processes must implement "privacy by design" by default, must use the highest possible privacy settings, must implement necessary safeguards for any stored or processed information (including, but not limited to, the implementation of pseudonymisation or full anonymisation of data), must clearly and unambiguously declare the types of data they collect, how they process it, for what reasons, for how long they retain it, and whether they share this data with any third parties, the types of data shared with third parties, how, why, and so on.

Data may not be processed unless there's a lawful basis for doing so, as per defined by the regulation. Generally, this means that in order to process a data subject's data on a lawful basis, it must be done in compliance with legal obligations, or done only after explicit, well-informed, unambiguous consent has been obtained from the data subject.

Because aspects of the regulation may evolve in time, in order to avoid the propagation of outdated information, it may be better to learn about the regulation from an authoritative source, as opposed to simply including the relevant information here in the package documentation (which may eventually become outdated as the regulation evolves).

EUR-Lex (a part of the official website of the European Union that provides information about EU law) provides extensive information about GDPR/DSGVO, available in 24 different languages (at the time of writing this), and available for download in PDF format. I would definitely recommend reading the information that they provide, in order to learn more about GDPR/DSGVO: - REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL

Intersoft Consulting also provides extensive information about GDPR/DSGVO, available in German and English, which could be useful to learn more about GDPR/DSGVO: - General Data Protection Regulation (GDPR) – Final text neatly arranged

Alternatively, there's a brief (non-authoritative) overview of GDPR/DSGVO available at Wikipedia: - General Data Protection Regulation