Snuffleupagus Documentation

Release stable

Sebastien Blot & Julien Voisin

Apr 25, 2024

CONTENTS

Snuffleupagus is a PHP7+ and PHP8+ module designed to drastically raise the cost of attacks against websites. This is achieved by killing entire bug classes and providing a powerful virtual-patching system, allowing the administrator to fix specific vulnerabilities without having to touch the PHP code.

CHAPTER

ONE

DOCUMENTATION

1.1 Features

Snuffleupagus has a lot of features that can be divided in two main categories: bug-classes killers and virtual-patching. The first category provides primitives to kill various bug families (like arbitrary code execution via `unserialize` for example) or raise the cost of exploitation. The second category is a highly configurable system to patch functions in php itself.

1.1.1 Bug classes killed or mitigated

`system` injections

The `system` function executes an external program and displays the output. It is used to interact with various external tools, like file-format converters. Unfortunately, passing user-controlled parameters to it often leads to arbitrary command execution.

> When allowing user-supplied data to be passed to this function, use *escapeshellarg()* or *escapeshellcmd()* to ensure that users cannot trick the system into executing arbitrary commands.
>
> —The PHP documentation about system

We're mitigating it by filtering the `$`, `|`, `;`, `` ` ``, `\n` and `&` chars in our default configuration, making it a lot harder for an attacker to inject arbitrary commands. This feature is even more effective when used along with *readonly_exec*.

Examples of related vulnerabilities

- CVE-2013-3630: Authenticated remote code execution in Moodle
- CVE-2014-1610: Unauthenticated remote code execution in DokuWiki
- CVE-2014-4688: Authenticated remote code execution in pfSense
- CVE-2017-7981: Authenticated remote code execution in Tuleap
- CVE-2018-20434: Authenticated remote code execution in LibreNMS
- CVE-2020-5791: Authenticated remote code execution in Nagios XI
- CVE-2020-8813: Unauthenticated remote code execution in Cacti
- Every single modem/router/switch/IoT/....

`mail`-related injections

This vulnerability has been known since 2011 and was popularized by RIPS in 2016. The last flag of the *mail* function can be used to pass various parameters to the underlying binary used to send emails; this can lead to an arbitrary file write, often meaning an arbitrary code execution.

> The `additional_parameters` parameter can be used to pass additional flags as command line options to the program configured to be used when sending mail
>
> —The PHP documentation about mail

We're killing it by preventing any extra options in `additional_parameters`. This feature is even more effective when used along with *readonly_exec*.

Examples of related vulnerabilities

- CVE-2017-7692: Authenticated remote code execution in SquirrelMail
- CVE-2016-10074: remote code execution in SwiftMailer
- CVE-2016-10033: remote code execution in PHPMailer
- CVE-2016-9920: Unauthenticated remote code execution in Roundcube
- CVE-2019-????: Unauthenticated remote code execution in Horde

Cookie stealing via XSS

The goto payload for XSS is often to steal cookies. Like *Suhosin*, we are encrypting the cookies with a secret key, an environment variable (usually the IP of the user) and the user's user-agent. This means that an attacker with an XSS won't be able to use the stolen cookie, since he can't spoof the content of the value of the environment variable for the user. Please do read the *documentation about this feature* if you're planning to use it.

This feature is roughly the same than the Suhosin one.

Having a secret server-side key will prevent anyone (even the user) from reading the content of the cookie, reducing the impact of an application storing sensitive data client-side.

Remote code execution via file-upload

Some PHP applications allows users to upload content like avatars to a forum. Unfortunately, content validation often isn't implemented properly (if at all), meaning arbitrary file upload often leads to an arbitrary code execution, contrary to the documentation.

> Not validating which file you operate on may mean that users can *access sensitive information* in other directories.
>
> —The PHP documentation about file uploads

We're killing it, like Suhosin, by automatically calling a script upon file upload, if it returns something else than `0`, the file will be removed (or stored in a quarantine, for further analysis).

We're recommending to use the vld project inside the script to ensure the file doesn't contain any valid PHP code, with something like this:

```
$ php -d vld.execute=0 -d vld.active=1 -d extension=vld.so $file
```

One could also filter on the file extensions, with something like this:

```
#!/bin/bash
exit $([[ $SP_FILENAME =~ *\.php* ]])
```

Examples of related vulnerabilities

- CVE-2017-6090: Unauthenticated remote code execution in PhpCollab
- EDB-38407: Authenticated remote code execution in GLPI
- CVE-2013-5576: Authenticated remote code execution in Joomla
- CVE-2019-15813: Authenticated remote code execution in Sentrifugo
- CVE-2019-17132: Authenticated remote code execution in vBulletin
- CVE-2020-10682: Authenticated remote code execution in CMS Made Simple
- EDB-19154: Authenticated remote code execution in qdPM

Unserialize-related magic

PHP is able to *serialize* arbitrary objects, to easily store them. Unfortunately, as demonstrated by Stefan Esser in his Shocking News in PHP Exploitation and Utilizing Code Reuse/ROP in PHP Application Exploits slides, it is often possible to gain arbitrary code execution upon deserialization of user-supplied serialized objects.

> Do not pass untrusted user input to `unserialize()` regardless of the options value of allowed_classes. Unserialization can result in code being loaded and executed due to object instantiation and autoloading and a malicious user may be able to exploit this.
>
> —The PHP documentation about serialize

We're killing it by exploiting the fact that PHP will discard any garbage found at the end of a serialized object, allowing us to simply append a HMAC at the end of strings generated by the `serialize`, hence guaranteeing that any object deserialized came from the application and wasn't tampered with.

We aren't encrypting it, like we do with the cookies, allowing this feature to be disabled (or switch into leaning mode) without the need to invalidate any data.

Warning: This feature can't be deployed on websites that already stored serialized objects (ie. in database), since they are missing the HMAC and thus will be detected as an attack. If you're in this situation, you should use this feature with the `simulation` mode, and switch it off once you don't have any messages in your logs.

A nice side-effect of this feature is that it will defeat various memory corruption issues related to the complexity of `unserialize`'s implementation, and the amount of control if provides to an attacker, like CVE-2016-9137, CVE-2016-9138, 2016-7124, CVE-2016-5771 and CVE-2016-5773.

A less subtle mitigation can be used to simply prevent the deserialization of objects altogether.

Examples of related vulnerabilities

- CVE-2012-5692: Unauthenticated remote code execution in IP.Board
- CVE-2014-1691: Unauthenticated remote code execution in Horde
- CVE-2015-7808: Unauthenticated remote code execution in vBulletin
- CVE-2015-8562: Unauthenticated remote code execution in Joomla
- CVE-2016-4010: Unauthenticated remote code execution in Magento
- CVE-2016-5726: Unauthenticated remote code execution in Simple Machines Forums
- CVE-2016-????: Unauthenticated remote code execution in Observium (leading to remote root)
- CVE-2017-2641: Unauthenticated remote code execution in Moodle
- CVE-2018-17057: Unauthenticated remote code execution in LimeSurvey
- CVE-2018-19274: Authenticated remote code execution in phpBB
- CVE-2019-6340: Unauthenticated remote code execution in Drupal

Weak-PRNG via rand/mt_rand

The functions `rand` and `mt_rand` are often used to generate random numbers used in sensitive context, like password generation, token creation. Unfortunately, as stated in the documentation, the quality of their entropy is low, leading to the generation of guessable values.

> This function does not generate cryptographically secure values, and should not be used for cryptographic purposes.
>
> —The PHP documentation about rand

We're addressing this issue by replacing every call to `rand` and `mt_rand` with a call to the `random_int`, a CSPRNG.

It's worth noting that the PHP documentation contains the following warning:

> `min` `max` range must be within the range `getrandmax()`. i.e. `(max - min) <= getrandmax()`. Otherwise, `rand()` may return poor-quality random numbers.
>
> —The PHP documentation about rand

This is of course addressed as well by the `harden_rand` feature.

Examples of related vulnerabilities

- CVE-2015-5267: Unauthenticated accounts takeover in in Moodle
- CVE-2014-9624: Captcha bypass in MantisBT
- CVE-2014-6412: Unauthenticated account takeover in Wordpress
- CVE-2015-????: Unauthenticated accounts takeover in Concrete5
- CVE-2013-6386: Unauthenticated accounts takeover in Drupal
- CVE-2010-????: Unauthenticated accounts takeover in MyBB
- CVE-2008-4102: Unauthenticated accounts takeover in Joomla
- CVE-2006-0632: Unauthenticated account takeover in phpBB

XXE

Despite the documentation saying nothing about this class of vulnerabilities, XML eXternal Entity (XXE) often leads to arbitrary file reading, SSRF and sometimes even arbitrary code execution.

XML documents can contain a Document Type Definition (DTD), enabling definition of XML entities. It is possible to define an (external) entity by a URI, that the parser will access and embed its content back into the document for further processing.

For example, providing an url like `file:///etc/passwd` will read the file's content. Since the file is not valid XML, the application will spit it out in an error message, thus leaking its content.

We're killing this class of vulnerabilities by calling the libxml_disable_entity_loader function with its parameter set to `true` at startup, and then *nop'ing* it, so it won't do anything if ever called again.

Examples of related vulnerabilities

- CVE-2015-5161: Unauthenticated arbitrary file disclosure on Magento
- CVE-2014-8790: Unauthenticated remote code execution in GetSimple CMS
- CVE-2011-4107: Authenticated local file disclosure in PHPMyAdmin

Cookie stealing via HTTP MITM

While it's possible to set the `secure` flag on cookies to prevent them from being transmitted over HTTP, and only allow its transmission over HTTPS. Snuffleupagus can automatically set this flag if the client is accessing the website over a secure connection.

This behaviour is suggested in the documentation:

> On the server-side, it's on the programmer to send this kind of cookie only on secure connection (e.g. with respect to `$_SERVER["HTTPS"]`).
>
> —The PHP documentation about setcookie

1.1.2 Exploitation, post-exploitation and general hardening

Virtual-patching

PHP itself exposes a number of functions that might be considered **dangerous** and that have limited legitimate use cases. `system()`, `exec()`, `dlopen()` - for example - fall into this category. By default, PHP only allows us to globally disable some functions.

However, (ie. `system()`) they might have legitimate use cases in processes such as self upgrade etc., making it impossible to effectively disable them - at the risk of breaking critical features.

Snuffleupagus allows the user to restrict usage of specific functions per file, or per file with a matching (sha256) hash, thus allowing the use of such functions **only** in the intended places. It can also restrict per CIDR, to restrict execution to users on the LAN for example. There are a *lot* of different filters, so make sure to read the *corresponding documentation*.

Furthermore, running the following script will generate an hash and line-based whitelist of dangerous functions, droping them everywhere else:

```php
<?php

function help($name) {
        die("Usage: $name [-h|--help] [--without-hash] folder\n");
}

if ($argc < 2) {
        help($argv[0]);
}

$functions_blacklist = ['shell_exec', 'exec', 'passthru', 'php_uname', 'popen',
        'posix_kill', 'posix_mkfifo', 'posix_setpgid', 'posix_setsid', 'posix_setuid',
        'posix_setgid', 'posix_uname', 'proc_close', 'proc_nice', 'proc_open',
        'proc_terminate', 'proc_open', 'proc_get_status', 'dl', 'pnctl_exec',
        'pnctl_fork', 'system', 'curl_exec', 'curl_multi_exec', 'function_exists'];

// In PHP < 8.0x you could execute code using assert()
if (PHP_VERSION_ID < 80000) {
        $functions_blacklist[] = 'assert';
}

$extensions = ['php', 'php7', 'php5', 'inc'];

$path = realpath($argv[count($argv) - 1]);
$parsedArgs = getopt('h', ['without-hash', 'help']);

if (isset($parsedArgs['h']) || isset($parsedArgs['help'])) {
        help($argv[0]);
}

$useHash = !isset($parsedArgs['without-hash']);

$output = Array();

$objects = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($path));
foreach($objects as $name => $object){
        if (FALSE === in_array (pathinfo($name, PATHINFO_EXTENSION), $extensions, true))
↪{
                continue;
        }

        $hash = '';
        $file_content = file_get_contents($name);

        if ($useHash) {
                $hash = '.hash("' . hash('sha256', $file_content) . '")';
        }

        $tokens = token_get_all($file_content);

        foreach ($tokens as $pos => $token) {
                if (!is_array($token)) {
                        continue;
```

(continues on next page)

(continued from previous page)

```
                }

                if (isset($token[1][0]) && '\\' === $token[1][0]) {
                        $token[1] = substr($token[1], 1);
                }

                if (!in_array($token[1], $functions_blacklist, true)) {
                        continue;
                }

                $prev_token = find_previous_token($tokens, $pos);

                // Ignore function definitions and class calls
                // function shell_exec() -> ignored
                // $db->exec() -> ignored
                // MyClass::assert() -> ignored
                if ($prev_token === T_FUNCTION
                        || $prev_token === T_DOUBLE_COLON
                        || $prev_token === T_OBJECT_OPERATOR) {
                        continue;
                }

                $output[] = 'sp.disable_function.function("' . $token[1] . '").filename("
↪' . $name . '")' . $hash . '.allow();' . "\n";
        }
}
foreach($functions_blacklist as $fun) {
        $output[] = 'sp.disable_function.function("' . $fun . '").drop();' . "\n";
}

foreach (array_unique($output) as $line) {
         echo $line;
}

function find_previous_token(array $tokens, int $pos): ?int
{
        for ($i = $pos - 1; $i >= 0; $i--) {
                $token = $tokens[$i];

                if ($token[0] === T_WHITESPACE) {
                        continue;
                }

                if (!is_array($token)) {
                        return null;
                }

                return $token[0];
        }

        return null;
}
```

The intent is to make post-exploitation process (such as backdooring of legitimate code, or RAT usage) a lot harder for the attacker.

Global strict mode

By default, PHP will coerce values of the wrong type into the expected one if possible. For example, if a function expecting an integer is given a string, it will be coerced in an integer.

PHP7 introduced a **strict mode**, in which variables won't be coerced anymore, and a TypeError exception will be raised if the types aren't matching. Scalar type declarations are optional, but you don't have to use them in your code to benefit from them, since every internal function from php has them.

This option provides a switch to globally activate this strict mode, helping to uncover vulnerabilities like the classical strcmp bypass and various other types mismatch.

This feature is largely inspired from the autostrict module from krakjoe.

PHP8 already has this feature for internal functions.

Preventing sloppy comparisons

The aforementioned *strict mode* only works with annotated types and native functions, so it doesn't cover every instances of type juggling during comparisons. Since comparison between different types in PHP is notoriously difficult to get right, Snuffleupagus offers a way to **always** use the `identical` operator instead of the `equal` one (see the operator section for PHP's documentation for more details), so that values with different types will always be treated as being different.

Keep in mind that this feature will not only affect the == operator, but also the in_array, array_search and array_keys functions.

PHP8 is implementing a subset of this feature.

Preventing execution of writable PHP files

If an attacker manages to upload an arbitrary file or to modify an existing one, odds are that (thanks to the default umask) this file is writable by the PHP process.

Snuffleupagus can prevent the execution of this kind of file. A good practice would be to use a different user to run PHP than for administrating the website, and using this feature to lock this up.

Whitelist of stream-wrappers

Php comes with a lot of different stream wrapper, and most of them are enabled by default.

The only way to tighten a bit this exposition surface is to use the allow_url_fopen/allow_url_include configuration options, but it's not possible to deactivate them on an individual basis.

Examples of related vulnerabilities

- RCE via phar://
- Data exfiltration via stream wrapper
- Inclusion via zip/phar

White and blacklist in `eval`

While eval is a dangerous primitive, tricky to use right, with almost no legitimate usage besides templating and building mathematical expressions based on user input, it's broadly (mis)used all around the web.

Snuffleupagus provides a white and blacklist mechanism, to explicitly allow and forbid specific function calls from being issued inside `eval`.

While it's heavily recommended to only use the whitelist feature, the blacklist one exists because some sysadmins might want to use it to catch automated script-kiddies attacks, while being confident that doing so won't break a single website.

Protection against cross site request forgery

Cross-site request forgery, sometimes abbreviated as *CSRF*, is when unauthorised commands are issued from a user that the application trusts. For example, if a user is authenticated on a banking website, an other site might present something like `<img src="http://mybank.com/transfer?from=user&to=attack&amount=1337EUR">`, effectively transferring money from the user's account to the attacker one.

Snuffleupagus can prevent this (in supported browsers) by setting the samesite attribute on cookies.

Dumping capabilities

It's possible to apply the `dump()` filter to any virtual-patching rule, to dump the complete web request, along with the filename and the corresponding line number. By using the *right* set of restrictive rules (or by using the *overly* restrictives ones in `simulation` mode), you might be able to gather interesting vulnerabilities used against your website.

Dumps are stored in the folder that you pass to the `dump()` filter, in files named `sp_dump.SHA` with `SHA` being the *sha256* of the rule that matched. This approach allows to mitigate denial of services attacks that could fill up your filesystem.

Misc low-hanging fruits in the default configuration file

Snuffleupagus is shipping with a default configuration file, containing various examples and ideas of things that you might want to enable (or not).

Available functions recon

Usually after compromising a website the attacker does some recon within its webshell, to check which functions are available to execute arbitrary code. Since it's not uncommon for some web-hosts to disable things like `system` or `passthru`, or to check if mitigations are enabled, like `open_basedir`. This behaviour can be detected by preventing the execution of functions like `ini_get` or `is_callable` with *suspicious* parameters.

`chmod` hardening

Some PHP applications are using broad rights when using the `chmod` function, like the infamous `chmod(777)` command, effectively making the file writable by everyone. Snuffleupagus is preventing this kind of behaviour by restricting the parameters that can be passed to `chmod`.

Arbitrary file inclusion hardening

Arbitrary file inclusion is a common vulnerability, that might be detected by preventing the inclusion of anything that doesn't match a strict set of file extensions in calls to `include` or `require`.

Enforcing certificate validation when using curl

While it might be convenient to disable certificate validation on preproduction or during tests, it's common to see that people are disabling it on production too. We're detecting/preventing this by not allowing the `CURLOPT_SSL_VERIFYPEER` and `CURLOPT_SSL_VERIFYHOST` options from being set to `0`.

Cheap error-based SQL injections detection

If a function performing a SQL query returns `FALSE` (indicating an error), it might be useful to dump the request for further analysis.

1.2 Installation

Snuffleupagus is tested against various PHP 7+ versions.

1.2.1 Manual installation

Depending on the system, we might already offer binary packages. You can check our *Download*. In that case you only need to activate the extension inside your `php.ini` and to configure it.

Requirements

The only dependency (at least on Debian) to compile Snuffleupagus is php7.0-dev or onwards.

Quickstart

```
git clone https://github.com/jvoisin/snuffleupagus
cd snuffleupagus/src
phpize
./configure --enable-snuffleupagus
make
make install
```

This should install the `snuffleupagus.so` file in your extension directory. The final step is adding an extension loading directive, and to specify the location of the *configuration file*, either in a `conf.d/20-snuffleupagus.ini` file, or directly in you `php.ini` if you prefer:

```
extension=snuffleupagus.so

# This is only an example,
# you can place your rules wherever you want.
sp.configuration_file=/etc/php/conf.d/snuffleupagus.rules
```

Be careful, on some distribution, there are separate configurations for cli/fpm/cgi/… be sure to edit the right one.

If you're using Gentoo, you might encounter the following error:

```
$ make
$ /bin/sh /root/snuffleupagus-0.5.0/src/libtool --mode=compile cc  -I. -I/root/
↪snuffleupagus-0.5.0/src -DPHP_ATOM_INC -I/root/snuffleupagus-0.5.0/src/include -I/root/
↪snuffleupagus-0.5.0/src/main -I/root/snuffleupagus-0.5.0/src -I/usr/lib64/php7.3/
↪include/php -I/usr/lib64/php7.3/include/php/main -I/usr/lib64/php7.3/include/php/TSRM -
↪I/usr/lib64/php7.3/include/php/Zend -I/usr/lib64/php7.3/include/php/ext -I/usr/lib64/
↪php7.3/include/php/ext/date/lib  -DHAVE_CONFIG_H  -g -O2 -Wall -Wextra -Wno-unused-
↪parameter -Wformat=2 -Wformat-security -D_FORTIFY_SOURCE=2 -fstack-protector   -c /
↪root/snuffleupagus-0.5.0/src/snuffleupagus.c -o snuffleupagus.lo
libtool: Version mismatch error.  This is libtool 2.4.6, but the
libtool: definition of this LT_INIT comes from an older release.
libtool: You should recreate aclocal.m4 with macros from libtool 2.4.6
libtool: and run autoconf again.
make: *** [Makefile:193: snuffleupagus.lo] Error 63
$
```

This is a documented php bug, solvable via:

```
rm -f aclocal.m4
phpize
aclocal && libtoolize --force && autoreconf
./configure --enable-snuffleupagus
make
```

1.2.2 Heroku installation

Heroku's official buildpack uses `Composer` to install all dependencies required by your PHP application. Careful with the default set of rules, since it might block the composer deployment, leading to the following errors:

```
heroku[web.1]: Starting process with command `vendor/bin/heroku-php-apache2 -F fpm_
↪custom.conf public/`
heroku[web.1]: Stopping all processes with SIGTERM
app[web.1]: Stopping httpd...
app[web.1]: SIGTERM received, attempting graceful shutdown...
app[web.1]: Stopping php-fpm...
app[web.1]: Shutdown complete.
heroku[web.1]: Process exited with status 143
app[web.1]: [heroku-exec] Starting
app[web.1]: Unable to determine Composer vendor-dir setting; is 'composer' executable on␣
↪path or 'composer.phar' in current working directory?
heroku[web.1]: Process exited with status 1
heroku[web.1]: State changed from starting to crashed
```

Requirements

To install snuffleupagus on heroku, simply follow the documentation, and edit the `composer.json` file, as well as the `Procfile` to load the additional PHP-FPM configuration.

composer.json

```
{
    "require": {
        "php": "~7.4.6"
    },
    "config": {
        "platform": {
            "php": "7.4.6"
        }
    },
    "scripts": {
        "compile": [
            "git clone https://github.com/jvoisin/snuffleupagus /tmp/snuffleupagus",
            "cd /tmp/snuffleupagus/src && phpize && ./configure --enable-snuffleupagus &&
↪ make && make install",
            "echo 'extension=snuffleupagus.so\nsp.allow_broken_configuration=on\nsp.
↪configuration_file=/dev/null' > /app/.heroku/php/etc/php/conf.d/999-ext-snuffleupagus.
↪ini"
        ]
    }
}
```

This configuration will compile Snuffleupagus to shared library, install it to the proper location and specify an empty configuration in `sp.configuration_file` to ensure that the `composer` deployment phase won't get killed by some rules.

PHP-FPM

```
; ext-snuffleupagus
php_admin_flag[sp.allow_broken_configuration] = off
php_admin_value[sp.configuration_file]        = /app/default.rules
```

The final step is to point `sp.configuration_file` to a rule set by setting the preference in an additional PHP-FPM configuration.

You should now be running Snuffleupagus in PHP on heroku:

```
app[web.1]: [05-Jul-2020 07:45:22 UTC] PHP Fatal error:  [snuffleupagus][0.0.0.
↪0][disabled_function] Aborted execution on call of the function 'exec', because its␣
↪argument '$command' content (id;whoami) matched a rule in /app/public/test2.php on␣
↪line 1
app[web.1]: 10.9.226.141 - - [05/Jul/2020:07:45:22 +0000] "GET /test2.php?cmd=id;whoami␣
↪HTTP/1.1" 500 - "-" "curl/7.68.0
heroku[router]: at=info method=GET path="/test2.php?cmd=id;whoami" host=heroku-x-
↪snuffleupagus.herokuapp.com request_id=012345678-9012-3456-7890-123456789012 fwd="1.2.
↪3.4" dyno=web.1 connect=0ms service=7ms status=500 bytes=169 protocol=http
```

1.2.3 Upgrading

Upgrading Snuffleupagus is as simple as recompiling it (or using a binary), replacing the file and restarting your webserver.

1.3 Configuration

> **Warning:** If you configure Snuffleupagus incorrectly, your website *might* not work correctly until either you fix your configuration, or revert your changes altogether.
>
> It's up to you to understand the *features*, read the present documentation about how to configure them, evaluate your threat model and write your configuration file accordingly.

Since PHP *ini-like* configuration model isn't flexible enough, Snuffleupagus is using its own format in the file specified by the directive `sp.configuration_file` **in** your `php.ini` file, like `sp.configuration_file=/etc/php/conf.d/snuffleupagus.rules`.

You can use the `,` separator to include multiple configuration files: `sp.configuration_file=/etc/php/conf.d/snuffleupagus.rules,/etc/php/conf.d/sp_wordpress.rules`.

We're also also supporting glob, so you can write something like: `sp.configuration_file=/etc/php/conf.d/*.rules,/etc/php/conf.d/extra/test.rules`.

To sum up, you should put this in your `php.ini`:

```
module=snuffleupagus.so
sp.configuration_file=/path/to/your/snuffleupagus/rules/file.rules
```

And the **snuffleupagus rules** into the `.rules` files.

Since our configuration format is a bit more complex than php's one, we have a `sp.allow_broken_configuration` parameter (`false` by default), that you can set to `true` if you want PHP to carry on if your Snuffleupagus' configuration contains syntax errors. You'll still get a big scary message in your logs of course. We do **not** recommend to use it of course, but sometimes it might be useful to be able to "debug in production" without breaking your website.

1.3.1 Configuration file format

Options are chainable by using dots (`.`).

Some options have a string parameter, that **must** be quoted with double quotes, e.g. `"string"`.

Comments are prefixed either with `#`, or `;`.

Some rules apply in a specific `function` (context) on a specific `variable` (data), like `disable_function`. Others can only be enabled/disabled, like `harden_random`.

Most of the features can be used in `simulation` mode by appending the `.simulation()` or `.sim()` option to them (eg. `sp.readonly_exec.simulation().enable();`) to see whether or not they could break your website. The simulation mode won't block the request, but will write a warning in the log.

The rules are evaluated in the order that they are written, the **first** one to match will terminate the evaluation (except for rules in simulation mode).

Rules can be split into lines and contain whitespace for easier readability and maintenance: (This feature is available since version 0.8.0.)

```
sp.disable_function.function("mail")
  .param("to").value_r("\\n")
  .alias("newline in mail() To:")
  .drop();
```

The terminating `;` is optional for now, but it should be used for future compatibility.

Rules, including comments, needs to be written in ASCII, other encodings aren't supported and might cause syntax errors and related issues like making all rules after non-ASCII symbols not considered for execution and silently discarded.

1.3.2 Miscellaneous

conditions

It's possible to use conditions to have configuration portable across several setups.

```
@condition PHP_VERSION_ID < 80000;
  # some rules
@condition PHP_VERSION_ID >= 80000;
  # some other rules
@end_condition;
```

Conditions accept variables and the special function `extension_loadod()`.

```
@condition extension_loaded("sqlite3");
sp.ini.key("sqlite3.extension_dir").ro();
@end_condition;
```

Conditions cannot be nested, but arithmetic and logical operations can be applied.

```
@condition extension_loaded("session") && PHP_VERSION_ID <= 80200;
set whitelist "my_fun,cos"
sp.eval_whitelist.list(whitelist).simulation().dump("/tmp/dump_result/");
@end_condition;
```

variables

You may set a configuration variable using the `set` keyword (or `@set`) and use it instead of arguments.

```
@set CMD "ls"
sp.disable_function.function("system").pos("0").value(CMD).allow();
```

global

This configuration variable contains parameters that are used by multiple features:

- `secret_key`: A secret key used by various cryptographic features, like cookies protection or unserialize protection, please ensure the length and complexity is sufficient. You can generate it with functions such as: `head -c 256 /dev/urandom | tr -dc 'a-zA-Z0-9'`.

```
sp.global.secret_key("44239bd400aa82e125337c9d4eb8315767411ccd");
```

- `cookie_env_var`: A environment variable used as part of cookies encryption. See the *relevant documentation*

log_media

This configuration variable allows to specify how logs should be written, either via `php` or `syslog`.

```
sp.log_media("php");
sp.log_media("syslog");
```

The default value for `sp.log_media` is `php`, to respect the principle of least astonishment. But since it's possible to modify php's logging system via php, it's heavily recommended to use the `syslog` option instead.

log_max_len

This configuration variable allows to specify (roughly) the size of the log.

```
sp.log_max_len("16");
```

The default value for `sp.log_max_len` is `255`.

1.3.3 Bugclass-killer features

global_strict

global_strict, disabled by default, will enable the strict mode globally, forcing PHP to throw a TypeError exception if an argument type being passed to a function does not match its corresponding declared parameter type.

It can either be `enabled` or `disabled`.

```
sp.global_strict.disable();
sp.global_strict.enable();
```

harden_random

harden_random, enabled by default, will silently replace the insecure rand and mt_rand functions with the secure PRNG random_int.

It can either be `enabled` or `disabled`.

```
sp.harden_random.enable();
sp.harden_random.disable();
```

Prevent sloppy comparison

Sloppy comparison prevention, disabled by default, will prevent php type juggling (==): two values with different types will always be different.

It can either be `enabled` or `disabled`.

```
sp.sloppy_comparison.enable();
sp.sloppy_comparison.disable();
```

unserialize_noclass

unserialize_noclass, available only on PHP8+ and disabled by default, will disable the deserialization of objects via `unserialize`. It's equivalent to setting the `options` parameter of `unserialize` to `false`, on every call. It can either be `enabled` or `disabled`.

```
sp.unserialize_noclass.enable();
sp.unserialize_noclass.disable();
```

unserialize_hmac

unserialize_hmac, disabled by default, will add an integrity check to `unserialize` calls, preventing arbitrary code execution in their context.

It can either be `enabled` or `disabled` and can be used in `simulation` mode.

```
sp.unserialize_hmac.enable();
sp.unserialize_hmac.disable();
```

Warning: This feature breaks web applications doing checks on the serialized representation of data on their own, like WordPress.

Cookies-related mitigations

Since snuffleupagus is providing several hardening features for cookies, there is a dedicated web page *here* about them.

INI Settings Protection

INI settings can be forced to a value, limited by min/max value or regular expression and set read-only mode.

First, this feature can be enabled or disabled:

```
sp.ini_protection.enable();
sp.ini_protection.disable();
```

The INI protection feature can be set to simulation mode, where violations are only reported, but rules are not enforced:

```
sp.ini_protection.simulation();
```

Rule violations can be set to drop as a global policy, or alternatively be set on individual rules using `.drop()`.

```
sp.ini_protection.policy_drop();
```

Rules can be set to fail silently without logging anything:

```
sp.ini_protection.policy_silent_fail();
## or write sp.ini_protection.policy_no_log(); as an alias
```

Read-only settings are implemented in a way that the PHP system itself can block the setting, which is very efficient. If you do not need to log read-only violations, these can be set to silent separately:

```
sp.ini_protection.policy_silent_ro();
```

A global access policy can be set to either read-only or read-write. Individual entries can be set to read-only/read-write as well using `.ro()`/`.rw()`.

```
sp.ini_protection.policy_readonly();
sp.ini_protection.policy_readwrite();
```

Individual rules are specified using `sp.ini`. These entries can have the following attributes:

- `.key("...")`: mandatory ini name.
- `.set("...")`: set the initial value. This overrides php.ini. checks are not performed for this initial value.
- `.min("...")` / `.max("...")`: value must be an integer between .min and .max. shorthand notation (e.g. 1k = 1024) is allowed
- `.regexp("...")`: value must match the regular expression
- `.allow_null()`: allow setting a NULL-value
- `.msg("...")`: message is shown in logs on rule violation instead of default message
- `.readonly()` / `.ro()` / .readwrite() / .rw(): set entry to read-only or read-write respectively. If no access keyword is provided, the entry inherits the default policy set by `sp.ini_protection.policy_*`-rules.

- `.drop()`: drop request on rule violation for this entry
- `.simulation()`: only log rule violation for this entry

Examples:

```
sp.ini.key("display_errors").set("0").ro();
sp.ini.key("default_socket_timeout").min("1").max("300").rw();
sp.ini.key("highlight.comment").regexp("^#[0-9a-fA-F]{6}$");
```

For more examples, check out the `config` directory.

readonly_exec

readonly_exec, disabled by default, will prevent the execution of writeable PHP files.

It can either be `enabled` or `disabled` and can be used in `simulation` mode. `extended_checks` can be specified to abort the execution if the executed file or the folder containing it is owned by the user the PHP process is running under.

Extended checks, enabled by default, can be explicitly enabled via `extended_checks` and disabled via `no_extended_checks`. The checks include:

- verifying the effective user id;
- verifying that the current folder isn't writable;
- verifying the current folder effective user id.

```
sp.readonly_exec.enable();
```

upload_validation

upload_validation, disabled by default, will call a given script upon a file upload, with the path to the file being uploaded as argument and various information about it in the environment:

- `SP_FILENAME`: the name of the uploaded file
- `SP_FILESIZE`: the size of the file being uploaded
- `SP_REMOTE_ADDR`: the ip address of the uploader
- `SP_CURRENT_FILE`: the current file being executed

This feature can be used, for example, to check if an uploaded file contains php code, using vld, via a python script, or a php one.

The upload will be **allowed** if the script returns the value `0`. Every other value will prevent the file from being uploaded.

It can either be `enabled` or `disabled` and can be used in `simulation` mode.

```
sp.upload_validation.script("/var/www/is_valid_php.py").enable();
```

xxe_protection

xxe_protection, disabled by default, will prevent XXE attacks by disabling the loading of external entities (`libxml_disable_entity_loader`) in the XML parser.

```
sp.xxe_protection.enable();
sp.xxe_protection.disable();
```

Whitelist of stream-wrappers

Stream-wrapper whitelist allows to explicitly whitelist some stream wrappers.

```
sp.wrappers_whitelist.list("file,php,phar");
```

Eval white and blacklist

eval_whitelist and eval_blacklist, disabled by default, allow to respectively specify functions allowed and forbidden from being called inside `eval`. The functions names are comma-separated.

```
sp.eval_blacklist.list("system,exec,shell_exec");
sp.eval_whitelist.list("strlen,strcmp").simulation();
```

The whitelist comes before the black one: if a function is both whitelisted and blacklisted, it'll be allowed.

1.3.4 Virtual-patching

Snuffleupagus provides virtual-patching via the `disable_function` directive, allowing you to stop or control dangerous behaviours. In the situation where you have a call to `system()` that lacks proper user-input validation, this could cause issues as it would lead to an **RCE**. The virtual-patching would allow this to be prevented.

```
# Allow `id.php` to restrict system() calls to `id`
sp.disable_function.function("system").filename("/var/www/html/id.php").param("cmd").
↪value("id").allow();
sp.disable_function.function("system").filename("/var/www/html/id.php").drop()
```

Of course, this is a trivial example, a lot can be achieved with this feature, as you will see below.

Filters

- `alias(description)`: human-readable `description` of the rule
- `cidr(ip/mask)`: match on the client's cidr
- `filename(name)`: exact match on the file's `name`
- `filename_r(regexp)`: file name matching the `regexp`
- `function(name)`: exact match on function `name`
- `function_r(regexp)`: function name matching the `regexp`
- `hash(sha256)`: exact match on the file's sha256 sum
- `line(line_number)`: exact match on the file's line.

- `param(name)`: exact match on the function's parameter `name`
- `param_r(regexp)`: match on the function's parameter `regexp`
- `param_type(type)`: exact match on the function's parameter `type`
- `pos(nth_argument)`: exact match on the nth argument, starting from `0`
- `ret(value)`: exact match on the function's return `value`
- `ret_r(regexp)`: match with a `regexp` on the function's return
- `ret_type(type_name)`: match on the `type_name` of the function's return value
- `value(value)`: exact match on a literal `value`
- `value_r(regexp)`: match on a value matching the `regexp`
- `var(name)`: exact match on a **local variable** `name`
- `key(name)`: exact match on the presence of `name` as a key in the hashtable
- `key_r(regexp)`: match with `regexp` on keys in the hashtable

The `type` must be one of the following values:

- `FALSE`: for boolean false
- `TRUE`: for boolean true
- `NULL`: for the **null** value
- `LONG`: for a long (also know as `integer`) value
- `DOUBLE`: for a **double** (also known as `float`) value
- `STRING`: for a string
- `OBJECT`: for a object
- `ARRAY`: for an array
- `RESOURCE`: for a resource

Actions

Every rule *must* have one action.

- `allow()`: **allow** the request if the rule matches
- `drop()`: **drop** the request if the rule matches

Modifications

- `dump(directory)`: dump the request in the `directory` if it matches the rule
- `simulation()`: enabled the simulation mode

Details

The `function` filter is able to do various dereferencing:

- `function("AwesomeClass::my_method")` will match the method `my_method` in the class `AwesomeClass`
- `function("AwesomeNamespace\\my_function")` will match the function `my_function` in the namespace `AwesomeNamespace`

It's also able to have calltrace constrains: `function(func1>func2)` will match only if `func2` is called **inside** of `func1`. Do note that their might be other functions called between them.

The `param` filter is able to do some dereferencing as well:

- `param($foo[bar])` will get a match on the value corresponding to the `bar` key in the hashtable `foo`. Remember that in PHP, almost every data structure is a hashtable. You can of course nest this like `param($foo[bar][$object->array['123']][$batman])`.
- The `var` filter will walk the calltrace until it finds the variable name, or the end of the calltrace, allowing the filter to match global variables: `.var("$_GET[\"param\"]")` will match on the GET parameter `param`.

The `filename` filter requires a leading `/`, since paths are absolutes (like `/var/www/mywebsite/lib/parse.php`). If you would like to have only one configuration file for several vhost in different folders, you can use the `filename_r` directive to match on the filename (like `/lib/parse\.php`). Please do note that this filter matches on the file where the function is **defined**, not the one where the function is **called from**.

For clarity, the presence of the `allow` or `drop` action is **mandatory**.

In the logs, the parameters and the return values of function are url-encoded, to accommodate fragile log processors.

> **Warning:** When you're writing rules, please do keep in mind that **the order matters**. For example, if you're denying a call to `system()` and then allowing it in a more narrowed way later, the call will be denied, because it'll match the deny first.

If you're paranoid, we're providing a php script to automatically generate hash of files containing dangerous functions, and blacklisting them everywhere else.

Limitations

It's currently not possible to:

- Hook every language construct, because each of them requires a specific implementation. It's also not possible to hook them via regular expression.
- Use extra-convoluted rules for matching, like `${$A}$$B->${'}[1]`, because if you're writing things like this, odds are that you're doing something wrong anyway.
- Hooks on `echo` and on `print` are equivalent: there is no way to hook one without hooking the other, at least for now). This is why hooked `print` will be displayed as `echo` in the logs.
- Hook *strlen*, since in latest PHP versions, this function is usually optimized away by the compiler.

Examples

Evaluation order of rules

The following rules will:

1. Allow calls to `system("id")`
2. Issue a trace in the logs on calls to `system` with its parameters starting with `ping`, and pursuing evaluation of the remaining rules.
3. Drop calls to `system`.

```
sp.disable_function.function("system").param("cmd").value("id").allow();
sp.disable_function.function("system").param("cmd").value_r("^ping").drop().simulation();
sp.disable_function.function("system").param("cmd").drop();
```

Miscellaneous examples

```
# This is the default configuration file for Snuffleupagus (https://snuffleupagus.rtfd.
↪io).
# It contains "reasonable" defaults that won't break your websites,
# and a lot of commented directives that you can enable if you want to
# have a better protection.

# Harden the PRNG
sp.harden_random.enable();

# Enable XXE protection
@condition extension_loaded("xml");
sp.xxe_protection.enable();
@end_condition;

# Global configuration variables
# sp.global.secret_key("YOU _DO_ NEED TO CHANGE THIS WITH SOME RANDOM CHARACTERS.");

# Globally activate strict mode
# https://www.php.net/manual/en/language.types.declarations.php#language.types.
↪declarations.strict
# sp.global_strict.enable();

# Prevent unserialize-related exploits
# sp.unserialize_hmac.enable();

# Only allow execution of read-only files. This is a low-hanging fruit that you should␣
↪enable.
# sp.readonly_exec.enable();

# PHP has a lot of wrappers, most of them aren't usually useful, you should
# only enable the ones you're using.
# sp.wrappers_whitelist.list("file,php,phar");

# Prevent sloppy comparisons.
```

(continues on next page)

(continued from previous page)

```
# sp.sloppy_comparison.enable();

# Use SameSite on session cookie
# https://snuffleupagus.readthedocs.io/features.html#protection-against-cross-site-
↪request-forgery
sp.cookie.name("PHPSESSID").samesite("lax");

# Harden the `chmod` function (0777 (oct = 511, 0666 = 438)
sp.disable_function.function("chmod").param("mode").value("438").drop();
sp.disable_function.function("chmod").param("mode").value("511").drop();

# Prevent various `mail`-related vulnerabilities
sp.disable_function.function("mail").param("additional_parameters").value_r("\\-").
↪drop();

# Since it's now burned, me might as well mitigate it publicly
sp.disable_function.function("putenv").param("setting").value_r("LD_").drop()
sp.disable_function.function("putenv").param("setting").value("PATH").drop()

# This one was burned in Nov 2019 - https://gist.github.com/LoadLow/
↪90b60bd5535d6c3927bb24d5f9955b80
sp.disable_function.function("putenv").param("setting").value_r("GCONV_").drop()

# Since people are stupid enough to use `extract` on things like $_GET or $_POST, we␣
↪might as well mitigate this vector
sp.disable_function.function("extract").pos("0").value_r("^_").drop()
sp.disable_function.function("extract").pos("1").value("0").drop()

# This is also burned:
# ini_set('open_basedir','..');chdir('..');...;chdir('..');ini_set('open_basedir','/');echo(file_
↪get_contents('/etc/passwd'));
# Since we have no way of matching on two parameters at the same time, we're
# blocking calls to open_basedir altogether: nobody is using it via ini_set anyway.
# Moreover, there are non-public bypasses that are also using this vector ;)
sp.disable_function.function("ini_set").param("varname").value_r("open_basedir").drop()

# Prevent various `include`-related vulnerabilities
sp.disable_function.function("require_once").value_r("\.(inc|phtml|php)$").allow();
sp.disable_function.function("include_once").value_r("\.(inc|phtml|php)$").allow();
sp.disable_function.function("require").value_r("\.(inc|phtml|php)$").allow();
sp.disable_function.function("include").value_r("\.(inc|phtml|php)$").allow();
sp.disable_function.function("require_once").drop()
sp.disable_function.function("include_once").drop()
sp.disable_function.function("require").drop()
sp.disable_function.function("include").drop()

# Prevent `system`-related injections
sp.disable_function.function("system").param("command").value_r("[$|;&`\\n\\(\\)\\\\]").
↪drop();
sp.disable_function.function("shell_exec").pos("0").value_r("[$|;&`\\n\\(\\)\\\\]").
↪drop();
sp.disable_function.function("exec").param("command").value_r("[$|;&`\\n\\(\\)\\\\]").
```

(continues on next page)

(continued from previous page)

```
↪drop();
sp.disable_function.function("proc_open").param("command").value_r("[$|;&`\\n\\(\\)\\\\]
↪").drop();

# Prevent runtime modification of interesting things
sp.disable_function.function("ini_set").param("varname").value("assert.active").drop();
sp.disable_function.function("ini_set").param("varname").value("zend.assertions").drop();
sp.disable_function.function("ini_set").param("varname").value("memory_limit").drop();
sp.disable_function.function("ini_set").param("varname").value("include_path").drop();
sp.disable_function.function("ini_set").param("varname").value("open_basedir").drop();

# Detect some backdoors via environment recon
sp.disable_function.function("ini_get").param("varname").value("allow_url_fopen").drop();
sp.disable_function.function("ini_get").param("varname").value("open_basedir").drop();
sp.disable_function.function("ini_get").param("varname").value_r("suhosin").drop();
sp.disable_function.function("function_exists").param("function_name").value("eval").
↪drop();
sp.disable_function.function("function_exists").param("function_name").value("exec").
↪drop();
sp.disable_function.function("function_exists").param("function_name").value("system").
↪drop();
sp.disable_function.function("function_exists").param("function_name").value("shell_exec
↪").drop();
sp.disable_function.function("function_exists").param("function_name").value("proc_open
↪").drop();
sp.disable_function.function("function_exists").param("function_name").value("passthru").
↪drop();
sp.disable_function.function("is_callable").param("var").value("eval").drop();
sp.disable_function.function("is_callable").param("var").value("exec").drop();
sp.disable_function.function("is_callable").param("var").value("system").drop();
sp.disable_function.function("is_callable").param("var").value("shell_exec").drop();
sp.disable_function.function("is_callable").param("var").value("proc_open").drop();
sp.disable_function.function("is_callable").param("var").value("passthru").drop();

# Ghetto error-based sqli detection
# sp.disable_function.function("mysql_query").ret("FALSE").drop();
# sp.disable_function.function("mysqli_query").ret("FALSE").drop();
# sp.disable_function.function("PDO::query").ret("FALSE").drop();

# Ensure that certificates are properly verified
sp.disable_function.function("curl_setopt").param("value").value("1").allow();
sp.disable_function.function("curl_setopt").param("value").value("2").allow();
# `81` is SSL_VERIFYHOST and `64` SSL_VERIFYPEER
sp.disable_function.function("curl_setopt").param("option").value("64").drop().alias(
↪"Please don't turn CURLOPT_SSL_VERIFYCLIENT off.");
sp.disable_function.function("curl_setopt").param("option").value("81").drop().alias(
↪"Please don't turn CURLOPT_SSL_VERIFYHOST off.");

# File upload
# On old PHP7 versions
#sp.disable_function.function("move_uploaded_file").param("destination").value_r("\\.ph
↪").drop();
```

(continues on next page)

(continued from previous page)

```
#sp.disable_function.function("move_uploaded_file").param("destination").value_r("\\.ht
↪").drop();
# On PHP7.4+
sp.disable_function.function("move_uploaded_file").param("new_path").value_r("\\.ph").
↪drop();
sp.disable_function.function("move_uploaded_file").param("new_path").value_r("\\.ht").
↪drop();

# Logging lockdown
sp.disable_function.function("ini_set").param("varname").value_r("error_log").drop()
sp.disable_function.function("ini_set").param("varname").value_r("error_reporting").
↪drop()
sp.disable_function.function("ini_set").param("varname").value_r("display_errors").drop()
```

1.4 Download

1.4.1 Arch Linux

Thanks to kpcyrd, Snuffleupagus is available in Archlinux' community repository.

We're also providing a PKGBUILD if you want to build the package yourself.

1.4.2 Alpine Linux

We're maintaining the package in Alpine: you can simply `apk add` it.

1.4.3 CloudLinux

Snuffleupagus is packaged there since 2019: you can `yum install alt-php*-snuffleupagus` it.

1.4.4 Debian and Ubuntu

We're currently not providing a Debian/Ubuntu repository, but you can grab the latest release on github, or build your own package by cloning the source code and typing `make debian`.

There is a bug open Debian-side to track the inclusion.

1.4.5 Fedora

Thanks to Rémo Collet, Snuffleupagus is packaged in Fedora!

1.4.6 FreeBSD

Thanks to Franco Fichtner, Snuffleupagus is packaged in FreeBSD!

1.4.7 Source code

We're currently using *github* as public code repository.

```
git clone https://github.com/jvoisin/snuffleupagus
```

1.5 Changelog

1.5.1 0.10.0 - Babar the Elephant 2023/09/20

New features

- Compatibility with PHP8.3
- Add *sp.log_max_len* to limit the maximum size of the log messages
- Add an example configuration for Xenforo 2.2.12

Breaking Changes

- Url encode functions arguments when logging them

Bug fixes

- Fix a possible NULL-byte truncation when outputting parameters in the logs
- Make readonly_exec play nice on readonly filesystems

1.5.2 0.9.0 - Elephant seal 2023/01/03

New features

- Compatibility with PHP8.2
- Add the ability block object unserialization globally.

1.5.3 0.8.3 - Elephant Gambit 2022/08/27

New features

- Add the ability to dump the parameter passed to *eval*
- Add the ability to match on *eval*'s parameter
- Add optional extended checks for *readonly_exec*
- Add config error for ini rules with identical key
- Add disabled functions return type to config export

Breaking Changes

- Mix the stacktrace in the sha256 for the filename of .dump()

Bug fixes

- Make it actually possible to configure sloppy comparison on latests PHP7
- Allow file:// prefix in include() wich readonly_exec mode
- Fix a possible crash when exporting function list
- Fix a minor memory leak when parsing cookie-related configuration

1.5.4 0.8.2 - Surus 2022/05/20

Bug fixes

- Fix compilation when ZTS is used
- Fix a possible infinite loop

1.5.5 0.8.1 - Batyr 2022/05/16

Bug fixes

- Fix the version number
- Fix a test on PHP7

Breaking Changes

- *disable_xxe* is changed to *xxe_protection*

1.5.6 0.8.0 - Woolly Mammoth 2022/05/15

New features

- Compatibility with PHP8.1
- Check for unsupported PHP version
- Backport of Suhosin-ng patches:
  - Maximum stack depth/recursion limit
  - Maximum length for session id
  - $_SERVER strip/encode
  - Configuration dump
  - Support for conditional rules
  - INI settings protection
  - Output SP logs to stderr
  - Ported Suhosin rules to SP

Improvements

- Massive simplification of the configuration parser
- Better memory management
- Removal of internal calls to *call_user_func*
- Increased portability of the default rules access different version of PHP
- Start SP as late as possible, to hook as many things as possible

Bug fixes

- XML and Session support are now checked at runtime instead of at compile time

1.5.7 0.7.1 - Proboscidea 2021/08/02

Improvements

- Improve compatibility with various *libpcre* configurations/versions
- Modernise the code by removing usage of *strtok*
- Improve the default rules' compatibility with php8
- Prevent XXE in php8 as well
- Improve a bit the verbosity of the logs
- Add a rules file for php8

Bug fixes

- Prevent a possible crash during configuration reloading
- Fix the default rules to catch dangerous *chmod* calls
- Fixed possible memory-leaks when hooking via regular expressions

1.5.8 0.7.0 - Los Elefantes 2021/01/02

New features

- PHP8 support
- Stacktraces in dumps
- The > operator now skips over functions

Improvements

- Move the CI from travis to gitlab-ci
- Some code simplifications and constifications
- PCRE2 is now used when possible
- The `generate_rules.php` script is now more portable

Bug fixes

- The strict mode can now be disabled

1.5.9 0.6.0 - Elephant in the room 2020/11/06

New features

- Allow empty configurations

Improvements

- More constification
- Snuffleupagus should now be able to get client's ip addresses in more cases
- Documented compatibility with Heroku
- Improved logging
- Added a couple of tests

1.5.10 0.5.1 - Order of the Elephant 2020/06/20

New features

- Add support for syslog

Improvements

- Improve OSX support
- Improve marginally of php8+ compatibility
- Improve php7.4 compatibility
- Improve the default ruleset
- Improve the documentation
- Improve the gitlab CI

1.5.11 0.5.0 - Elephant Flats 2019/06/12

Improvements

- Tighten a bit a command-injection prevention rule in the default rules set
- Increased the portability of the testsuite
- Improved documentation
- Usual code cleanup
- Snuffleupagus will throw an informative error when compiled for PHP5
- Snuffleupagus will throw an informative error when compiled without PCRE support
- The testsuite is now run on Alpine, Fedora, Debian and Ubuntu.
- Some rules against now-known vulnerabilities/techniques were added

Bug fixes

- PHP7.4 is fully supported, without any compilation warning
- Snuffleupagus can now be used with PHP compiled without sessions support as a builtin (which is the case on Alpine).
- Fix a compilation warning on FreeBSD
- Cookies hardening is now supported on PHP7.3+

1.5.12 0.4.1 - Loxodonta 2018/12/21

Improvements

- Improve and clarify the documentation
- Add support for PHP7.3
- Improve the coverage, we have reached 99% of coverage
- Improve *mb_string* hooking logic
- The script that check uploaded file is now available in PHP

Bug fixes

- Fix segfault on 32-bit for PHP7.3
- Fix segfault when using *sloppy_comparison* feature with array

1.5.13 0.4.0 - Oliphant Chuckerbutty 2018/08/31

New features

- Add the possibility to whitelist stream wrappers
- Snuffleupagus is now using php's logging mechanisms, instead of outputting its log directly into the syslog.
- PHP is now prevented from ever disabling certificate verification thanks to a few lines in our default configuration.

Improvements

- Significant code simplification for cookies handling thanks to Remi Collet
- Our `sloppy comparison` feature is now complete
- Snuffleupagus won't start with an invalid config anymore, except if the `sp.allow_broken_configuration` is set.
- It's now possible to place virtual-patches on the return value of user-defined functions.
- Since Snuffleupagus is used by more and more organisations, we added a bunch of them in our propaganda page.

Bug fixes

- Add some missing pieces of documentation and fix some links
- Fix the `make install` command
- Fix various compilation warnings
- Snuffleupagus is now running on platforms that aren't using the glibc, thanks to an external contributor Antoine Tenart

1.5.14 0.3.1 - Elephant Arch 2018/08/20

Improvements

- Disable XXE and harden PRNG by default
- Use `SameSite` on PHP's session cookie in the default rules
- Relax a bit what files can be included in the default rules
- Add the possibility to ignore files hashes when generating rules
- The `filename` filter is now accepting phar paths

Bug fixes

- The harden rand_feature is not ignoring parameters anymore in function calls
- Fix possible crashes/hangs when using php-fpm's pools
- Fix an infinite loop on `echo` hook
- Fix an issue with `filename` filter
- Fix some documentation issues
- Fix the Arch Linux's PKGBUILD

1.5.15 0.3.0 - Dentalium elephantinum 2018/07/17

New features

- Session cookies can now be encrypted
- Some occurrences of type juggling can now be eradicated
- It's now possible to hook *echo* and *print*

Improvements

- The *.filename()* filter is now matching on the file where the function is called instead on the one where it's defined.
- Vastly optimize the way we hook native functions
- The format of the logs has been streamlined to ease their processing

Bug fixes

- Better handling of filters for built-in functions
- Fix various possible integer overflows
- Fix an annoying memory leak impacting mostly *mod_php*

1.5.16 0.2.2 - Elephant Moraine 2018/04/12

New features

- The *.dump()* filter is now supported for *unserialize*, *readonly_exec*, and *eval* black/whitelist

Improvements

- Add some assertions
- Add more rules examples
- Provide a script to check for malicious file uploads
- Significant performances improvement (at least +20%)
- Significantly improve the performances of our default rules set
- Our readme file is now shinier
- Minor code simplification

Bug fixes

- Fix a crash related to variadic functions

1.5.17 0.2.1 - Elephant Point 2018/02/07

Bug fixes

- The testsuite can now be successfully run as root
- Fix a double execution when snuffleupagus is used with some other extensions
- Fix an execution-context related crash

Improvements

- Support PCRE2, since it's required for PHP7.3
- Improve a bit the portability of the code
- Minor code simplification

1.5.18 0.2.0 - Elephant Rally - 2018/01/18

New features

- Glob support in `sp.configuration_file`
- Whitelist/blacklist functions in `eval`
- `phpinfo` shows if the configuration is valid or not

Bug fixes

- Off-by-one in configuration parsing fixed
- Minor cookie-encryption related memory leaks fixes
- Various crashes spotted by fr33tux fixes
- Configuration files with windows EOL are correctly handled

Improvements

- General code clean-up
- Documentation overhaul
- Compilation on FreeBSD and CentOS
- Select which cookies to encrypt via regular expressions
- Match on return values from user-defined functions

External contributions

- Simplification and clean up of our linked-list implementation by smagnin

1.5.19 0.1.0 - Mighty Mammoth - 2017/12/21

- Initial release

1.6 FAQ

1.6.1 General

What is Snuffleupagus?

Snuffleupagus is a PHP7+ module designed to drastically raise the cost of attacks against websites. This is achieved by killing entire bug classes and providing a powerful virtual-patching system, allowing the administrator to fix specific vulnerabilities without having to touch the PHP code.

Where does the name *Snuffleupagus* come from?

> Aloysius Snuffleupagus, more commonly known as Mr. Snuffleupagus, or Snuffy for short, is one of the characters on Sesame Street, the educational television program for young children.
>
> He was created as a woolly mammoth without tusks or (visible) ears, and has a long thick pointed tail, similar in shape to that of a dinosaur or other reptile. He has long thick brown hair and a trunk, or "snuffle", that drags along the ground. He is Big Bird's best friend and has a baby sister named Alice. He also attends "Snufflegarten".
>
> —Wikipedia

Why is Snuffleupagus called Snuffleupagus?

Like PHP's ElePHPant, we thought that using an elephant as a mascot would be a great idea.

Who are you and why did you write Snuffleupagus?

The project started at NBS System, a web hosting company (meaning that we're dealing with PHP code all day long), with a strong focus on security. We do have several layers of hardening (kernel, WAF, IDS, etc), but we had nothing for PHP7.

Nowadays, Snuffleupagus is maintained by Julien (jvoisin) Voisin.

Why not Suhosin?

We're huge fans of Suhosin, unfortunately:

- it doesn't work very well on PHP7
- it has some oudated features and misses new ones
- it doesn't cope very well with our various industrialization needs
- it has some shortcomings by design

We're using the disable_function directive, but unfortunately, it doesn't provide enough usable granularity (guess how many CMSs are using the system function to perform various mandatory maintenance tasks).

This is why we decided to write our own hardening module, in the spirit of Suhosin, with virtual-patching support, as well as other cool new features.

What license is Snuffleupagus released under and why?

Snuffleupagus is licensed under the LGPL was developed by the fine people from NBS System, and is maintained by Julien (jvoisin) Voisin.

We chose the LGPL because we don't care that much how you're using Snuffleupagus, but we'd like to force people to make their improvements/contributions available to everyone.

The complete license text is shipped with the sources and can be found under `LICENSE`.

For compatibility with older PHP versions, some original PHP source code was copied or ported back to older versions. This source code resides in `src/sp_php_compat.c` and `src/sp_php_compat.h` and retains its original license The PHP License, version 3.01, also included with the sources as `PHP_LICENSE`.

What is the different between SNuffleupaugs and a (WAF) like ModSecurity?

ModSecurity and the other Web Application Firewall (WAF) are working by inspecting the http traffic. Snuffleupagus being a PHP module, is operating directly inside your website's code, with a lesser overhead, as well as a better understanding of what is currently happening inside your application.

Should I use Snuffleupagus?

Yes.

Even if you're not using the virtual-patching capabilities, Snuffleupagus comes with various passive features that won't break your website while killing numerous vulnerabilities.

Please keep in mind that you are not only protecting yourself and your users/customers, but also other people on the internet that might be attacked by your server if it becomes compromised.

How mature is this project?

This project has been floating around since early 2016 and we did the first commit the 28 of December of the same year. It's currently stable, and is usable and used in production.

Are you saying that PHP isn't secure?

We don't like PHP's approach of security; namely (sometimes) adding warnings in the documentation and trusting the developer to not do any mistake, instead of focusing on the root cause and killing the bug class once and for all.

Moreover, it seems that the current attitude toward security in the PHP world is to blame the user instead of acknowledging issues, as stated in their documentation. We do think that a security issue that "requires the use of code or settings known to be insecure" is still a security issue, and should be treated as such.

We don't have the pretension to state that Snuffleupagus will magically solve all your security issues, but we believe that it might definitely help.

Sounds great, but is it working?

We've been using it in production since a couple of years, and it thwarted numerous known and unknown attacks. If you want some evidences, one of the developer published in June 2019 a blogpost showcasing how efficient Snuffleupagus was versus *major* web vulnerabilities from 2018/2019.

Why should I send you bugs, security issues and patches?

Snuffleupagus is an open-source security software, by reporting (or fixing) bugs, or implementing new features, you are helping others to protect themselves.

We're also firm believers in the *Beerbounty* system: we are happy to offer you beers when/if we ever meet if you helped the project in any way. If you don't like beer, we're sure that we'll find something else, don't worry.

1.6.2 Installation and configuration

Can snuffleupagus break my application?

Yes.

Some options won't break anything, like *harden-rand*, but some like *global_strict* or overly-restrictive *virtual-patching* rules might pretty well break your website. It's up to you to configure Snuffleupagus accordingly to your needs.

You can also enable the `simulation` mode on features that you're not sure about, to see what snuffleupagus would do to your application, before activating them for good.

How can I find out the problem when my application breaks?

By checking the logs; Snuffleupagus systematically prefix them with `[snuffleupagus]`.

Does Snuffleupagus run on Windows?

No idea, feel free to try.

Does Snuggleupagus run on HHVM?

No it doesn't, since HHVM's API is really different from PHP7's one. We're not currently planning to rewrite Snuffleupagus to support it.

Will Snuffleupagus run on my old PHP 5?

No.

Since PHP5 is deprecated since the end of 2018, you should think about moving to PHP7. You can (and should) use Suhosin in the meantime.

1.6.3 Help and support

I found a security issue

If you believe you have found a security issue affecting Snuffleupagus, then we would be more than happy to hear from you!

We promise to treat any reported issue seriously and, if the investigation confirms it affects Snuffleupagus, to patch it within a reasonable time, release a public announcement that describes the issue, discuss potential impact of the vulnerability, reference applicable patches or workarounds, and credit the discoverer.

Please do send a mail to [Julien (jvoisin) Voisin](https://dustri.org) should you find a security issue.

I found a bug. How can I report it?

We do have an issue tracker on Github. Please make sure to include as much information as possible when reporting your issue, such as your operating system, your version of PHP 7, your version of Snuffleupagus, your logs, the problematic php code, the request, a brief description, ... long story short, give us everything that you can.

If you're feeling extra-nice, you can try to debug it yourself, it's not that hard.

Where can I find even more help?

The *configuration page* might be what you're looking for. If you're adventurous, you can also check the issue tracker (make sure to check the closed issues too).

1.6.4 Unimplemented mitigations and abandoned ideas

Constant time comparisons

We didn't manage to perform time-based side-channel attacks on strings against real world PHP application, and the results that we gathered on tailored test cases weren't concluding: for simplicity's sake, we chose to not implement a mitigation against this class of attacks.

We would be happy to be proven wrong, and reconsider implementing this feature, if someone can manage to get better results than us.

The possibility of having this natively in PHP has been discussed, but as 2017, nothing has been merged yet.

Nop'ing function execution

Snuffleupagus can be configured to either *allow* or *drop* the execution of particular functions and optionally *log* and *dump* them, but it doesn't provide any mechanism to *nop* their execution.

We thought about adding this, but didn't for several reasons:

- What should the return value of a *nop'ed* function be?
- It would add confusion between `drop`, `nop` and `log`.
- Usually, when a specific function is called, either it's a dangerous one and you want to stop the execution immediately, or you want to let it continue and log it. There isn't really any middle-ground, or at least we failed to find any.

1.7 Propaganda

This pages lists various mentions, articles, usages and presentations about Snuffleupagus.

1.7.1 Talks

2017

- BerlinSide0x08 - slides
- Hack.lu 2017 - slides - video
- BlackAlps - slides - video

2018

- Pass the Salt - slides - video
- 44con - slides

2020

- Modern PHP security - sec4dev 2020, Vienna - Synacktiv - sec4dev 2020

2022

- Custom php Introspection for 0-Day Research - GreHack 2022, Grenoble - Groumpf and Laluka - transcript and blogpost

2023

- S01-E35-FR | Spip email/eval n-day - Analysis with Snuffleupagus, with @olivier_boschko (fr)

1.7.2 Mentions

2017

- Habr - PHP- № 118 – , (ru) - Habr
- Intrinsec's blog - Hack.lu 2017 (fr) - Intrinsec's blog
- Paragon Initiative Enterprises Blog - The 2018 Guide to Building Secure PHP Software

2018

- Habr - PHP- № 138 (ru) - Habr
- PhpStorm's blog - PHP Annotated Monthly - PhpStorm's blog
- PHP Weekly
- New variant in wp-gdpr-compliance vulnerability and fixing it with virtual patching - alertot

2019

- PhpStorm's blog - PHP Annotated - PhpStorm's blog
- Habr - PHP- № 160 (ru) - Habr

2020

- Modern PHP Security Part 2: Breaching and hardening the PHP engine - Detectify's blog

2021

- Habr - PHP № 196 (ru) - Habr
- OWASP's PHP Configuration Cheat Sheet - OWASP

2022

- RCE on Spip and Root-Me, v2! - Laluka's blog

1.7.3 Articles

2017

- Killing php bug classes at berlinsides - dustri.org
- Snuffleu…what? - fr33tux.org
- Behold the Snuffleupagus - memze.ro
- How to harden AdwCleaner's web backend using PHP - Malwarebyte's blog
- First release of Snuffleupagus - dustri.org
- PHP Magazine - phpmagazine.net

2018

- Snuffleupagus 0.3.0 - Dentalium elephantinum - dustri.org
- Snuffleupagus version 0.3.0 - Dentalium elephantinum (fr) - LinuxFr

2019

- Snuffleupagus PHP- (ru) - opennet.ru
- What the f*ck is a Snuffleupagus? - Living The Dream
- Snuffleupagus: Open source security tool hardens PHP sites against cyber-attacks - The Daily Swig
- Snuffleupagus, an excellent module to block vulnerabilities in PHP applications - linuxadictos.com
- Snuffleupagus versus recent high-profile vulnerabilities - dustri.org

2020

- Snuffleupagus, un excelente módulo para bloquear vulnerabilidades en aplicaciones PHP (es) - linuxadictos.com
- Snuffleupagus 0.5.1, PHP- (ru) - opennet.ru
- Snuffleupagus versus recent high-profile vulnerabilities, again! - dustri.org
- Snuffleupagus, módulo para bloquear vulnerabilidades en aplicaciones PHP - (es) - underc0de.org

2021

- Sortie de Snuffleupagus 0.7.0 - Los Elefantes (fr) - linuxfr
- Virtual patching CVE-2021-29447 with Snuffleupagus - dustri.org

2022

- Lightweight post-exploitation hardening in PHP via call-site freezing and ghetto-CFI with Snuffleupagus - dustri.org
- Increasing PHP security with Snuffleupagus - blog.frehi.be

1.7.4 Papers

- Sécurisez vos applications php avec Snuffleupagus (fr) (paywall) - 2018-03-2018

1.7.5 Notable users

- AdwCleaner's backend- a notorious anti-pup
- Alertot - a Chilean continuous web security monitoring company
- Control Web Panel - a free modern and intuitive control panel for servers and VPS
- Mailu - mail server as Docker images
- Mangadex - a major manga website
- NBS System - a French hosting/security company and author of snuffleupagus
- Net4All - a Swiss hosting company
- Oceanet Technology - a French hosting company
- The Swedish team of the NATO's CCDCOE Locked Shields exercise, winner of the 2021 and 2023 editions.
- SwissCenter - a Swiss datacenter & web hosting company
- Toolslib - an Alexa top 10k website
- cPanel - one of the most popular web hosting control panel

1.8 Cookies

Some cookies-related features might prevent other extensions from hooking the setcookie function. Pay attention to the loading order of your extensions in this case.

1.8.1 auto_cookie_secure

auto_cookie_secure, disabled by default, will automatically mark cookies as secure when the web page is requested over HTTPS.

It can either be `enabled` or `disabled`.

```
sp.auto_cookie_secure.enable();
sp.auto_cookie_secure.disable();
```

1.8.2 cookie_samesite

samesite, disabled by default, adds the samesite attribute to cookies. It prevents CSRF but is not implemented by all web browsers yet. Note that this is orthogonal to PHP7.3+ SameSite support.

It can either be set to `strict` or `lax`:

- The `lax` attribute prevents cookies from being sent cross-domain for "dangerous" methods, like `POST`, `PUT` or `DELETE`.
- The `strict` one prevents any cookies from beind sent cross-domain.

```
sp.cookie.name("cookie1").samesite("lax");
sp.cookie.name_r("^cookie[0-9]+").samesite("lax");
sp.cookie.name("cookie2").samesite("strict");;
```

1.8.3 Cookie encryption

The encryption is done via the tweetnacl library, thus using curve25519, xsalsa20 and poly1305 for the encryption. We chose this library because of its portability, license (public-domain), simplicity and reduced size (a single *.h* and *.c* file.).

The key is derived from multiple sources, such as:

- The `secret_key` provided in the configuration in the `sp.global.secret_key` option. It's recommended to use something like `head -c 256 /dev/urandom | tr -dc 'a-zA-Z0-9'` as a value.
- An optional environment variable, such as `REMOTE_ADDR` (remote IP address) or the *extended master secret* from TLS connections (RFC7627) in the `sp.global.cookie_env_var` option.
- The user-agent.

Warning: To use this feature, you **must** set the *global.secret_key* variable and **should** set the *global.cookie_env_var* one too. This design decision prevents an attacker from trivially bruteforcing or re-using session cookies. If the simulation mode isn't specified in the configuration, snuffleupagus will drop any request that it was unable to decrypt.

Since PHP doesn't handle session cookie and non-session cookie in the same way, so does Snuffleupagus.

Session cookie

For the session cookie, the encryption happens server-side: Nothing is encrypted in the cookie: neither the cookie's name (usually `PHPSESSID`) nor its content (the session's name). What is in fact encrypted, is the session's content, on the server (usually stored in `/tmp/sess_<XXXX>` files).

Session encryption, disabled by default, will activate transparent session encryption. It can either be `enabled` or `disabled` and can be used in `simulation` mode.

```
sp.session.encrypt();
sp.session.simulation();
```

Non-session cookie

For the non-session cookie, the cookie's name is left untouched, only its value is encrypted.

Cookie encryption, disabled by default, will activate transparent encryption of specific cookies.

It can either be `enabled` or `disabled` and can be used in `simulation` mode.

```
sp.cookie.name("my_cookie_name").encrypt();
sp.cookie.name("another_cookie_name").encrypt();
```

Removing the user-agent part

Some web browser extensions, such as uMatrix might be configured to change the user-agent on a regular basis. If you think that some of your users might be using configurations like this, you might want to disable the mixing of the user-agent in the cookie's encryption key. The simplest way to do so is to set the environment variable `HTTP_USER_AGENT` to a fixed value before passing it to your php process.

We think that this use case is too exotic to be worth implementing as a proper configuration directive.

Choosing the proper environment variable

It's up to you to choose a meaningful environment variable to derive the key from. Suhosin is using the `REMOTE_ADDR` one, tying the validity of the cookie to the IP address of the user; unfortunately, nowadays, people are roaming a lot on their smartphone, hopping from WiFi to 4G.

This is why we recommend, if possible, to use the *extended master secret* from TLS connections (RFC7627) instead. The will make the validity of the cookie TLS-dependent, by using the `SSL_SESSION_ID` variable.

- In Apache, it is possible to enable by adding `SSLOptions StdEnvVars` in your Apache2 configuration.
- In nginx, you have to use `fastcgi_param SSL_SESSION_ID $ssl_session_id if_not_empty;`.

If you aren't using TLS (you should be), you can always use the `REMOTE_ADDR` one, or `X-Real-IP` if you're behind a reverse proxy.

CHAPTER

TWO

GREETINGS

We would like to thank the following people:

- Suhosin, for paving the way.
- Hardened PHP, for *everything* they did, especially the Month of PHP Security.
- The people behind the RIPS scanner, for their ground breaking work
- NBS System, for creating and open-sourcing this piece of software
- Websec.fr, for keeping our interesting vulnerabilities alive
- Web developers around the world, for being so imaginative