» Snuffleupagus

An elephant with some salt,
in your php stack,
killing bug classes,
and virtual-patching,
what is remaining.

» Disclaimer

We gave subsets of this talks at other conferences,

you might experience a déjà vu feeling[1] .

1· But fear not, we added a bunch of cool new stuff!

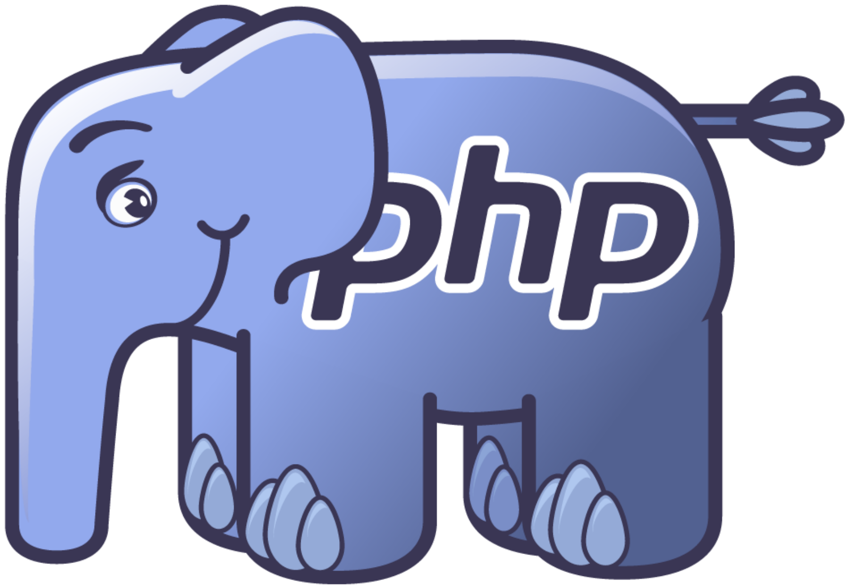
» Bonjour

» Bonjour

- We're super happy to be here
- We're both French[1], and are working together
- In the security team of a company called ***NBS System***
- It's a hosting company, for websites and stuff
- You might also know it as "the cloud"

1. Enjoy our frenglish.

» What are we trying to fix?



Reducing the ***ratio of shell/day*** happening on PHP7+ websites on the internet

» PHP in a nutshell

Fig 1. The security team casually reading some php code

» More seriously

- We're hosting a ***lot*** of websites, most of them written in PHP.
- PHP is known to be an "interesting" language[1] and some of its users are highly "creative".

How can we prevent our customers (and people on the web) to get pwned on a daily basis?

1. Also known as a ***trigger-happy multi-barrel footgun***

» What we currently have

- We've got a dedicated security team
- We've got kick-ass OS-level hardening (grsecurity ♥)
- We've got a pile of custom IDS machinery
- We've got a fancy (and open sauce) WAF called ***naxsi***

» What we currently have

- We've got a dedicated security team
- We've got kick-ass OS-level hardening (grsecurity ♥)
- We've got a pile of custom IDS machinery
- We've got a fancy (and open sauce) WAF called ***naxsi***

But some vulnerabilities are still not patchable without touching the PHP code, but we don't want to, even with a 6 meter[1] pole

1· Metric system is the only valid unit system.

» Can't we harden PHP itself?

- ***Suhosin*** did it, it worked great, but we're in 2018 now:
    - It has super-cool features
    - It lacks some fancy ones
    - It's painful to industrialize
    - It's on life-support
    - It doesn't fly on PHP7+

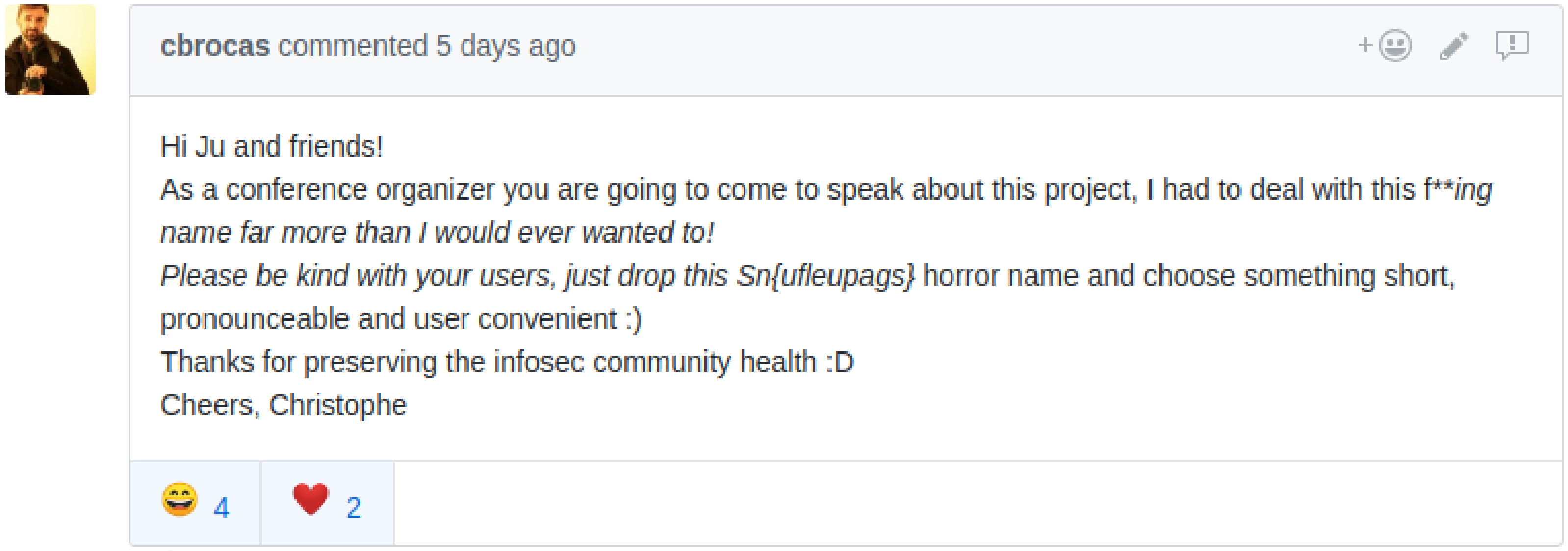
» Here comes NIH syndrom!

Fig 1. Us, ready to conquer the world with our new project!

» So we wrote our own hardening module, in C!

Fig 1. The magnificent Snuffleupagus

» Snuffleu-what?



cbrocas commented 5 days ago

Hi Ju and friends!
As a conference organizer you are going to come to speak about this project, I had to deal with this f***ing name far more than I would ever wanted to!*
Please be kind with your users, just drop this Sn{ufleupags} horror name and choose something short, pronounceable and user convenient :)
Thanks for preserving the infosec community health :D
Cheers, Christophe

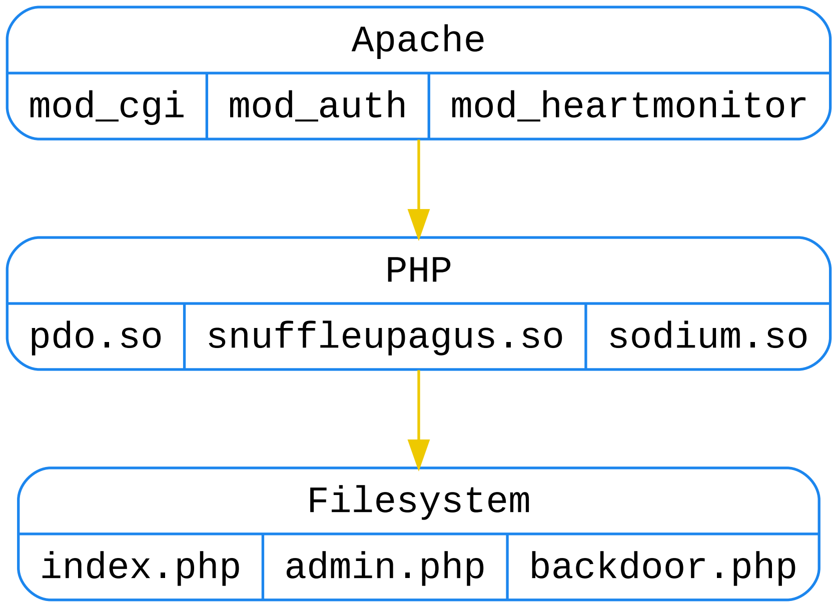
😄 4 ❤️ 2

» Snuffleupagus?!

> Aloysius Snuffleupagus, more commonly known as Mr. Snuffleupagus, Snuffleupagus or Snuffy for short, is one of the characters on ***Sesame Street***.
>
> 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.
>
> — wikipedia

» MAGNIFICENT §!1§!!!1§§

» Where does it live



» PHP-level virtual patching[1]

1· Also known as *Snuffleupatchgus*

» The issue with "vanilla" php hardening

- `disable_function` can globally forbid usage of arbitrary functions
- Your CMS is using `system` for its update mechanism
- Either forbid `system` or keep your website up to date
- This is why we can't have nice things.

» How we're helping

- Disable `system` globally:

```
sp.disable_functions.function("system").drop();
```

- Allows `system` calls in a specific file

```
sp.disable_functions.function("system").filename("up.php").allow();
sp.disable_functions.function("system").drop();
```

- Allow `system` calls in a file, with a matching sha256:

```
sp.disable_functions.function("system").filename("up.php").hash("13..a").allow();
sp.disable_functions.function("system").drop();
```

We even provide a **user-friendly** script to generate a configuration file, freezing dangerous functions usage.

» What can we do with php-level virtual-patching?

» About the syntax

We designed the rules syntax like this:

- 24 different filters
- Documentation for everything
- Lots of examples

to be able to easily patch:

- every ***wordpress*** CVE since 2010
- the ***RIPS advent calendar***
- a lot of ***high-profile*** web exploits
- our own 0dayz[1]

[1] Come to the workshop on Friday to see some of them ;)

» Examples

```
sp.disable_function("PHPThingy::MyClass::method_one>internal_func").drop();
sp.disable_function("admin_cron_thingy").cidr("127.0.0.1/32").allow();
sp.disable_function("admin_cron_thingy").drop();
sp.disable_function.function("render_tab3").var("_REQUEST[tab]").value_r("\"").drop();
sp.disable_function.function("system").pos("0").value_r("[^a-z]").drop();
```

» What can we do with this?

» `system()` injections

» What the documentation is saying

> 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***.

» What people are doing

```php
<?php
$ip_addr = system("dig +short " . $_GET["address"]);
echo "The ip adress of $_GET['address'] is $ip_addr";
?>
```

» What we're getting

- `CVE-2017-7692`: Authen RCE on SquirrelMail
- `CVE-2016-9565`: Unauth RCE on Nagios Core
- `CVE-2014-1610`: Unauth RCE on DokuWiki
- ***Every single*** shitty modem/router/switch/IoT.

» How we're (kinda) killing it

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

» `mail` related RCE

» What the documentation is saying

> 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

Known since 2011, popularized by RIPS.

» What people are doing

```
// Olol, sending some emails
mail(..., $_GET['a']);
```

» What we're getting

- `CVE-2017-7692`: Authen RCE in SquirrelMail
- `CVE-2016-10074`: RCE in SwiftMailer
- `CVE-2016-10033`: RCE in PHPMailer
- `CVE-2016-9920`: Unauth RCE in Roundcube
- RCE in a lot of webmails

» How we're (kinda) killing it

```
sp.disable_function.function("mail").param("additional_parameters").value_r("\-").drop();
```

» Writing rules

Fig 1. The security team realising that it needs to write a lot of rules.

» Nobody has time to write rules

So lets kill some bug classes!

» Session-cookie stealing via XSS

Like suhosin, we're encrypting cookies with a secret key tied to:

- The ***user-agent*** of the client
- A ***static key***
- And ***environnment variable*** that you can set to:
    - The ***ip address***[1]
    - The ***TLS extended master key***
    - …

[1] Not the best idea ever: in 2017, people are roaming ***a lot***.

» Misc cookies things

- If you're coming over https, your cookies get the `secure` flag
- If cookies are encrypted, they are `httpOnly`
- Support for `SameSite` to kill CSRF

» RCE via file-upload

» What the documentation is saying

> Not validating which file you operate on may mean that users can access ***sensitive information*** in other directories.

» What people are doing

```
$uploaddir = '/var/www/uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)
```

» What we're getting

- `CVE-2001-1032` : RCE in PHP-Nuke via file-upload
- ...
- *15 years later*
- ...
- `CVE-2016-9187` : RCE in Moodle via file-upload

> `There are 850 CVE entries that match your search`
>
> — cve.mitre.org

» How we're killing it

Suhosin style:

```
sp.upload_validation.script("tests/upload_validation.sh").enable();
```

One trick is to rely on `vld`[1] to ensure file doesn't contain php code:

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

[1] Vulcan Logic Disassembler. (yes)

» Unserialize

» What the documentation is saying

> ***Do not*** pass untrusted user input to `unserialize()` [...]. Unserialization can result in code being loaded and executed [...].

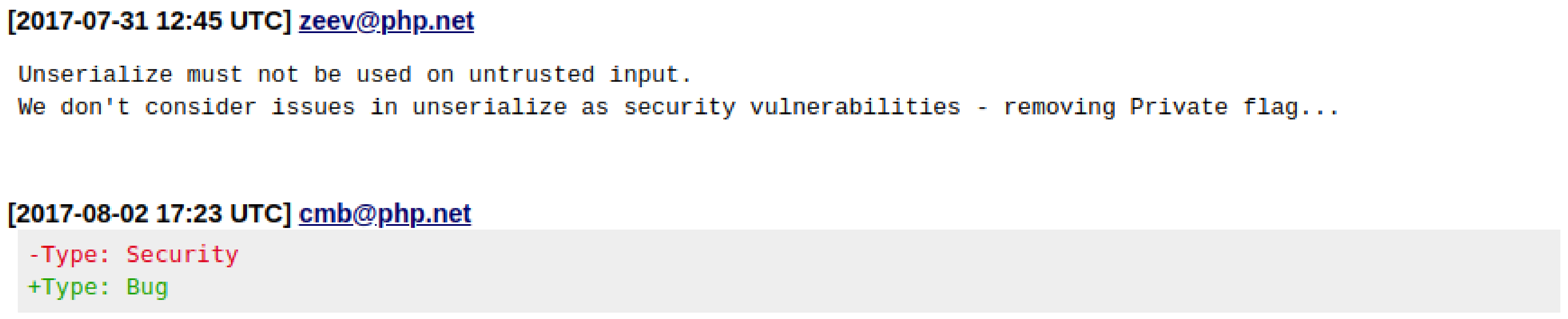
» What people are doing

```
$my_object = unserialize($_GET['o']);
```

» PHP annecdote

Fig 1. Rant about PHP in 3… 2… 1…

» Memory corruptions are *not* security issues


[2017-07-31 12:45 UTC] zeev@php.net

```
Unserialize must not be used on untrusted input.
We don't consider issues in unserialize as security vulnerabilities - removing Private flag...
```

[2017-08-02 17:23 UTC] cmb@php.net

```
-Type: Security
+Type: Bug
```

Fig 1. In PHP's world, unsanitized outputs are out of scope

» What we're getting

- `CVE-2012-5692`: unauth RCE in IP.Board
- `CVE-2014-1691`: Unauth RCE in Horde
- `CVE-2015-7808`: Unauth RCE in vBulletin
- `CVE-2015-8562`: Unauth RCE in Joomla
- `CVE-2016-????`: Unauth RCE in Observium (leading to remote root)
- `CVE-2016-5726`: Unauth RCE in Simple Machines Forums
- `CVE-2016-4010`: Unauth RCE in Magento
- `CVE-2017-2641`: Unauth RCE in Moodle

» How we're killing it

Php will discard any garbage found at the end of a serialized object: we're simply appending a ***hmac*** at the end of strings generated by `serialize`.

It looks like this:

```
s:1:"a";650609b417904d0d9bbf1fc44a975d13ecdf6b02b715c1a06271fb3b673f25b1
```

» `rand` and its friends

» What the documentation is saying

> This function ***does not*** generate cryptographically secure values, and ***should not*** be used for cryptographic purposes.

» What people are doing

```
$password_reset_token = rand(1,9) . rand(1,9) . [...] . rand(1, 9);
```

» What we're getting

- `CVE-2008-4102`: Auth bypass in Joomla
- ...
- `CVE-2015-5267`: Auth bypass in Moodle
- Various captcha bypasses

» How we're killing it

We're simply replacing every call to `rand` and `mt_rand` with `random_int`.

» XXE

» What the documentation is saying

Not a single warning ;)

» What people are doing

```php
$xmlfile = file_get_contents('php://input');
$dom = new DOMDocument();
$dom->loadXML($xmlfile);
$data = simplexml_import_dom($dom);
```

» What we're getting

- `CVE-2011-4107`: Authen LFI in PHPMyAdmin
- ...
- `CVE-2015-5161`: Unauth arbitrary file reading on Magento

» How we're killing it

We're calling `libxml_disable_entity_loader(true)` at startup, and ***nop'ing*** its call.

» Stream wrappers

» What the documentation is saying

> PHP comes with many built-in wrappers for various URL-style protocols for use with the filesystem functions such as `fopen()`, `copy()`, `file_exists()` and filesize().

Wrappers like: `file://`, `http://`, `ftp://`, `php://`, `zlib://`, `data://`, `glob://`, `phar://`, `ssh2://`, `rar://`, `ogg://`, `expect://`, ...

» What we're getting

- Various exfiltration means
- Memory corruptions for everyone
- RCE via `phar://` upon file access
- Zip bombs
- Whitelist bypasses via `zip://`
- You name it

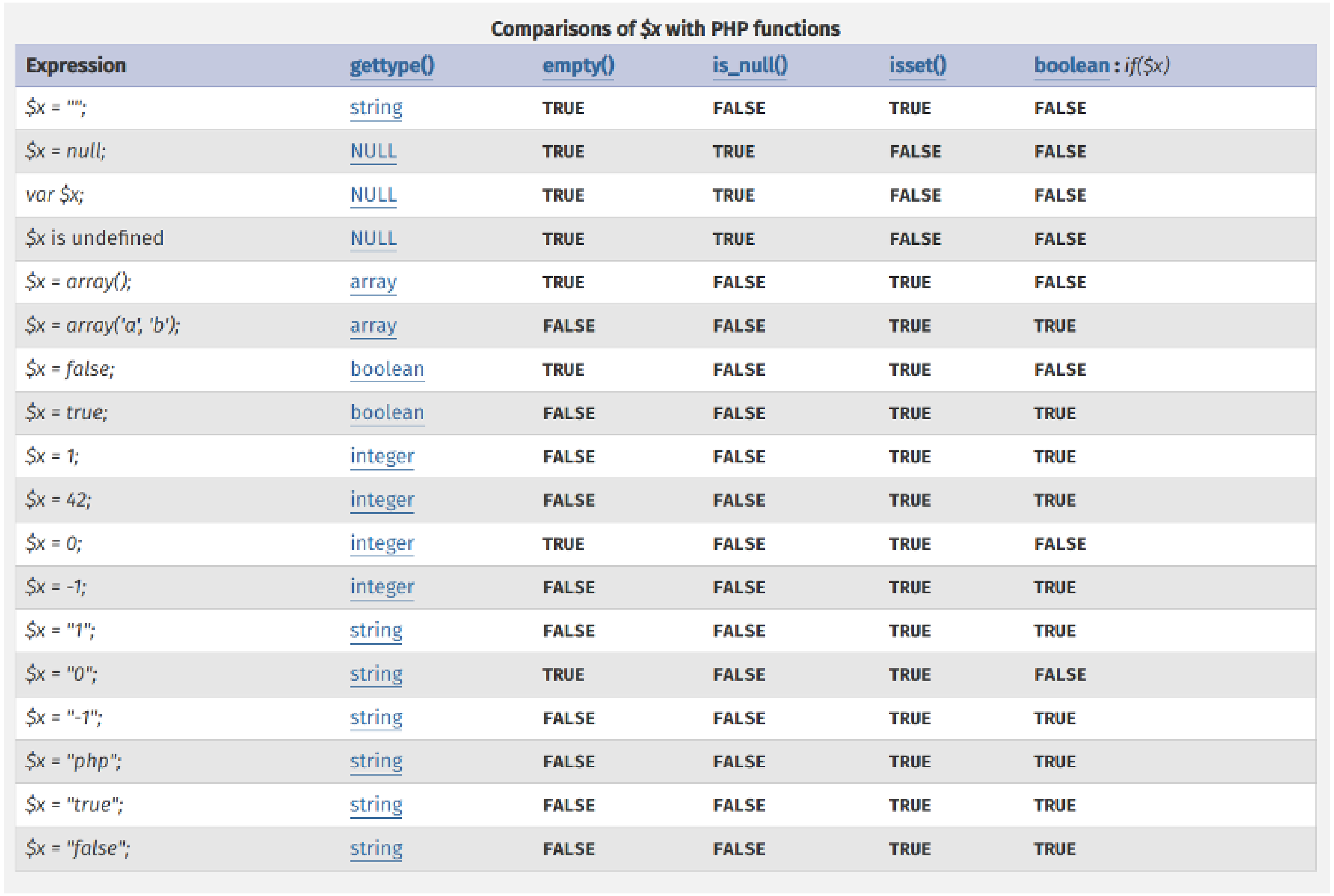
» How we're killing it

With a simple whitelist:

```
sp.wrapper_whitelist("file,php");
```

» "Smart" comparisons

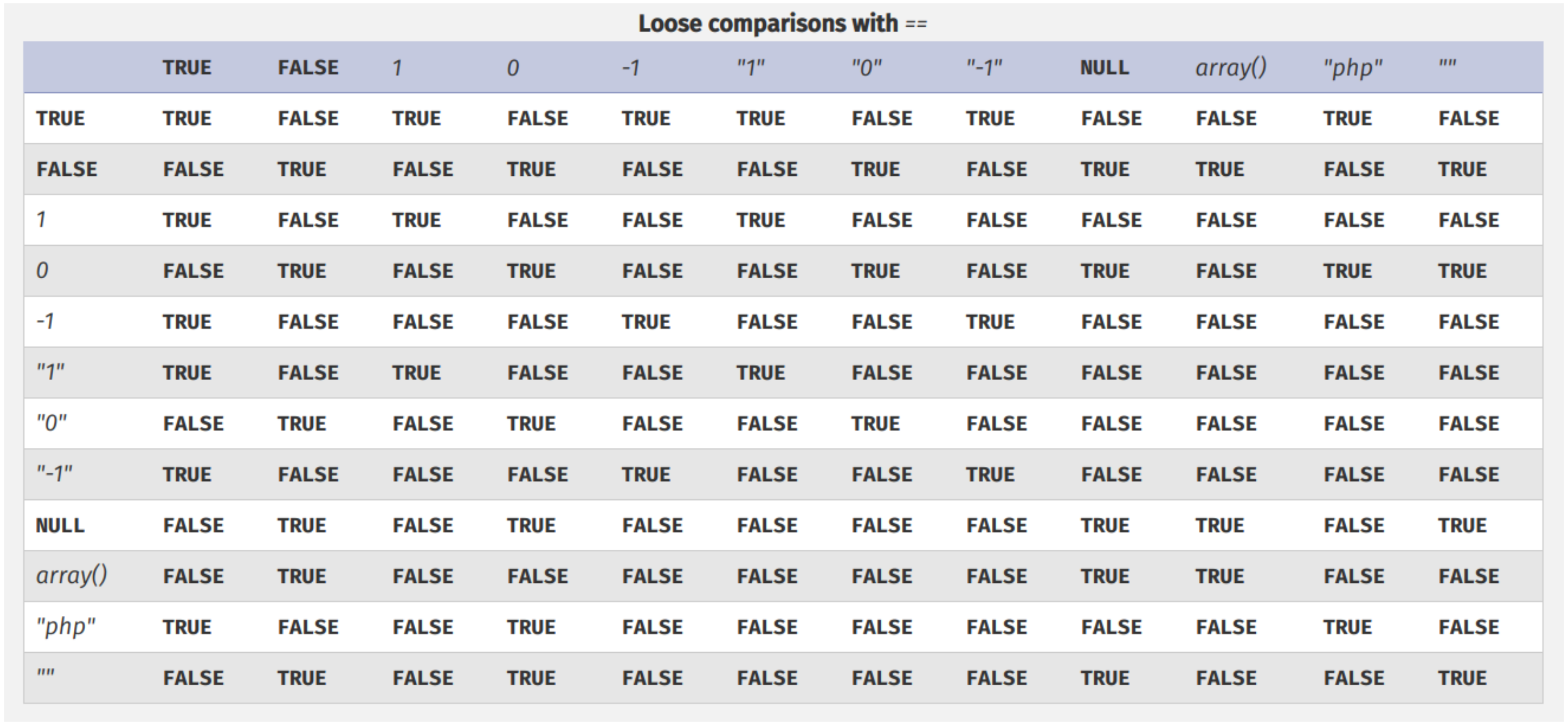
» What the documentation is saying



Comparisons of $x with PHP functions

Expression	gettype()	empty()	is_null()	isset()	boolean : if($x)
$x = "";	string	TRUE	FALSE	TRUE	FALSE
$x = null;	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x is undefined	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	array	TRUE	FALSE	TRUE	FALSE
$x = array('a', 'b');	array	FALSE	FALSE	TRUE	TRUE
$x = false;	boolean	TRUE	FALSE	TRUE	FALSE
$x = true;	boolean	FALSE	FALSE	TRUE	TRUE
$x = 1;	integer	FALSE	FALSE	TRUE	TRUE
$x = 42;	integer	FALSE	FALSE	TRUE	TRUE
$x = 0;	integer	TRUE	FALSE	TRUE	FALSE
$x = -1;	integer	FALSE	FALSE	TRUE	TRUE
$x = "1";	string	FALSE	FALSE	TRUE	TRUE
$x = "0";	string	TRUE	FALSE	TRUE	FALSE
$x = "-1";	string	FALSE	FALSE	TRUE	TRUE
$x = "php";	string	FALSE	FALSE	TRUE	TRUE
$x = "true";	string	FALSE	FALSE	TRUE	TRUE
$x = "false";	string	FALSE	FALSE	TRUE	TRUE

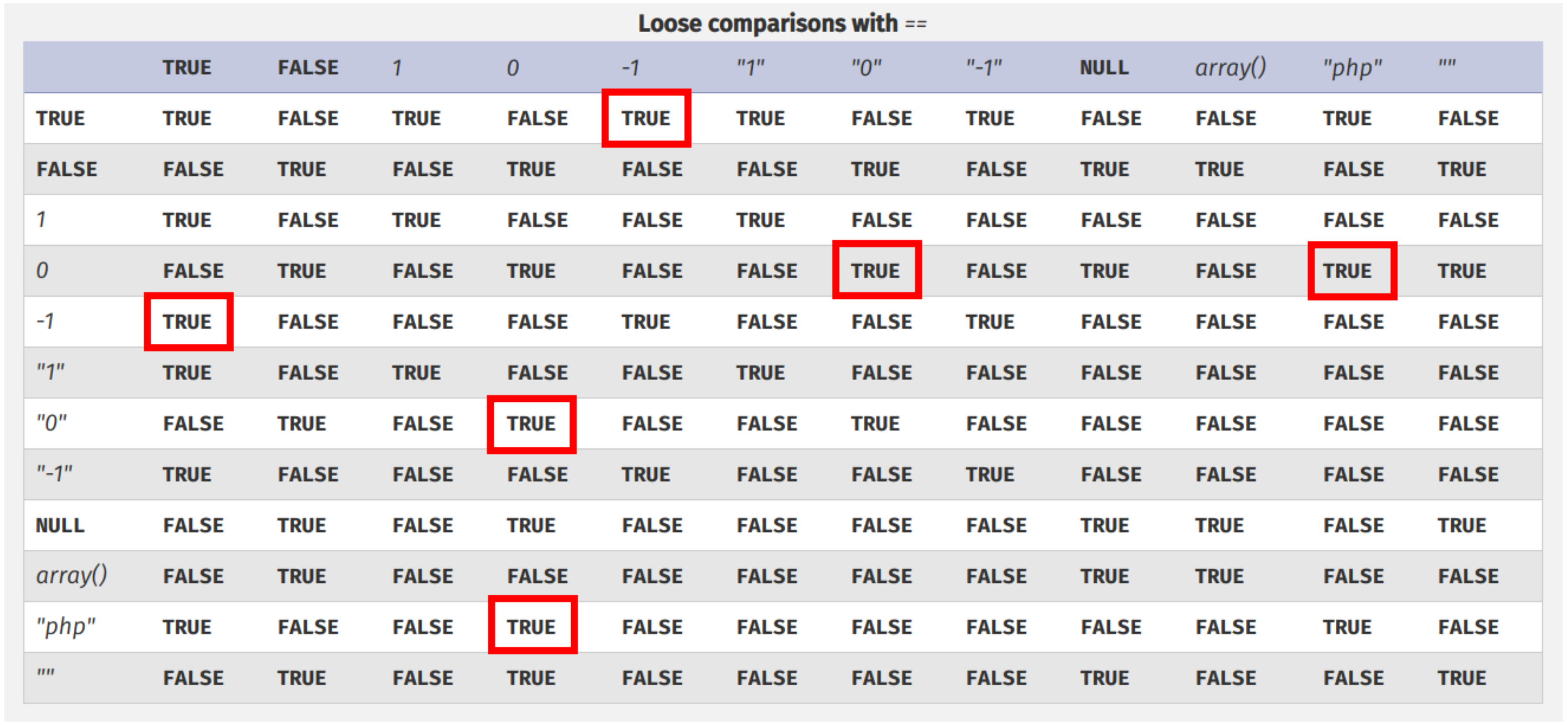
» What the documentation is saying (cont.)



Loose comparisons with ==

	TRUE	FALSE	*1*	*0*	*-1*	*"1"*	*"0"*	*"-1"*	NULL	*array()*	*"php"*	*""*
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

» What the documentation is saying (cont.)



Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

» What people are doing

Doing comparisons like PHP is a "normal" language, with things like:

- `if ($a == $_GET['password'])`
- `array_search($a, $my_array)`
- `in_array($a, $my_array)`
- `$val = $a?"yay":"nay";`
- `sha1('aaroZmOk') != sha1('aaK1STfY')`
- `'0010e2' != '1e3'`

» What we're getting

Launch `grep -Rn '[^=]==[^=]'` in any php application, and be "amazed"

- Password comparison
- CSRF tokens
- Password reset
- User id
- Currencies amounts comparison
- Every single comparison of data

» How we're killing it

- Global `strict` mode taking advantage of type annotation
- Silently replacing `==` with `===`

» Unrelated misc things

```
# chmod hardening
sp.disable_function.function("chmod").param("mode").value_r("7$");
sp.disable_function.function("chmod").param("mode").value_r("o\+w");

# backdoors detection
sp.disable_function.function("ini_get").param("var_name").value("open_basedir");
sp.disable_function.function("is_callable").param("var").value("system");

# prevent execution of writeable files
sp.readonly_exec.enable();

# Ghetto sqli detection
sp.disable_functions.function_r("mysqli?_query").ret("FALSE").dump().allow();
sp.disable_functions.function_r("PDO::query").ret("FALSE").dump().allow();

# Ensure that certificates are properly verified
sp.disable_function.function("curl_setopt_array")
    .param("options[CURLOPT_SSL_VERIFYHOST]").value("0").drop();
sp.disable_function.function("curl_setopt_array")
    .param("options[CURLOPT_SSL_VERIFYPEER]").value("0").drop();
```

» Free 0dayz

Fig 1. The security team catching juicy vulnerabilities

» Harvesting 0days

If you've got something like this

```
$line = system("grep $var dict.txt");
```

You can do something like that

```
sp.disable_function.function("system").var("var").regexp("[;`&|\n]").dump().allow();
```

And wait until someone finds a vuln to collect a working exploit.

» Performance impact

- Currently deployed on (at least) one Alexa1 top 1k website.
- We're using it on some customers
- No performance impact noticed
- We're (kinda) only hooking the functions that you specify
- Filter-matching is written with performances in mind

» Speed!

Fig 1. A regular php stack with Snuffleupagus running at full speed.

» What's left to do

- Killing more bug-classes like SQLI[1]
- Provide more hardening features
- Improve the virtual patching capabilities
- Party party party
- Give a workshop Friday morning

[1] We're working on it ;)

» What workshop?

- We'll give a workshop Friday morning, about
  - Deploying Snuffleupagus
  - Patching some real-world[1] vulnerabilities
  - Discuss patching strategies and mitigations details
- Careful, the whole workshop will be held with a thick French accent.

1· And previously unknown

» Where can you get this wonder?

- https://github.com/nbs-system/snuffleupagus for the sauce code
- https://snuffleupagus.rtfd.io for the (amazing) documentation
- Come talk to us, we're friendly!
- Friday during the workshop

» Mandatory final quote

> There are only two kinds of languages: the ones people complain about and the ones nobody uses.
>
> — Bjarne Stroustrup

Did you know that more than ***¾ of the web*** is using PHP?

» Cheers

- The ***RIPS*** people for their awesome scanner
- ***SectionEins*** for Suhosin and inspiration
- The ***HardenedPHP*** project for leading the way
- ***websec.fr*** for showcasting our most convoluted exploits
- Our ~~guinea pigs~~ ***friends*** who alpha-tested everything
- Folks that ~~called us names~~ gave us constructive feedback
- 44con for accepting our talk ♥

» Questions?

EVERY
DAY I'M
SNUFFLIN'