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

Posted **Mar 24, 2026**

By **thinhnee**

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.

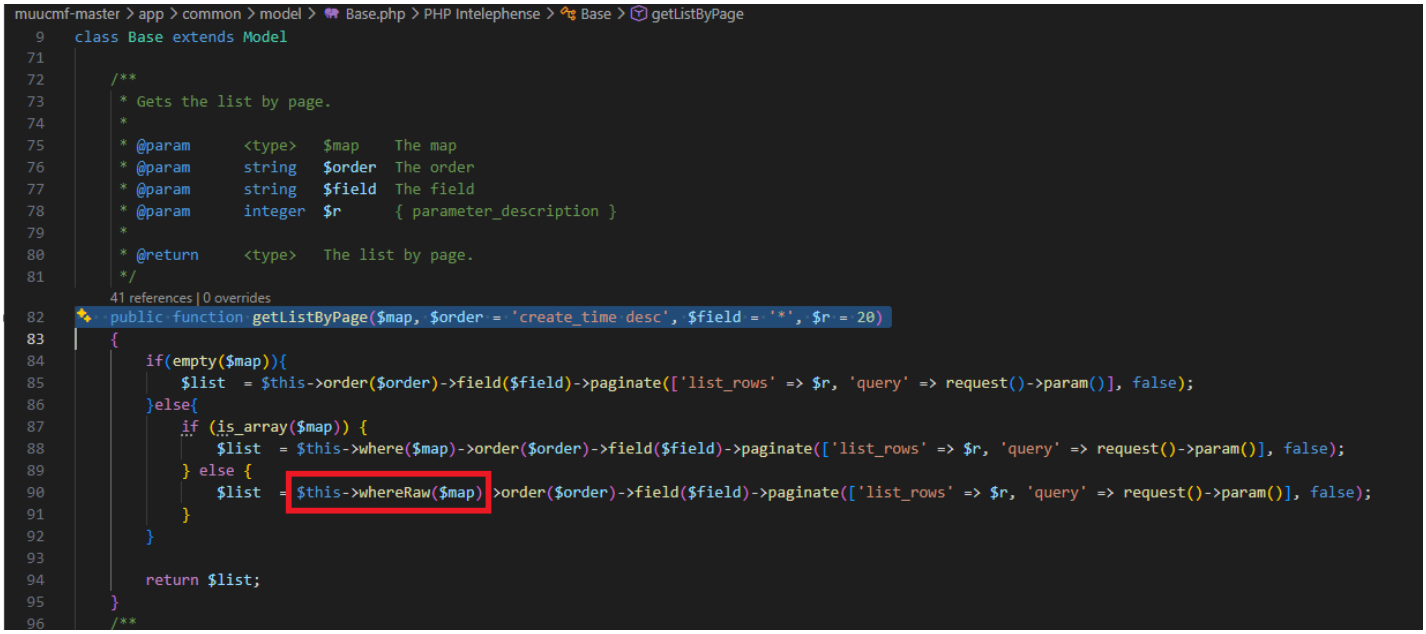
Vulnerability: SQL Injection

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. Ananlysis

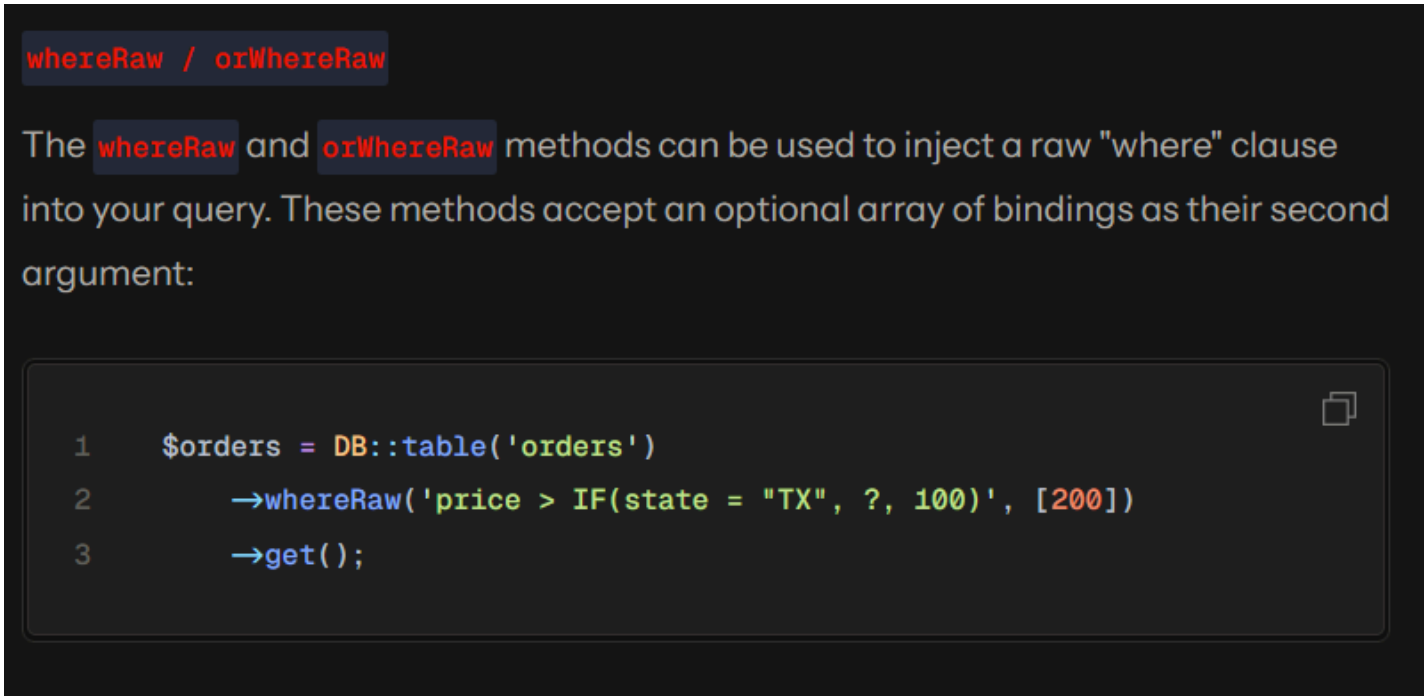
While review the source code, I found an interesting function. In a `muucmf-master\app\common\model\Base.php` I found that a `getListByPage()` function using `whereRaw` function.



muucmf-master > app > common > model > Base.php > PHP Intelephense > Base > getListByPage

```php
class Base extends Model

    /**
     * Gets the list by page.
     *
     * @param      <type>   $map    The map
     * @param      string   $order  The order
     * @param      string   $field  The field
     * @param      integer  $r      { parameter_description }
     *
     * @return     <type>   The list by page.
     */
    41 references | 0 overrides
    public function getListByPage($map, $order = 'create_time desc', $field = '*', $r = 20)
    {
        if(empty($map)){
            $list  = $this->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
        }else{
            if (is_array($map)) {
                $list  = $this->where($map)->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
            } else {
                $list  = $this->whereRaw($map)->order($order)->field($field)->paginate(['list_rows' => $r, 'query' => request()->param()], false);
            }
        }

        return $list;
    }
    /**
```

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



whereRaw / orWhereRaw

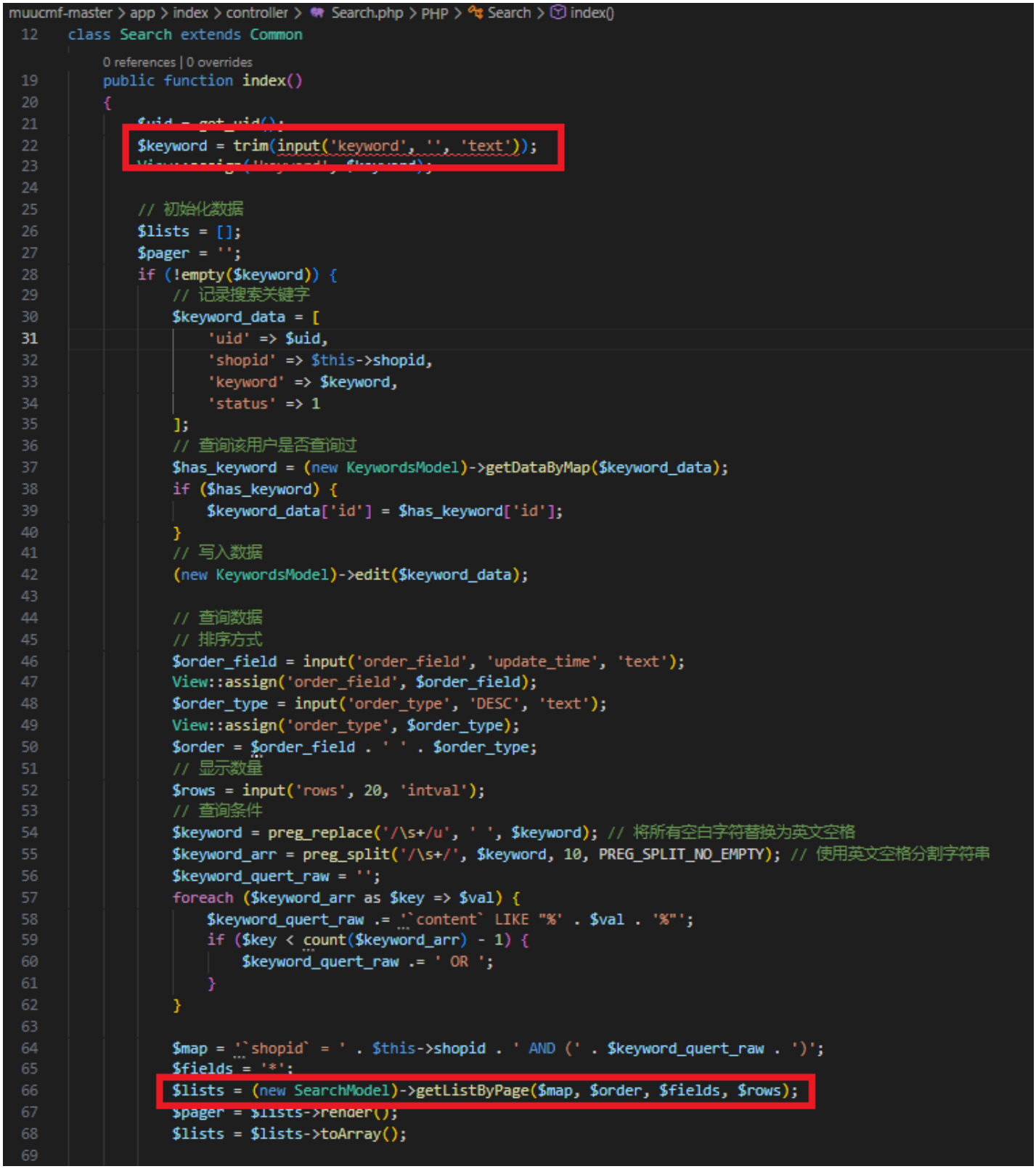
The `whereRaw` and `orWhereRaw` methods can be used to inject a raw "where" clause into your query. These methods accept an optional array of bindings as their second argument:

```php
$orders = DB::table('orders')
    ->whereRaw('price > IF(state = "TX", ?, 100)', [200])
    ->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. Coolllllll.

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()`.



PHP

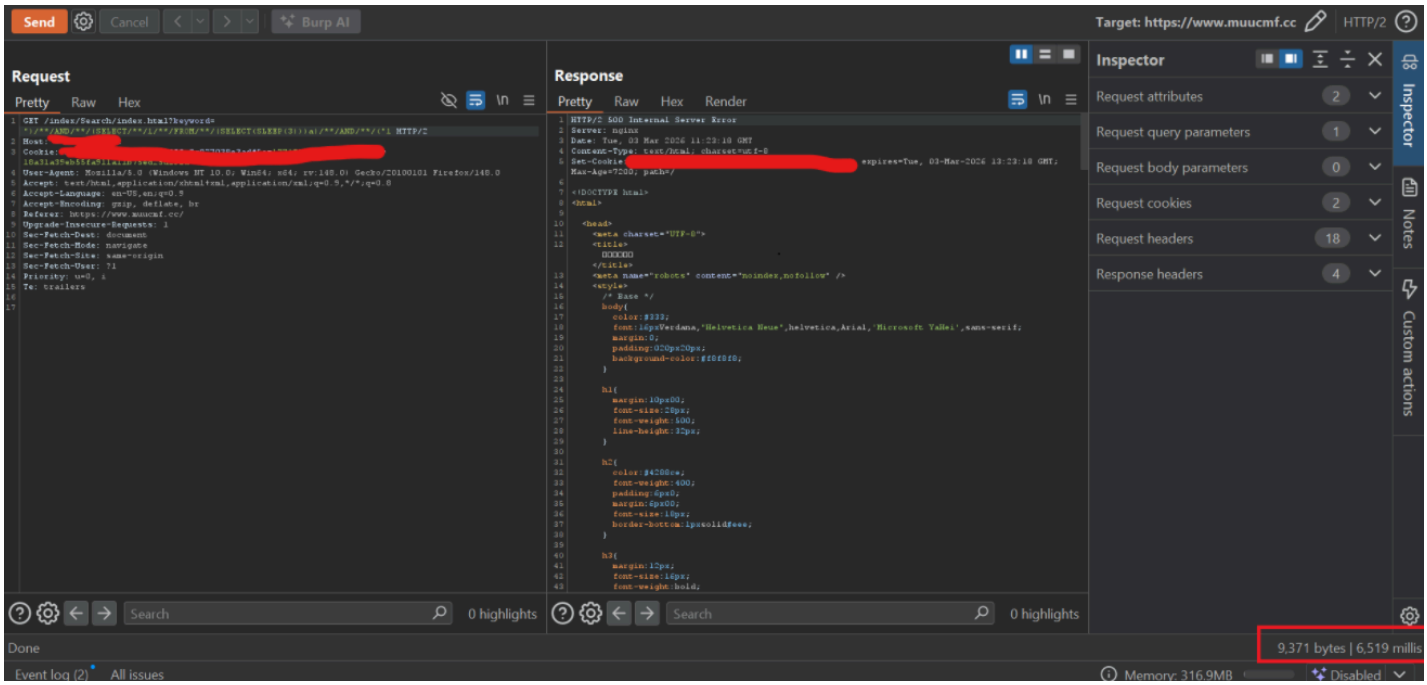
```php
        $uid = get_uid();
        $keyword = trim(input('keyword', '', 'text')); // let user enter the inp
        View::assign('keyword', $keyword);
        // ...bla bla...
        $keyword = preg_replace('/\s+/u', ' ', $keyword);
        $keyword_arr = preg_split('/\s+/', $keyword, 10, PREG_SPLIT_NO_EMPTY);
        $keyword_quert_raw = '';
        foreach ($keyword_arr as $key => $val) {
            $keyword_quert_raw .= ``content` LIKE "%' . $val . '%"';
                if ($key < count($keyword_arr) - 1) {
                    $keyword_quert_raw .= ' OR ';
                }
            }

        $map = ``shopid` = ' . $this->shopid . ' AND (' . $keyword_quert_raw . '
        $fields = '*';
        $lists = (new SearchModel)->getListByPage($map, $order, $fields, $rows);
        //.... 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.

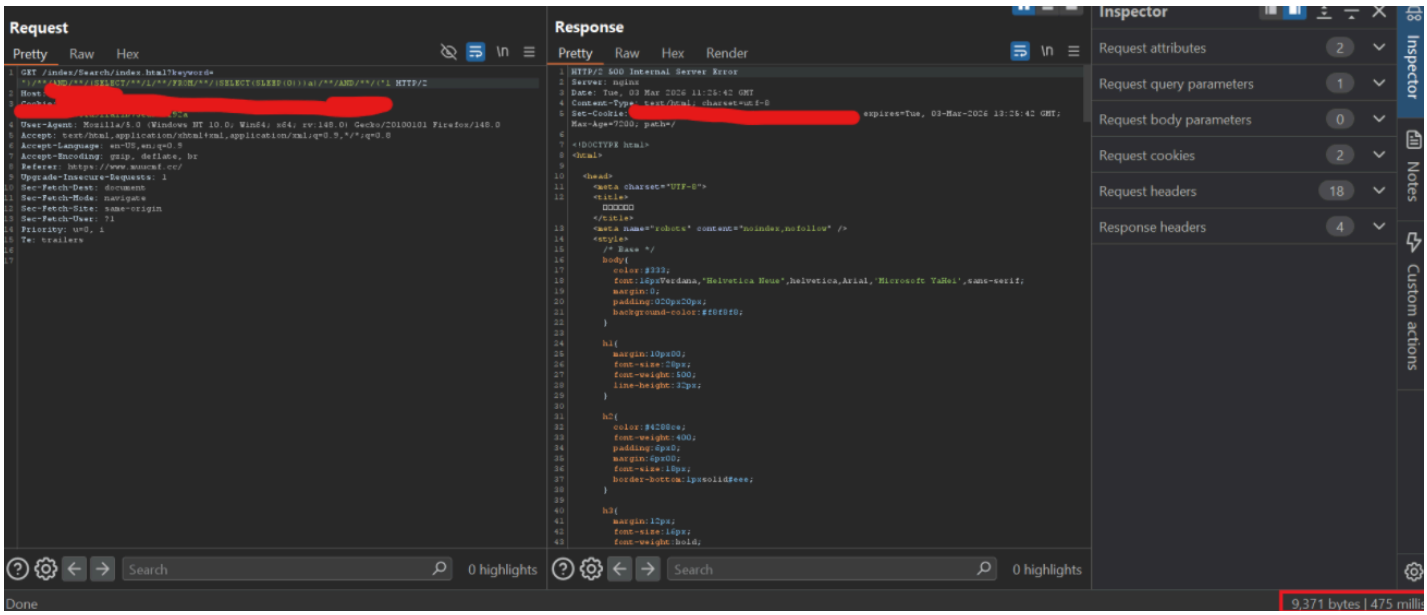
Exploit:

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.

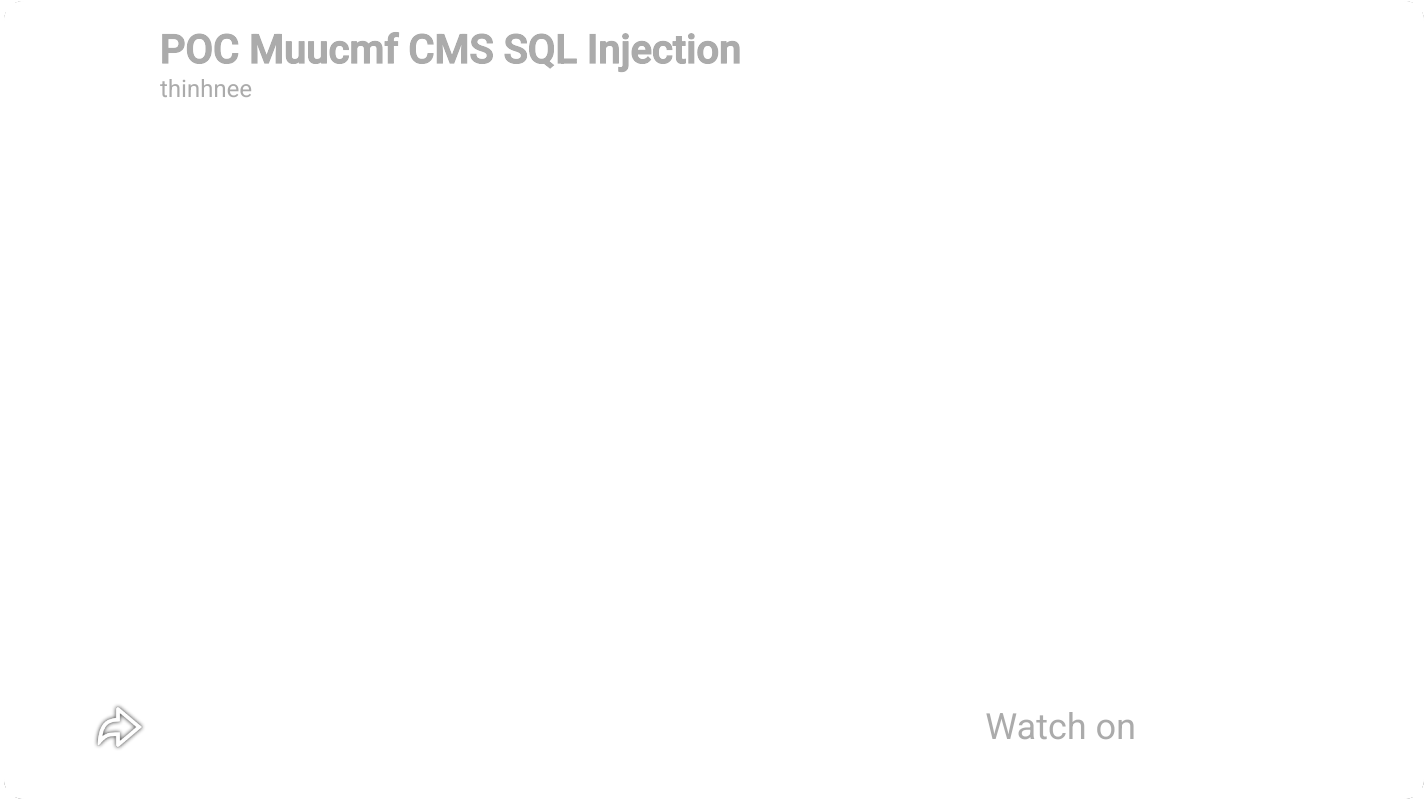


Exploit with sleep 3 sec (may be deplay cause network)



Exploit with sleep 0 sec

Video demo:



Vulnerability Research, Web Security

cve sqli poc exploit

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 recieve 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

-