**MeetCute Security**

**Meteor Security background**

**Login methodology:** Facebook login, which uses OAuth 2.0

MeetCute registers with Facebook, and obtains a Secret and an ID.

1. When **User** tells MeetCute to access Facebook, **User** is sent to Facebook where she tells Facebook that she would indeed like to give MeetCute permissions to specific information.
2. Facebook redirects **User** back to MeetCute, along with an Authorization Code.
3. MeetCute then passes that Authorization Code along with its Secret back to Facebook in return for a Security Token.
4. MeetCute then makes requests to Facebook on behalf of **User** by bundling the Security Token along with requests.

**Risks:**

1. NoSQL Injection: Due to the technology stack - Meteor & NoSQL:

NoSQL Injection risk for messaging: Fake messages

* Need: Message/User ID’s
* Why?
* All the data is mirrored client/server
* Users have admin access to DB

**Preventing NoSQL Injection**

-Remove **Insecure** package

Prevent admin privileges on database

-Create meteor methods & utilized **check** package

Allows for validation of fields - prevent Mongo operators as args

Separate client code from database logic

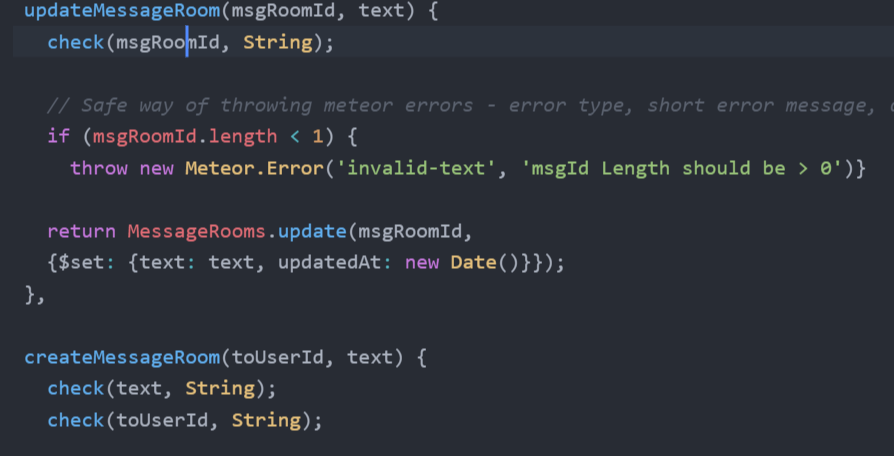
Meteor.call - parallel client/server run

-Limit exposed data to the client database - removed **autopublish**

We expose userId’s and messageRoom ID’s which should be kept private

**Add MeteorMethods:** Prevent CRUD operations from console/to miniMongoDB

1. The client sends a request to the server to run the method in a secure environment, just like an AJAX request would work
2. A simulation of the method runs directly on the client to attempt to predict the outcome of the server call using the available information



1. **Insecure package -** this package is for prototyping only. It allows methods such as insert, update, remove to be called from the client. It will be removed so that the application

can restrict database access. Grants any connected client the right to modify data in the database, just as if they had server access.

2. **Autopublish package** - takes server side database & mirrors all of its content on the client (not passwords/private items)

Meteor mirrors data on the client - but not all the data. Need to define which subset of data that the client is allowed to have.

-Tell meteor what you want to send to the client, not what not to send.

-Ensure you only publish the data that you want the client to be able to access. Any data subscribed to will be mirrored on the client due to MiniMongo so that it is local to the client.

Meteor has two ways of dealing with crud operations for documents - methods + allow/deny

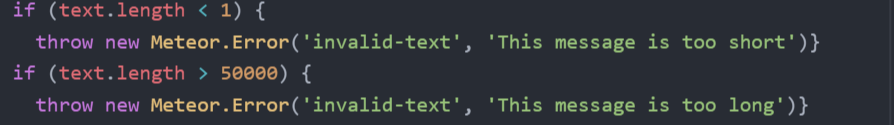
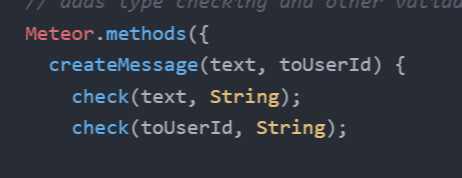
Ensure we can pass some of the applicable level 1 security standards identified by ASVS

Review OWASP testing guide

**Results of Project Security Work:**

Removed packages called autopublish & insecure for security purposes. Heavy refactoring of application methods & CRUD operations to work without these. Added input checks to further prevent NoSQL injection:

**Checking Input:**



**Manual Security Tests:**

-Navigate to all screens while logged out

-Log in, navigate to screen, log out, perform action to validate it no longer works.

**Future Security Considerations:**

**Facebook Login Security Risks**

Uses OAuth as previously mentioned.

Application gives up control of its userbase

Controlled by OAuth provider (Facebook)

Application could get shut off at any time by provider

A hacked Facebook grants access to MeetCute + **all** other applications

Mitigate w/ two factor auth/password on login

Shares data w/ Facebook - likes/comments

Currently showing full name from Facebook

* Users could export names from browser
  + Sell the data
  + Harass users/stalking
* Mitigate: Generate username/show first name only.

**We should add browser-policy package:**

Prevent code piggy backing

**X-Frame-Options:** Limit sites that can frame the app

Protects against: Clickjacking

**Content-Security-Policy:** Attack detection

Reports/Mitigates: XSS

**References:**

https://www.youtube.com/watch?v=xAdwQ4UAoj0

https://atmospherejs.com/meteor/insecure

https://www.discovermeteor.com/blog/meteor-and-security/

<https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf>

<https://en.wikipedia.org/wiki/OAuth>

<http://stackoverflow.com/questions/4727226/on-a-high-level-how-does-oauth-2-work>