Security Document

Kevin Niemiller

Cincinnati Road Activity application uses an account entry/sign-up package where the application is hashing every password that is entered on sign-up and when a user signs in. The application uses a SHA256 hashing algorithm to perform the hash. It also uses SALT, which adds random encrypted hash codes to the hashed password. SALT helps to prevent hacking that uses lookup tables and rainbow tables.

Meteor allows the developer to use a SSL\TLS connection. To do this with meteor, a command of meteor add force-ssl is performed which forces Meteor to use SSL\TLS connection. This ensures that all interactions to the server are always encrypted to protect users from active spoofing attacks. Along with this, the standard SSL\TLS certificate must be created. The meteor application bundle does not provide a place to put the certificate, so a proxy server that terminates SSL\TLS in front of the Meteor bundle and set the standard x-forwarded-proto header to get SSL\TLS to work.

Cincinnati Road Activity uses MongoDB as it's database, a collection \ no-SQL based database. MongoDB does not suffer from SQL injection. The reason for this is because MongoDB does not parse inputs/data. Since MongoDB does not parse structured text to figure out what it needs to do with it, there is no possibility of misinterpreting user input as instructions, which means there is no security hole of SQL injection.

Cincinnati Road Activity is a Meteor application, which allows the developer to only "publish" certain information/data from the server to the client. If nothing is published, then the client will never receive data from the database. This allows the developer to hide/reduce/secure information that he doesn't want the client to get to. With Cincinnati Road Activity, I chose to publish all data that is relevant to the user that is logged in. This way, the user can only see data that he already has access to and cannot access other people's data.

Web applications have to worry about both client side and server side cross site scripting hacks. With Meteor, it currently does not handle server side rendering so cross site scripting does not apply to the server side, but can still apply to the client side. For the client side, meteor uses a package called browser-policy which allows the developer to set different rules to help prevent a cross site scripting hack. Some of these rules include the ability to only load resources from the current origin of the application, not allow eval or similiar functions, remove the ability to use inline scripts, and to block external services such as AJAX, Websockets, and similar techniques.