

---

This module is currently deprecated and unmaintained. Please check #148 which explain how to do this with modern versions of socket.io and using passport directly.

## passport.socketio

Access passport.js user information from a socket.io connection.

### Installation

```
1 npm install passport.socketio
```

### Example usage

```
1
2 // initialize our modules
3 var io          = require("socket.io")(server),
4     sessionStore = require('awesomeSessionStore'), // find a
5               working session store (have a look at the readme)
6     passportSocketIo = require("passport.socketio");
7
8 // With Socket.io < 1.0
9 io.set('authorization', passportSocketIo.authorize({
10   cookieParser: express.cookieParser,
11   key:          'express.sid',          // the name of the cookie where
12               express/connect stores its session_id
13   secret:       'session_secret',      // the session_secret to parse the
14               cookie
15   store:        sessionStore,          // we NEED to use a sessionstore.
16               no memorystore please
17   success:      onAuthorizeSuccess,    // *optional* callback on success -
18               read more below
19   fail:         onAuthorizeFail,       // *optional* callback on fail/
20               error - read more below
21 }));
22
23 //With Socket.io >= 1.0
24 io.use(passportSocketIo.authorize({
25   cookieParser: cookieParser,          // the same middleware you
26               registrer in express
27   key:          'express.sid',          // the name of the cookie where
28               express/connect stores its session_id
29   secret:       'session_secret',      // the session_secret to parse the
30               cookie
```

---

```
22   store:      sessionStore,      // we NEED to use a sessionstore.
    no memorystore please
23   success:    onAuthorizeSuccess, // *optional* callback on success
    - read more below
24   fail:       onAuthorizeFail,    // *optional* callback on fail/
    error - read more below
25   }));
26
27   function onAuthorizeSuccess(data, accept){
28     console.log('successful connection to socket.io');
29
30     // The accept-callback still allows us to decide whether to
31     // accept the connection or not.
32     accept(null, true);
33
34     // OR
35
36     // If you use socket.io@1.X the callback looks different
37     accept();
38   }
39
40   function onAuthorizeFail(data, message, error, accept){
41     if(error)
42       throw new Error(message);
43     console.log('failed connection to socket.io:', message);
44
45     // We use this callback to log all of our failed connections.
46     accept(null, false);
47
48     // OR
49
50     // If you use socket.io@1.X the callback looks different
51     // If you don't want to accept the connection
52     if(error)
53       accept(new Error(message));
54     // this error will be sent to the user as a special error-package
55     // see: http://socket.io/docs/client-api/#socket > error-object
56   }
```

## passport.socketio - Options

### store [function] required:

Always provide one. If you don't know what sessionStore to use, have a look at this list. Also be sure to use the same sessionStore or at least a connection to *the same collection/table/whatever*. And don't forget your `express.session()` middleware: `app.use(express.session({ store : awesomeSessionStore }));` For further info about this middleware see the official docu-

---

mentation.

You can also check the simple example below using a redis store.

```
1 //in your app.js
2 var sessionStore = new redisStore();
3
4 app.use(session({
5   key: 'express.sid',
6   store: sessionStore,
7   secret: 'keyboard cat'
8 }));
9
10 //in your passport.socketio setup
11 //With Socket.io >= 1.0 (you will have the same setup for Socket.io <1)
12 io.use(passportSocketIo.authorize({
13   cookieParser: require('cookie-parser'), //optional your cookie-parser
14   key: 'express.sid', //make sure is the same as in your
15   secret: 'keyboard cat', //make sure is the same as in your
16   store: sessionStore, //you need to use the same
17   success: onAuthorizeSuccess, // *optional* callback on success
18   fail: onAuthorizeFail, // *optional* callback on fail/
19   error
20 }));
```

### **cookieParser [function] optional:**

Optional cookieParser from express. Express 3 is `express.cookieParser` in Express 4 `require('cookie-parser')`.

Defaults to `require('cookie-parser')`.

### **key [string] optional:**

Defaults to `'connect.sid'`. But you're always better off to be sure and set your own key. Don't forget to also change it in your `express.session(): app.use(express.session({ key: 'your.sid-key' }));`

---

### **secret [string] optional:**

As with `key`, also the secret you provide is optional. *But:* be sure to have one. That's always safer. You can set it like the key: `app.use(express.session({ secret: 'pinkie ate my cupcakes!' }));`

### **passport [function] optional:**

Defaults to `require('passport')`. If you want, you can provide your own instance of passport for whatever reason.

### **success [function] optional:**

Callback which will be called everytime a *authorized* user successfully connects to your socket.io instance. **Always** be sure to accept/reject the connection. For that, there are two parameters: `function(data[object], accept[function])`. `data` contains all the user-information from passport. The second parameter is for accepting/rejecting connections. Use it like this if you use socket.io under 1.0:

```
1 // accept connection
2 accept(null, true);
3
4 // reject connection (for whatever reason)
5 accept(null, false);
```

And like this if you use the newest version of socket.io@1.X

```
1 // accept connection
2 accept();
3
4 // reject connection (for whatever reason)
5 accept(new Error('optional reason'));
```

### **fail [function] optional:**

The name of this callback may be a little confusing. While it is called when a not-authorized-user connects, it is also called when there's a error. For debugging reasons you are provided with two additional parameters `function(data[object], message[string], error[bool], accept[function])`: (socket.io @ < 1.X)

```
1 /* ... */
```

---

```

2 function onAuthorizeFail(data, message, error, accept){
3   // error indicates whether the fail is due to an error or just a
   // unauthorized client
4   if(error){
5     throw new Error(message);
6   } else {
7     console.log(message);
8     // the same accept-method as above in the success-callback
9     accept(null, false);
10  }
11 }
12
13 // or
14 // This function accepts every client unless there's an error
15 function onAuthorizeFail(data, message, error, accept){
16   console.log(message);
17   accept(null, !error);
18 }

```

Socket.io@1.X:

```

1 function onAuthorizeFail(data, message, error, accept){
2   // error indicates whether the fail is due to an error or just a
   // unauthorized client
3   if(error) throw new Error(message);
4   // send the (not-fatal) error-message to the client and deny the
   // connection
5   return accept(new Error(message));
6 }
7
8 // or
9 // This function accepts every client unless there's an critical error
10 function onAuthorizeFail(data, message, error, accept){
11   if(error) throw new Error(message);
12   return accept();
13 }

```

You can use the `message` parameter for debugging/logging/etc uses.

### **socket.handshake.user (prior to v1)**

This property was removed in v1. See `socket.request.user`

### **socket.request.user (as of v1)**

This property is always available from inside a `io.on('connection')` handler. If the user is authorized via passport, you can access all the properties from there. **Plus** you have the `socket`.

---

`request.user.logged_in` property which tells you whether the user is currently authorized or not.

**Note:** This property was named `socket.handshake.user` prior to v1

## Additional methods

### `passportSocketIo.filterSocketsByUser`

This function gives you the ability to filter all connected sockets via a user property. Needs two parameters `function(io, function(user))`. Example:

```
1 passportSocketIo.filterSocketsByUser(io, function(user){
2   return user.gender === 'female';
3 }).forEach(function(socket){
4   socket.emit('message', 'hello, woman!');
5 });
```

## CORS-Workaround:

If you happen to have to work with Cross-Origin-Requests (marked by socket.io v0.9 as `handshake.xdomain` and by socket.io v1.0 as `request.xdomain`) then here's a workaround:

## Clientside:

You have to provide the session-cookie. If you haven't set a name yet, do it like this: `app.use(express.session({ key: 'your.sid-key' }));`

```
1 // Note: there's no readCookie-function built in.
2 // Get your own in the internetz
3 socket = io.connect('//' + window.location.host, {
4   query: 'session_id=' + readCookie('your.sid-key')
5 });
```

## Serverside:

Nope, there's nothing to do on the server side. Just be sure that the cookies names match.

---

## Notes:

- Does **NOT** support cookie-based sessions. eg: `express.cookieSession`
- If the connection fails, check if you are requesting from a client via CORS. Check `socket.handshake.xdomain === true` (`socket.request.xdomain === true` with `socket.io v1`) as there are no cookies sent. For a workaround look at the code above.

## Contribute

You are always welcome to open an issue or provide a pull-request! Also check out the unit tests:

```
1 npm test
```

## License

Licensed under the MIT-License. 2012-2013 José F. Romaniello.