

# Role-Based Access Control

In a secure web-app system

Jonathan Hsu, Patrick Liao, Zaprin Ignatiev

# Problem

How do we restrict or allow a user's access to content on a web app?

Example: If a user is not a member of a private category, they should not be able to view it

# Threat Model

**Attacker** 

---

## **Assumptions:**

- Attacker is registered as a regular user

## **Motivations:**

- Create, Read, Update, Delete

## **Capabilities:**

- Forge HTTP requests
- Disguise as a user with more privileges

**Defender** 

---

## **Motivations:**

- Protect user data
- Verification of user identity
- Protect content from unauthorized access

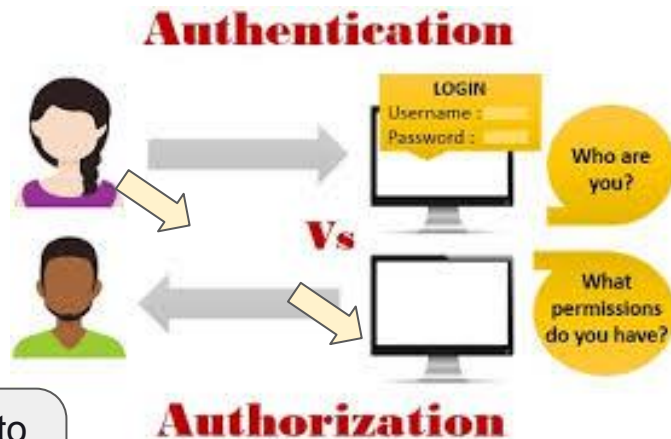
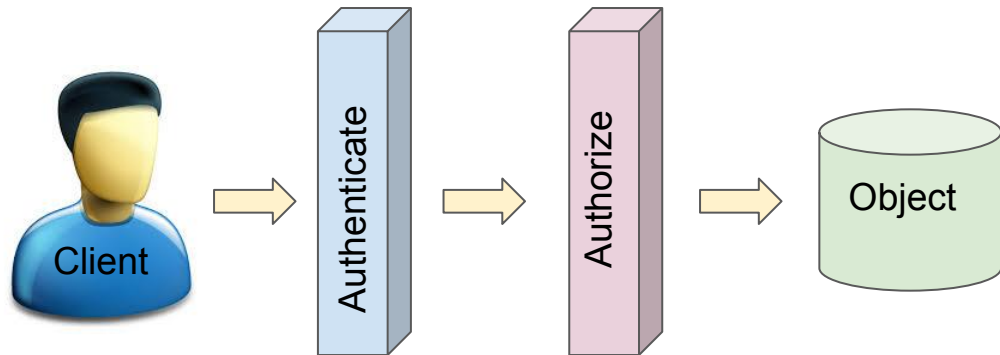
# Access Control



# What is Access Control?

Access control is a security technique that regulates who or what can view, use or change data in a computing environment.

At a high level, access control is a selective restriction of access to data. It consists of two main components: **authentication** and **authorization**, which focuses on data security.



Daniel Crowley, head of research for IBM's X-Force Red

# Access Control Authentication



## Login

Username

Password

Log In

Or

Create an Account

[<= Back to Forum](#)

# Authentication

The structure of the table used to hold user information:

## **users**

user_id	username	hashed_password	admin
1	jonathan	*****	1
2	sam	*****	0
3	pete	*****	0
4	dave	*****	0

# Sessions



User

HTTP  
Client

HTTP  
Server

Login  
POST  
username=david  
password=davidh

Login successful?

1. create session id
2. return session id in cookie
3. store session id in database

Set-Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C

SESSION ID

Sessionid  
Username  
createDate  
expireDate  
lastAccessDate

HTTP is Stateless

Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C

Lookup Session ID

1. session match a username?
2. session still valid?

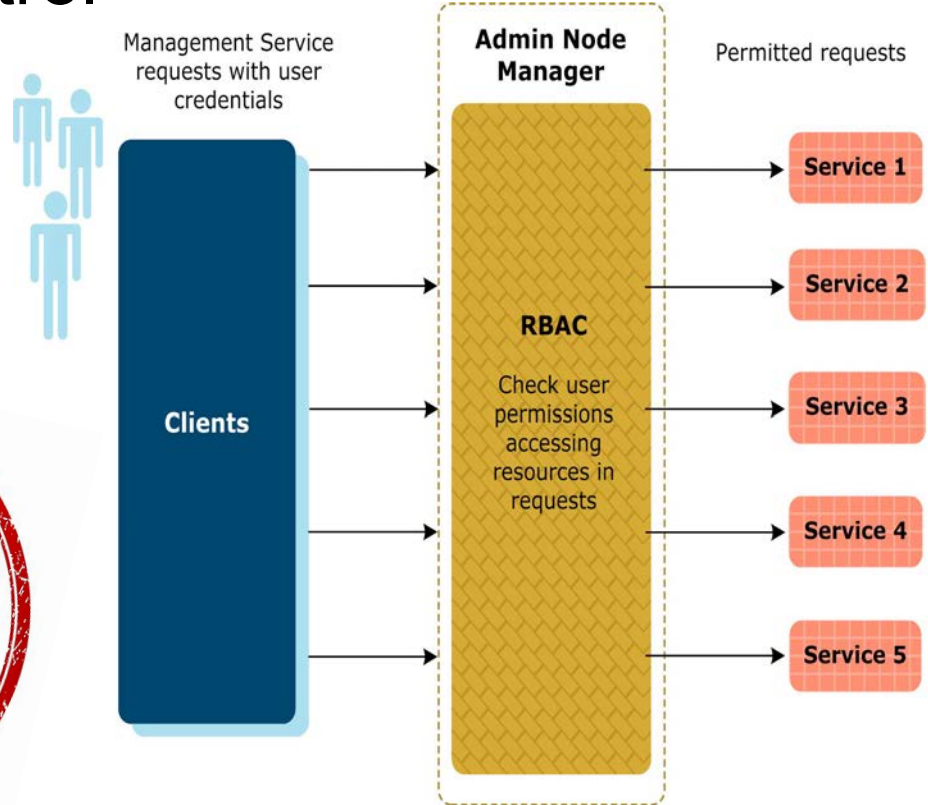
Content for 'david'

Database





# Role-Based Access Control (Authorization)

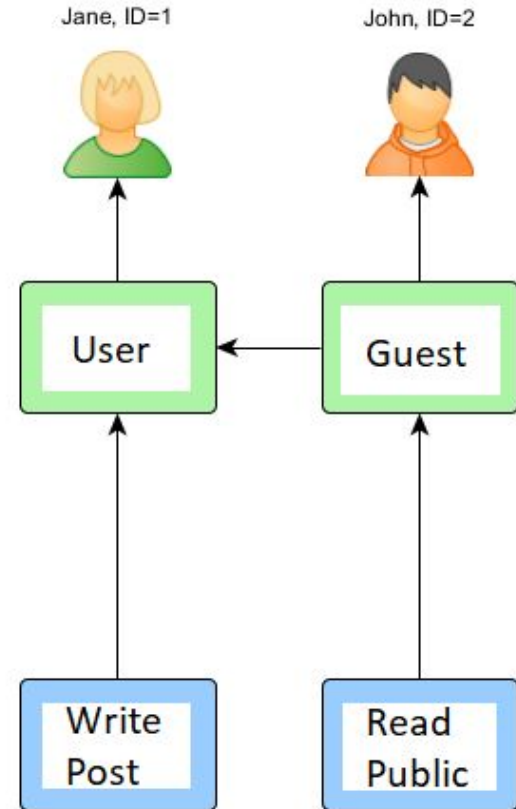


# Why Hierarchical Role Based Access Control (HRBAC)?

Specific user roles and operations within those roles.

Don't have to configure operations for each user as roles extend each other.

Changing the privileges for a role is simple because it inherits from other roles.



# Why Discretionary Access Control (DAC)?

To establish and change ownership/policies of resources.

To allow owners of private categories to add members and promote certain members to moderators.

## Discretionary Access Control



In discretionary access control (DAC), **owner of a resource decides** how it can be shared

- Owner can choose to give **read or write access to other users**

# HRBAC Model

	Admin (member of all categories)	Owner (member of category)	Moderator (member of category)	User	Guest
READ	public categories member categories	public categories member categories	public categories member categories	public categories member categories	public categories
CREATE	category post/reply in public category post/reply in member category add user to member category add moderator to member category	category post/reply in public category post/reply in member category add user to member category add moderator to member category	category post/reply in public category post/reply in member category add user to member category	category post/reply in public category post/reply in member category	
UPDATE	owned post owned reply name of member category visibility of member category owner of category	owned post owned reply name of member category visibility of member category	owned post owned reply	owned post owned reply	
DELETE	owned post owned reply user from member category post/reply in member category member category moderator from member category	owned post owned reply user from member category post/reply in member category member category moderator from member category	owned post owned reply user from member category post/reply in member category	owned post owned reply	

- User and member have the same privileges, except a member is a user who has access to at least one private category.

# HRBAC and DAC Feature

**categories**

category_id	title	public
1	News	1
2	Private	0

**roles within categories**

category_id	user_id	role
1	2	1
1	3	2
2	3	1
2	4	3

Roles:

1. Owner
2. Moderator
3. Member

- User and member have the same privileges, except a member is a user who has access to at least one private category.

# Access Control Implementation

```
    return ret
},
functionArgs: function
    var l = fn.
    if ( !l ) r

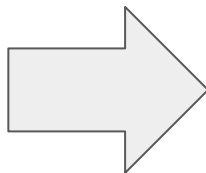
    var args =
    while ( l--
        args[l]
    return

},
key:quote, //ob
functionCode: [
    quote:quote
    339
    340
    341
    342
    343
    344
    345
    346
    347
    348
    349
    350
    351
    352
    353
```

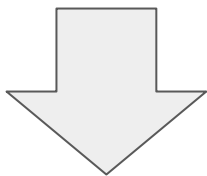


# 1. Define Roles and Library usage

We used a library called  
`accesscontrol`, which helps  
us set the user roles.



As well as check permission.



```
const permission = (user_role === 'owner')
  ? ac.can(user_role).deleteOwn('category')
  : ac.can(user_role).deleteAny('category');
return permission.granted;
```

```
const ac = new AccessControl();
ac.grant('user')
  .createOwn('post')
  .updateOwn('post')
  .deleteOwn('post')
  .createOwn('reply')
  .updateOwn('reply')
  .deleteOwn('reply')
  .createOwn('category')
  .grant('moderator')
  .extend('user')
  .deleteAny('post')
  .deleteAny('reply')
  .deleteOwn('category')
  .grant('owner')
  .extend('moderator')
  .updateOwn('category')
  .grant('admin')
  .extend('moderator')
  .deleteAny('category');
ac.deny('admin').deleteAny('category');
```

## 2. Check User Against User Table

We check if the user exists and if he is an admin. If the user doesn't exist he is a guest.

### **users**

user_id	username	hashed_password	admin
1	jonathan	*****	1
2	sam	*****	0
3	pete	*****	0
4	dave	*****	0



### 3. Check Ownership/Membership Against Resource

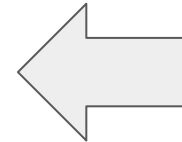
We check ownership, by comparing the user id to the resource owner's id in the database, this determines if the user can modify the given resource.

posts					
category_id (ref)	post_id (unique)	user_id (ref)	date_created	title	content
1	1	2	5/19/2019 7:08	My post title	Content of this post
1	2	3	5/19/2019 15:08	Other Post	Content Lorem ipsu
2	3	3	5/18/2019 16:08	Say hi	I'm pete

replies					
category_id (ref)	post_id (ref)	reply_id (unique)	user_id (ref)	date_created	content
1	1	1	3	5/19/2019 18:15	content of reply
2	3	2	4	5/19/2019 22:11	hello! I'm dave

roles

category_id	user_id	role
1	2	1
1	3	2
2	3	1
2	4	3



Role:

1. Owner
2. Moderator
3. Member

## 4. Check Operation Against Role

If step 3 did not return false, we check to determine if the role can execute the operation requested on the specific attribute, based on step 1 ( Roles Definition).

```
const permission = (user_role === 'owner')  
  ? ac.can(user_role).deleteOwn('category')  
  : ac.can(user_role).deleteAny('category');  
return permission.granted;
```

# Features of our web forum

Basic Functionality (Node, express, SQLite, javascript)

Create, read, update and delete (CRUD) for:

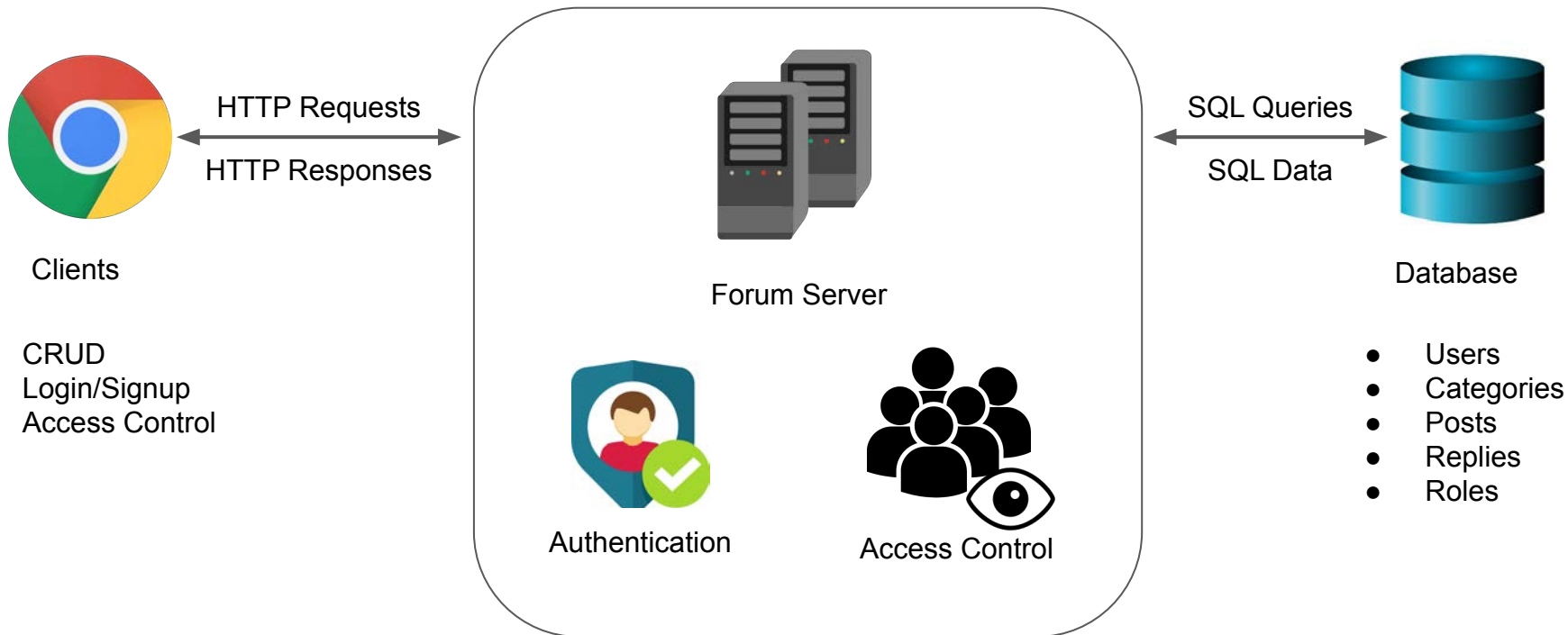
Categories, Posts (threads), Replies

Authentication (Passport, Express Session)

Access Control

Private categories, edit/delete permissions, roles

# Structure



# Sample of database content

## categories

category_id (unique)	title	public
1	News	1
2	Private	0

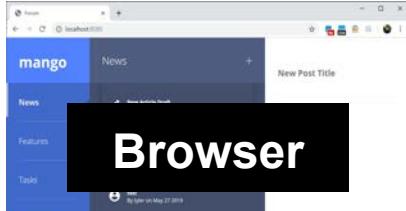
## posts

category_id (ref)	post_id (unique)	user_id (ref)	date_created	title	content
1	1	2	5/19/2019 7:08	My post title	Content of this post
1	2	3	5/19/2019 15:08	Other Post	Content Lorem ipsu
2	3	3	5/18/2019 16:08	Say hi	I'm pete

## replies

category_id (ref)	post_id (ref)	reply_id (unique)	user_id (ref)	date_created	content
1	1	1	3	5/19/2019 18:15	content of reply
2	3	2	4	5/19/2019 22:11	hello! I'm dave

# One Example: Create a post



Display category/posts

Write post

Display new post

Read category

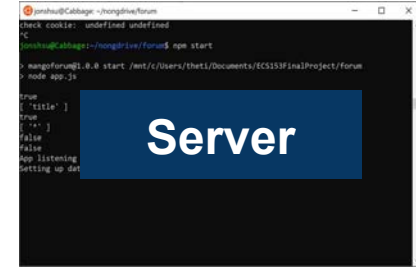
Category posts

Create post

Success

Read category

Category posts



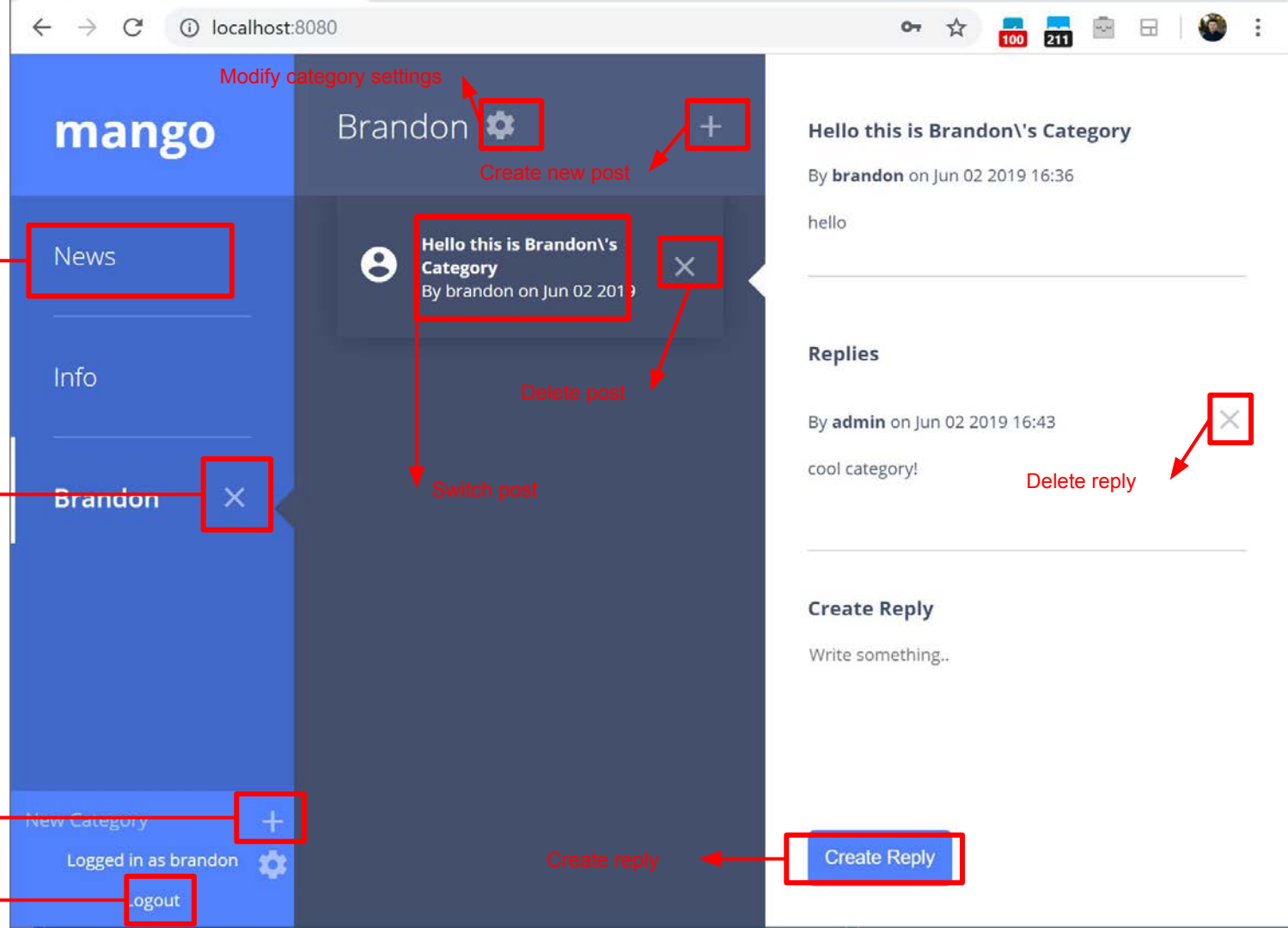
Get user from session

Check if operation is allowed

Read/Insert in database

# Basic Functionality

# Functionality: Forum Functions





Brandon 



## Category Settings



Visibility:

- ☐ Public  
☒ Private

Update Visibility

Change visibility

Users:

Username	Type	Manage
brandon	Owner	
cameron	User ▼	Remove User
Add User	User ▼	Add User

Add/remove/modify users

## Functionality: Category Settings, Post Creation

Brandon 



New Post Title



New Article Draft  
By brandon

Write something..

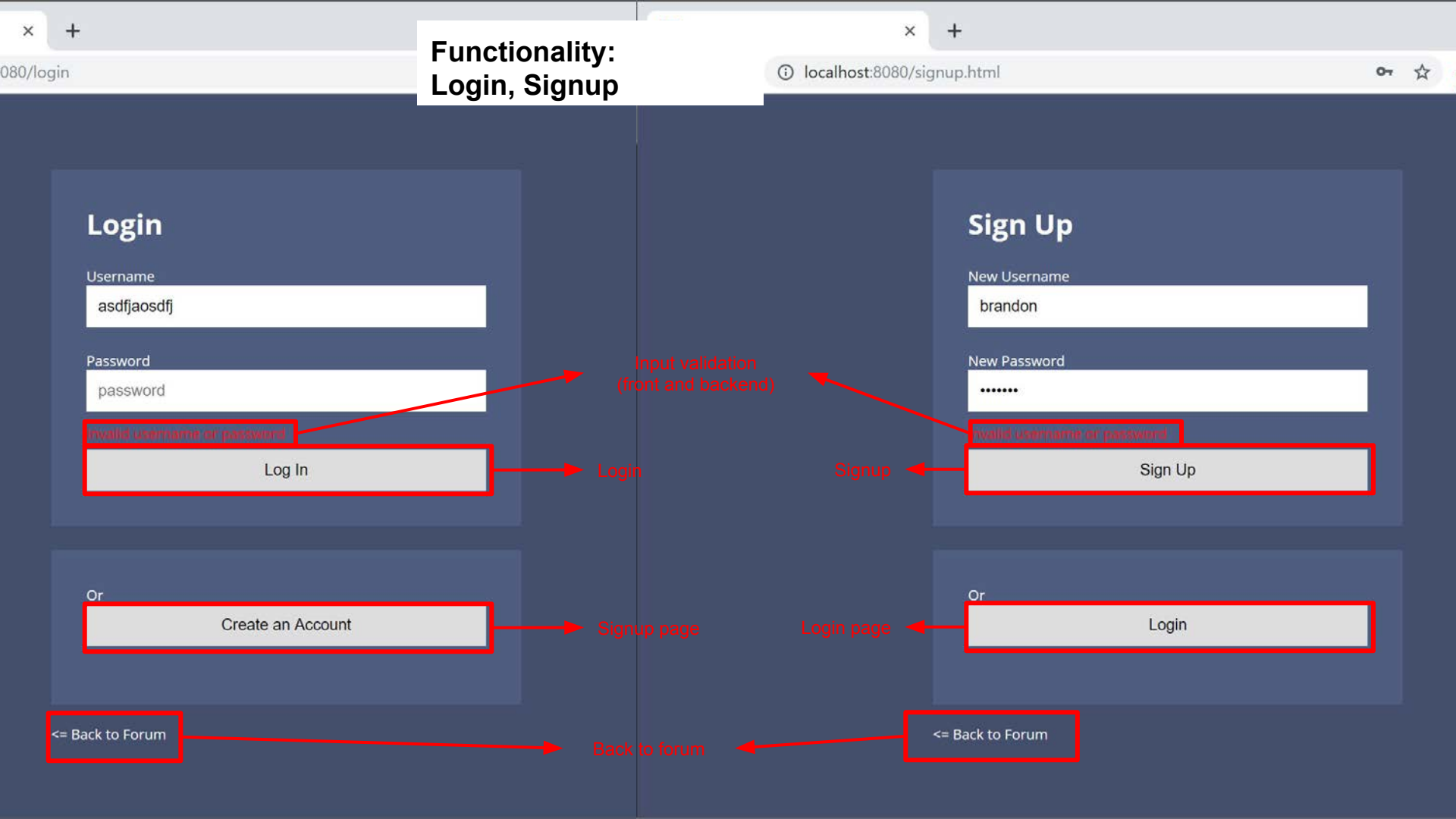


Hello this is Brandon's  
Category  
By brandon on Jun 02 2019



Create Post

Create post



# Selected Experimental Results

Authentication/Sessions

Password Management

Authorization (Access Control)

Brute Force/DDoS

CORS/SOP

# Authentication/Sessions

We see that invalid credentials returns 401 ↓

The screenshot shows a REST client interface with a POST request to `localhost:8080/login`. The request body is a JSON object: `{"username": "brandon", "password": "incorrectpassword"}`. The response status is `401 Unauthorized` with a time of 6 ms and a size of 309 B. The response body shows a `Login failure` message.

```
POST localhost:8080/login
```

localhost:8080/login

POST localhost:8080/login

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code Comments (0)

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 { "username": "brandon", "password": "incorrectpassword" }
```

Body Cookies (1) Headers (9) Test Results Status: 401 Unauthorized Time: 6 ms Size: 309 B Download

Pretty Raw Preview HTML

```
1 Login failure
```

The screenshot shows a REST client interface with a POST request to `localhost:8080/login`. The request body is a JSON object: `{"username": "brandon", "password": "brandon"}`. The response status is `200 OK` with a time of 17 ms and a size of 299 B. The response body shows a session cookie named `connect.sid`.

```
POST localhost:8080/login
```

POST localhost:8080/login

Params Authorization Headers (9) Body Pre-request Script Tests Cookies

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 { "username": "brandon", "password": "brandon" }
```

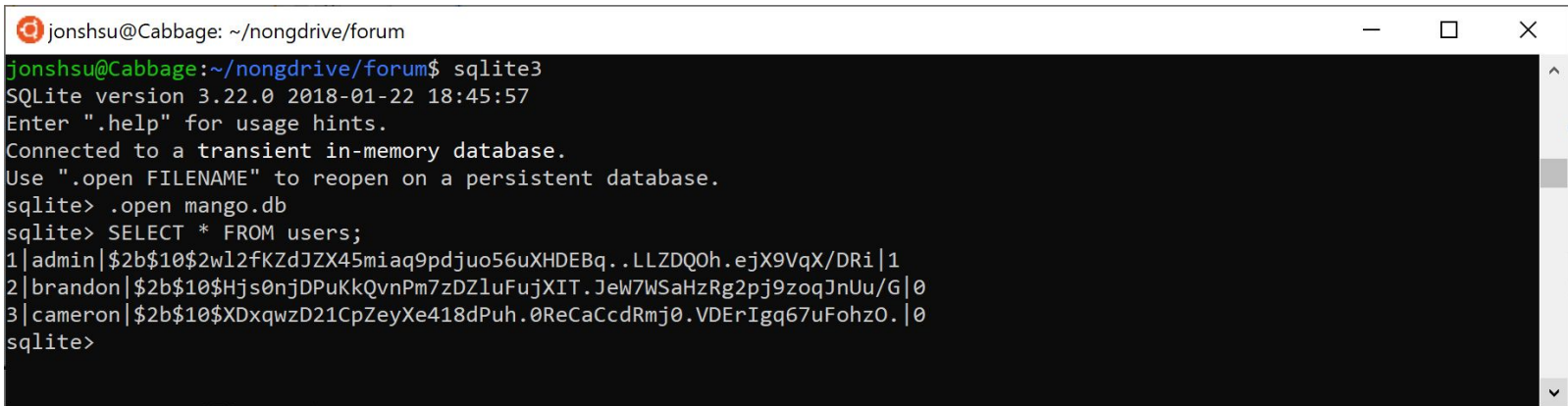
Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 17 ms Size: 299 B

Name	Value	Domain	Path	Expires	HttpOnly
connect.sid	s%3A4T0QC96j3WyfGqN1RNheSTaZuAW-NRxdJoReX4icmo3Sw2gsKG5cTgLvVEeRq1o12f0246f		/	Wed, 12 Jun 2019 08:55:57 GMT	true

And valid login sends session cookie ↑

# Password Management

We see that passwords are hashed  
with bcrypt

A terminal window with a title bar showing the user 'jonshsu' at host 'Cabbage' in the directory '~/nongdrive/forum'. The terminal content shows the execution of 'sqlite3', which displays its version (3.22.0) and timestamp (2018-01-22 18:45:57). It prompts for '.help' and shows it is connected to a transient in-memory database. The user enters '.open mango.db' to connect to a persistent database. Then, the user enters 'SELECT \* FROM users;', which returns a table with three rows of user data. Each row contains an ID, a username, a bcrypt-hashed password, and an email address.

```
jonshsu@Cabbage: ~/nongdrive/forum
jonshsu@Cabbage:~/nongdrive/forum$ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open mango.db
sqlite> SELECT * FROM users;
1|admin|$2b$10$2w12fKZdJZX45miaq9pdjuo56uXHDEBq..LLZDQOh.ejX9VqX/DRI|1
2|brandon|$2b$10$Hjs0njDPuKkQvnPm7zDZ1uFujXIT.JeW7WSaHzRg2pj9zoqJnUu/G|0
3|cameron|$2b$10$XDxqwzD21CpZeyXe418dPuh.0ReCaCcdRmj0.VDErIqg67uFohz0.|0
sqlite>
```

# Authorization (Access Control)

We see that unauthorized operations are not allowed through our access control function

The screenshot shows a web browser's developer tools interface. The top bar indicates the environment is 'No Environment'. The active tab is 'delete\_unauthorized'. The request method is 'DELETE' and the URL is 'localhost:8080/deletepost?categoryid=1&postid=1'. The 'Params' tab is selected, showing a table of query parameters:

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	categoryid	1	
<input checked="" type="checkbox"/>	postid	1	
	Key	Value	Description

Below the table, the 'Body' tab is selected, showing the response status: 'Status: 401 Unauthorized', 'Time: 594 ms', and 'Size: 319 B'. The response body is displayed in 'Pretty' format, showing a JSON object: 

```
{ 1: 'Post: failed to delete'}
```

# Brute Force/DDoS

We get a “429 too many requests” status if too many requests are sent

Collection Runner

File Edit View Help

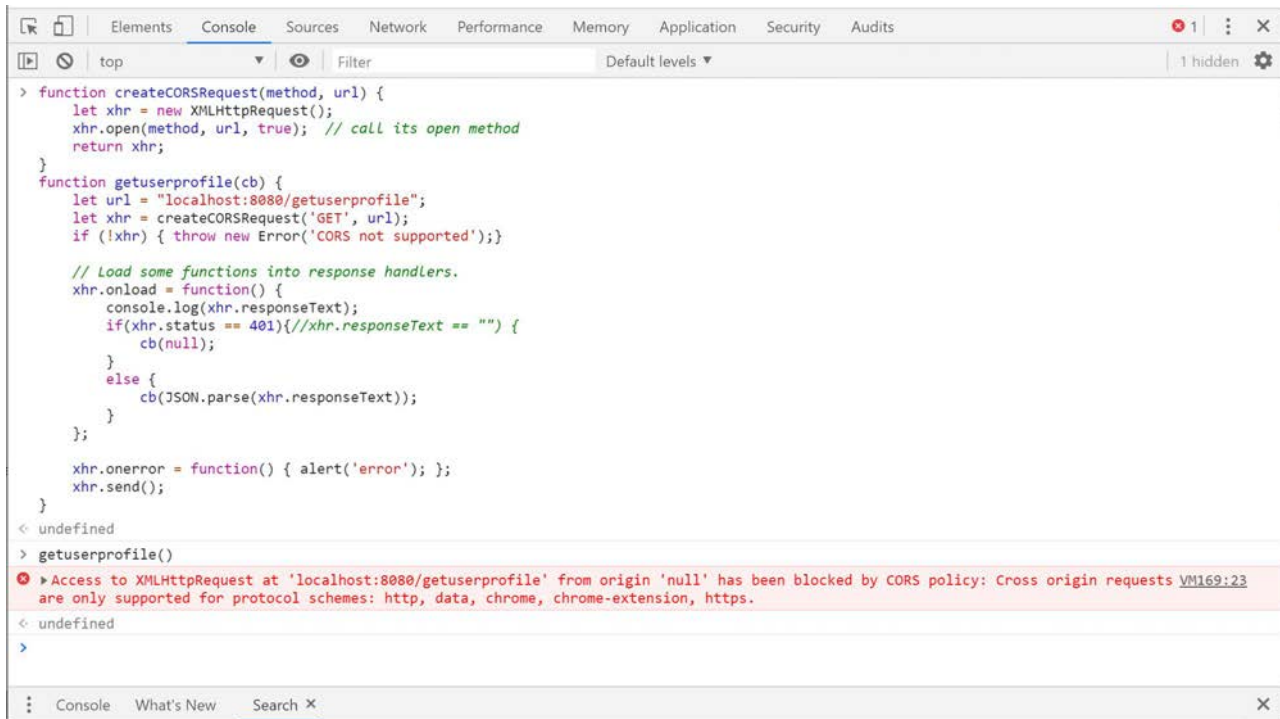
Collection Runner Run Results My Workspace Run In Command Line Docs

0 PASSED 0 FAILED post No Environment just now Run Summary Export Results Retry New

Iteration	Method	Endpoint	Path	Status	Time	Size
99	GET	localhost:8080/getpost?c...	post / Get Post	200 OK	11 ms	276 B
This request does not have any tests.						
100	GET	localhost:8080/getpost?c...	post / Get Post	200 OK	5 ms	276 B
This request does not have any tests.						
101	GET	localhost:8080/getpost?c...	post / Get Post	429 Too Many Requests	36 ms	42 B
This request does not have any tests.						
102	GET	localhost:8080/getpost?c...	post / Get Post	429 Too Many Requests	4 ms	42 B

# CORS/SOP

We see that requests from other origins are not allowed



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays a JavaScript function `createCORSRequest` and its usage in `getuserprofile`. A red error message is visible at the bottom of the console, indicating a CORS policy violation. The error message states: 'Access to XMLHttpRequest at 'localhost:8080/getuserprofile' from origin 'null' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: http, data, chrome, chrome-extension, https.'

```
> function createCORSRequest(method, url) {  
  let xhr = new XMLHttpRequest();  
  xhr.open(method, url, true); // call its open method  
  return xhr;  
}  
  
function getuserprofile(cb) {  
  let url = "localhost:8080/getuserprofile";  
  let xhr = createCORSRequest('GET', url);  
  if (!xhr) { throw new Error('CORS not supported');}  
  
  // Load some functions into response handlers.  
  xhr.onload = function() {  
    console.log(xhr.responseText);  
    if (xhr.status == 401) { // xhr.responseText == "" {  
      cb(null);  
    }  
    else {  
      cb(JSON.parse(xhr.responseText));  
    }  
  };  
  
  xhr.onerror = function() { alert('error'); };  
  xhr.send();  
}  
  
< undefined  
> getuserprofile()  
  
< undefined  
>
```

Access to XMLHttpRequest at 'localhost:8080/getuserprofile' from origin 'null' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: http, data, chrome, chrome-extension, https.

Console What's New Search X



# Conclusions

Access control is a flexible method to manage user privileges and acts as an extra layer of security.

There are plenty of web security topics to explore and these are some of the basics.

# Related Works

- <https://www.phpbb.com/> - “free and open source forum software”

# Future Work

CSRF through [csrf](#)

[HTTPS](#)

Additional [authentication strategies](#)

Request-specific rate limiting

Use a frontend library like React