Authentication Tokens

Authentication Overview

- Registration
  - User sends username and password
  - Validate password strength
  - Store salted hash of the password
- Authentication
  - User sends username/password
  - Retrieve the stored salted hash
  - Salt and hash the provided password
  - If both salted hashes are identical, the user is authenticated

Authentication

- With authentication, we want users to be able to:
  - Access their private data
  - Make authenticated posts to the server

- When you take these actions on a web app:
  - The app should verify that **you** made the request
  - Only serve your private data to you
  - Do not let anyone else make posts in your name

Authentication

- How does the server verify that you are the one who made a specific request?
- We only have authentication, so...
  - .. Users type their username and password with every single protected request?..
  - Authenticate using the stored [salted hash of the] password?..
  - **No**! Terrible user experience
  - You would not use this site

Authentication

- How does the server verify that you are the one who made a specific request?
- We only have authentication, so...
  - .. Store the username/password in cookies and send them on every request?..
  - Password is stored client-side in plain text
  - **No**! Never store passwords in plain text
    - Not even client-side! Don't do it

Authentication Tokens

- Instead of authenticating the username/password on every request:
  - **Issue an authentication token**
- When a user is authenticated, generate a random authentication token
- Store this token in your database and mark the username who authenticated
- Set the token as a cookie for that user
- Whenever a request come with that token, treat them as the user associated with that token

Authentication Tokens

- With authentication tokens:
  - We have a login system 🎉
- As long as the user has the authentication token as their cookie, they are logged in*
  - All their requests are authenticated by the server

*You can/should set these tokens to expire either client-side (cookie expiration), server-side (storing an expiration timestamp in your database and ignoring the token after that timestamp), or both

Authentication Tokens

- Authentication tokens need to be random
  - The token must have enough entropy that is cannot be guessed
  - Eg. An attacker should not be able to send requests with random tokens until one matches a logged in user
- Generally, there should be at least 2^80 unique tokens that could be generated (80 bits of entropy)
  - More is better!

Authentication

- Once a token is generated, set it as a cookie
- Now the token will be sent with all subsequent requests
- Use the token to lookup the user
- The possession of the token verifies that this user did authenticate in the past

Storing Authentication Tokens

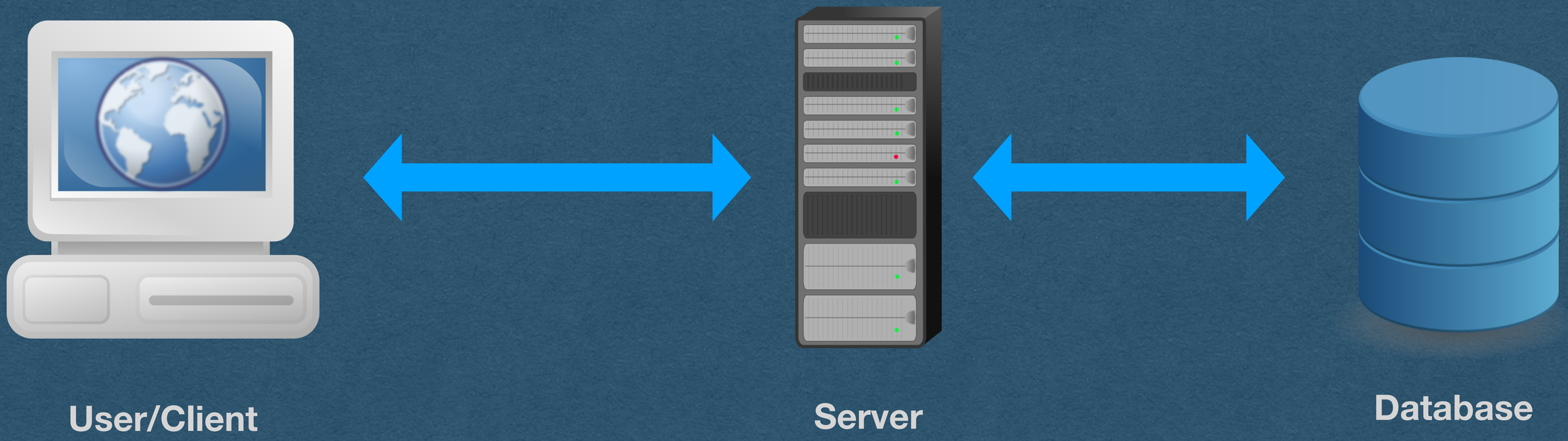
- **Caution**: These tokens need to be stored on the server
- These tokens are as sensitive as passwords!
  - Stealing a token and setting a cookie with that value grants access to an account without even needing a password
- **Solution**: Only store hashes of the tokens
  - Can salt for extra security/paranoia (Not necessary since the entropy is so high)
  - Salting also makes DB lookups more difficult

Authentication w/ Tokens

- Check each request for a cookie with a token
  - Lookup the hash of the token in the database
  - If the token is found, read the associated username
  - Proceed as though this request was made by that user
- If the token is invalid or no cookie is set
  - Do not respect the request and return a 400-level response code:
    - 401 Unauthorized - User is not logged in
    - 403 Forbidden - User in logged in, but trying to something they are not allowed to do
- Ensure all sensitive pages/features are secured this way!
  - The front end cannot be trusted (NEVER trust your users)

Registration and Login

- Let's look at the whole process of creating an account and logging in



Registration and Login

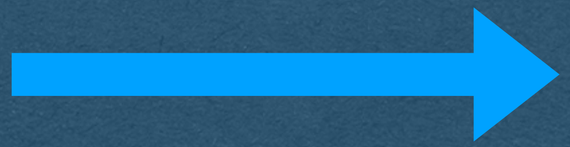
- User registers a new account

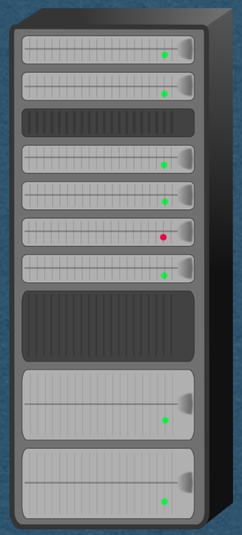
username

XX_UBStudent_XX

password

P@$$w0rd





User/Client

Server

Database

Registration and Login

- Server generates a random salt

User/Client

Server

username
XX_UBStudent_XX
password
P@$$w0rd
salt
hJ33fqAwscJmp3MacoQ8uO

Database

Registration and Login

- Append the salt to the password and hash

User/Client

Server

username

XX_UBStudent_XX

password

P@$$w0rd

salt

hJ33fqAwscJmp3MacoQ8uO

hash

f(P@$$w0rdhJ33fqAwscJmp3MacoQ8uO)

==

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

Database

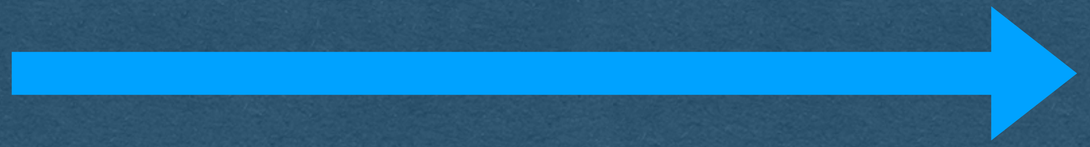
Registration and Login

- Discard the plain text password
- Store username, salt, and salted hash in DB

User/Client

Server

username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8



username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
password hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8

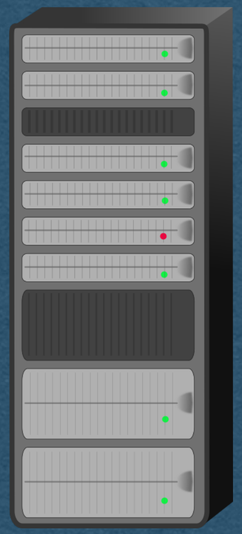
Database

Registration and Login

- When using bcrypt, salt and hash are stored as a single value

username
XX_UBStudent_XX
bcrypt salt + hash
$2b$12$9IoK6aMp5snMH2FvZ8rcWexyveqs8mFKojG7JvqVhfRxSDmwfAZHW

User/Client



Server

Database

Registration and Login

- User wants to login
- Provide username and password

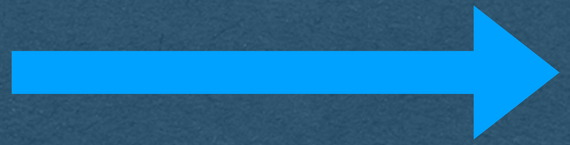
User/Client

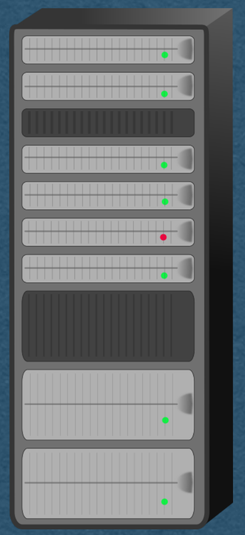
username

XX_UBStudent_XX

password

P@$$w0rd





Server

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- User wants to login
- Provide username and password

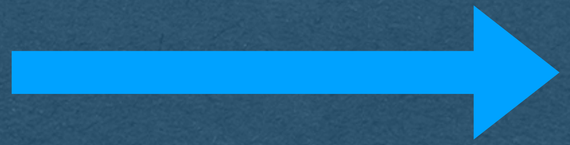
User/Client

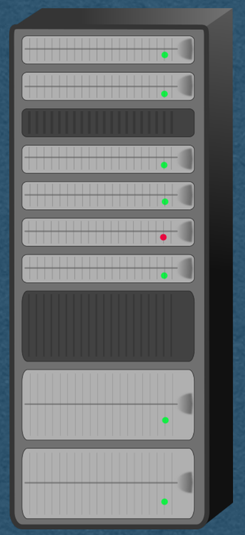
username

XX_UBStudent_XX

password

P@$$w0rd





Server

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- Server pulls the hash and salt for this username

User/Client

Server

find({"username", "XX_UBStudent_XX"})

username
XX_UBStudent_XX
password
P@$$w0rd

username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8

Database

username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
password hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8

Registration and Login

- Server now has everything it needs for authentication

User/Client

Server

username

XX_UBStudent_XX

password

P@$$w0rd

salt

hJ33fqAwscJmp3MacoQ8uO

hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- Append the salt to the password provided at login and hash

User/Client

Server

username

XX_UBStudent_XX

password

P@$$w0rd

salt

hJ33fqAwscJmp3MacoQ8uO

new hash

f(P@$$w0rdhJ33fqAwscJmp3MacoQ8uO)

==

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- If the two hashes do not match exactly, the user is not authenticated
- This password is changed to be 1 char off
  - The hashes are completely different

User/Client

Server

username

XX_UBStudent_XX

password

Pa$$w0rd

salt

hJ33fqAwscJmp3MacoQ8uO

new hash

f(Pa$$w0rdhJ33fqAwscJmp3MacoQ8uO)

==

296069c8595a636361af0c65acfd819178ee1f9ce235c554ce1a0c37598138f5

hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- If the two hashes match, the user is authenticated as this user
- Server is done with all values related to the password

User/Client

Server

username

XX_UBStudent_XX

password

P@$$w0rd

salt

hJ33fqAwscJmp3MacoQ8uO

new hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

Database

Registration and Login

- Generate an authentication token
- Store a **hash** of the token in your DB

User/Client

Server

username

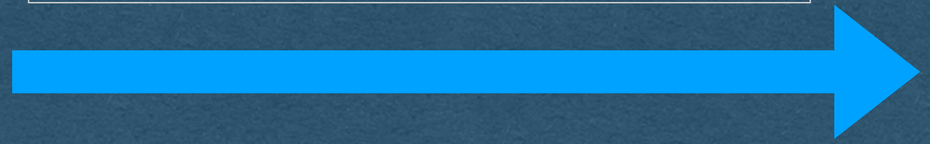
XX_UBStudent_XX

authentication token

jixFFgT1xPhXKcLrOavlQO

update XX_UBStudent_XX's record with

{"hashed authentication token":
"754b1fb1b5ab6787441bfb410a40a7b99
29f712497fb1f57788f4a6b699e1d7b"}



Database

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

hashed authentication token

754b1fb1b5ab6787441bfb410a40a7b
9929f712497fb1f57788f4a6b699e1d7b

Registration and Login

- Set the plain text of the token to a cookie
- The user is now logged in

<u>username</u>

XX_UBStudent_XX

<u>salt</u>

hJ33fqAwscJmp3MacoQ8uO

<u>password hash</u>

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

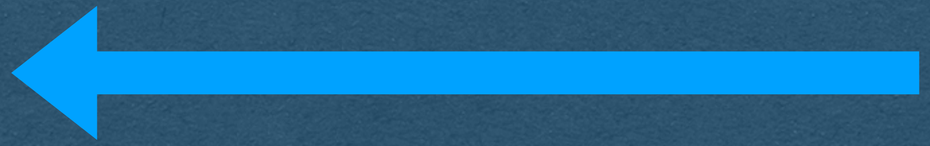
<u>hashed authentication token</u>

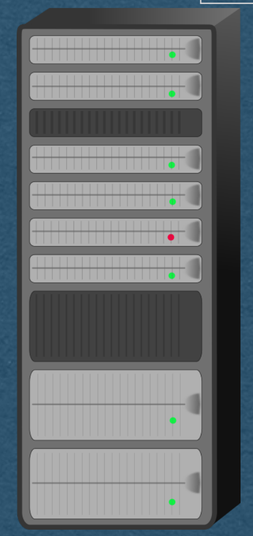
754b1fb1b5ab6787441bfb410a40a7b
9929f712497fb1f57788f4a6b699e1d7b

User/Client

<u>Set authentication cookie</u>

jixFFgT1xPhXKcLrOavlQO





Server

Database

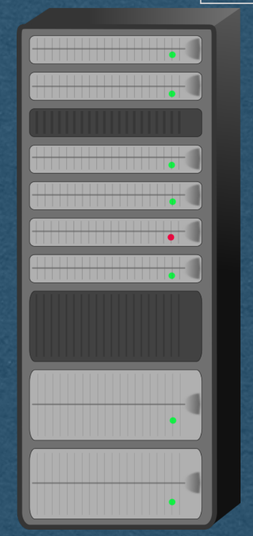
Registration and Login

- Token is sent on all subsequent requests

User/Client

Request containing authentication cookie

jixFFgT1xPhXKcLrOavlQO



Server

Database

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fd
d8b86b84a94b85c7cb91e9e10008ec8

hashed authentication token

754b1fb1b5ab6787441bfb410a40a7b
9929f712497fb1f57788f4a6b699e1d7b

Registration and Login

- Read the token from the cookie and hash it
- If not cookie exists, the user is not logged in

User/Client

Server

token to verify

jixFFgT1xPhXKcLrOavlQO

hash of token to verify

754b1fb1b5ab6787441bfb410a40a7b9929f712497fb1f57788f4a6b699e1d7b

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

hashed authentication token

754b1fb1b5ab6787441bfb410a40a7b9929f712497fb1f57788f4a6b699e1d7b

Database

Registration and Login

- Look up the hash in the DB
- If a record is returned, that's the logged in user and request is authenticated

find({"hashed authentication token": "754b1fb1b5ab6787441bfb410a40a7b9929f712497fb1f57788f4a6b699e1d7b"})

User/Client

Server

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

hashed authentication token

754b1fb1b5ab6787441bfb410a40a7b9929f712497fb1f57788f4a6b699e1d7b

Database

username

XX_UBStudent_XX

salt

hJ33fqAwscJmp3MacoQ8uO

password hash

0144c52c5b68f662f4529fd43093873fdd8b86b84a94b85c7cb91e9e10008ec8

hashed authentication token

754b1fb1b5ab6787441bfb410a40a7b9929f712497fb1f57788f4a6b699e1d7b

Registration and Login

- We now have the verified username of the requester

User/Client

Server

username
XX_UBStudent_XX

Database

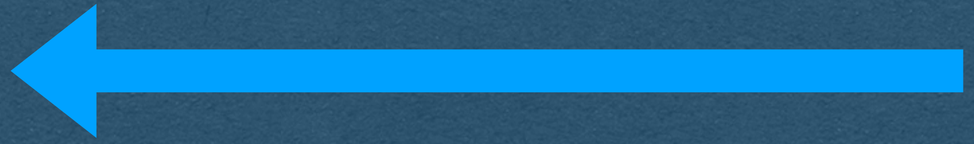
username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
password hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8
hashed authentication token
754b1fb1b5ab6787441bfb410a40a7b 9929f712497fb1f57788f4a6b699e1d7b

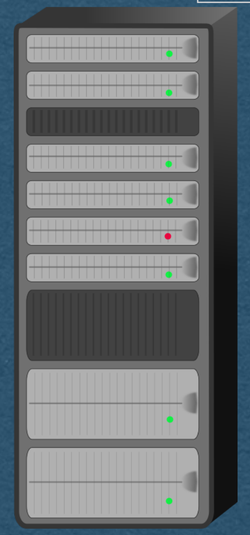
Registration and Login

- We now have the verified username of the requester

username
XX_UBStudent_XX
salt
hJ33fqAwscJmp3MacoQ8uO
password hash
0144c52c5b68f662f4529fd43093873fd d8b86b84a94b85c7cb91e9e10008ec8
hashed authentication token
754b1fb1b5ab6787441bfb410a40a7b 9929f712497fb1f57788f4a6b699e1d7b

Response that can contain XX_UBStudent_XX's private data





User/Client

Server

Database

Cookie Hijacking

- We're now using cookies for authentication
- The possession of the token verifies that this user did authenticate in the past

- What if someone steals your cookies?
  - They can authenticate as you without needing your password