COMP 431/531: Web Development

Lecture 18: Authorization

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431

Announcements & Reminders

- HW #5 (Front-end) due Thu. Nov. 6th at 11:59pm

  classroom hw5 repo: https://classroom.github.com/a/hIFk8cgA

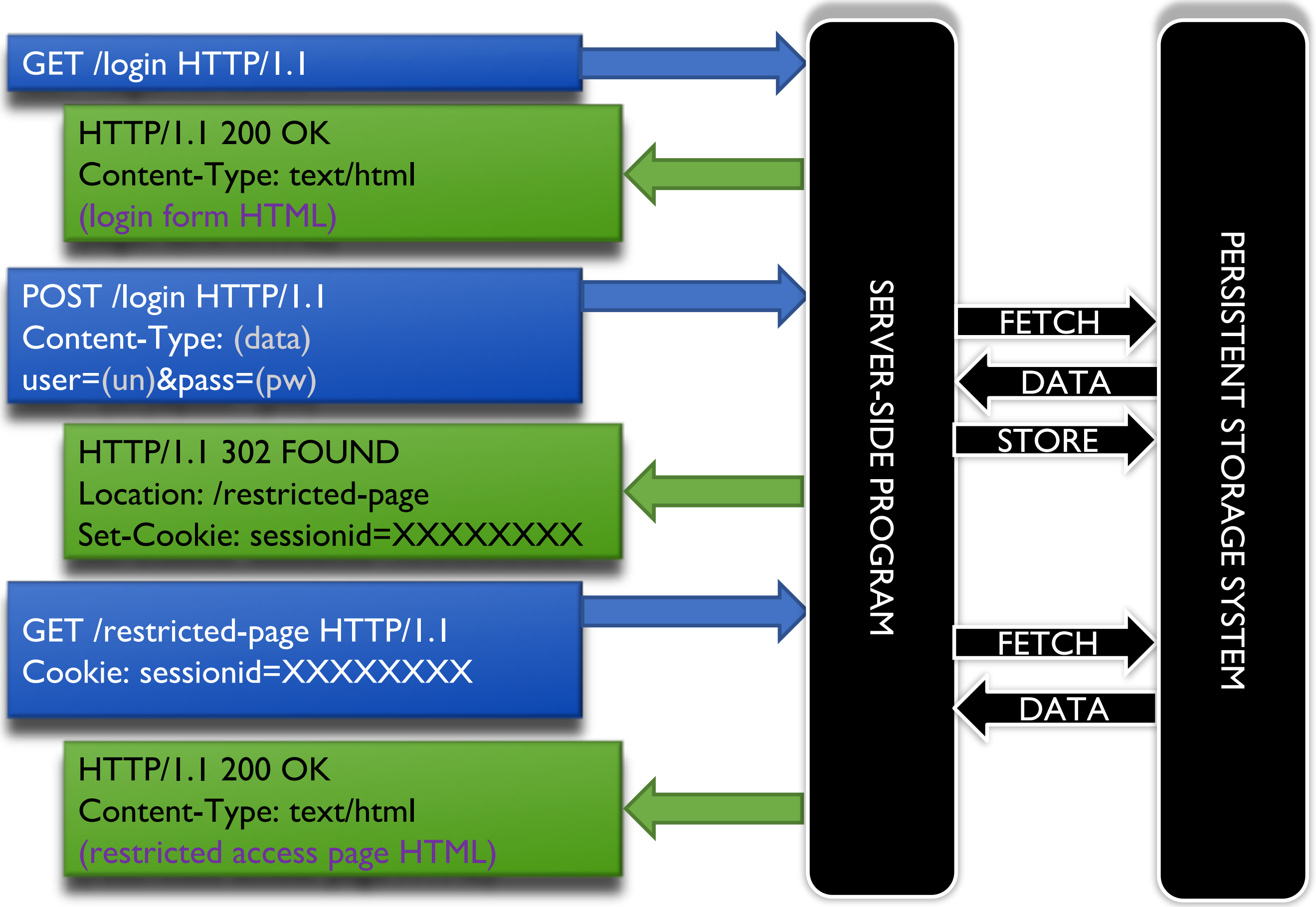
- Comp 531 Oral Presentations will be on 11/25, 12/2, and 12/4

  https://classroom.github.com/a/_IUk3HZa

Comp 531 Presentation

- Topic
  - New technology
  - Technology comparison
  - E-commerce
  - Security
  - Big Data
- Presentation
  - Give a 7 minute group talk (2 members per group)
  - Slides, websites, demo, etc…

Authorization



Note: "sessionid" will be attached to all future requests

Acknowledgment: figure courtesy of Matthew Schurr

Cookies

- POST /login

  { username & password }

in plain sight!

- Server returns a cookie

- Browser "eats" cookie, returns it with subsequent requests
- PUT /logout, server returns empty cookie for browser to eat

Authenticate with Cookie Parser In Node

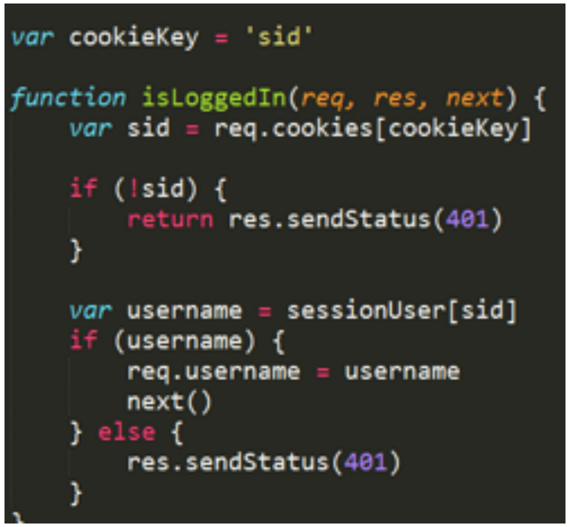
>> npm install cookie-parser --save

index.js

const cookieParser = require('cookie-parser');

app.use(cookieParser());

auth.js



```
var cookieKey = 'sid'

function isLoggedIn(req, res, next) {
    var sid = req.cookies[cookieKey]

    if (!sid) {
        return res.sendStatus(401)
    }

    var username = sessionUser[sid]
    if (username) {
        req.username = username
        next()
    } else {
        res.sendStatus(401)
    }
```

Mini Exercise

- Install cookie-parser, add it as middleware to index.js
- Download https://www.clear.rice.edu/comp431/sample/RiceBookServer/src/auth.js , place in backend/src
- Add const auth = require('./src/auth.js') in index.js
- In index.js, pass express app to auth so isLoggedIn is middleware for all endpoints except GET '/'
- Start server (node index.js)
- Go to http://localhost:3000/ and test that you get:
  - No error for: GET '/' (returns {"hello":"world"})
  - Authorization error for GET '/articles' and GET '/articles/1'

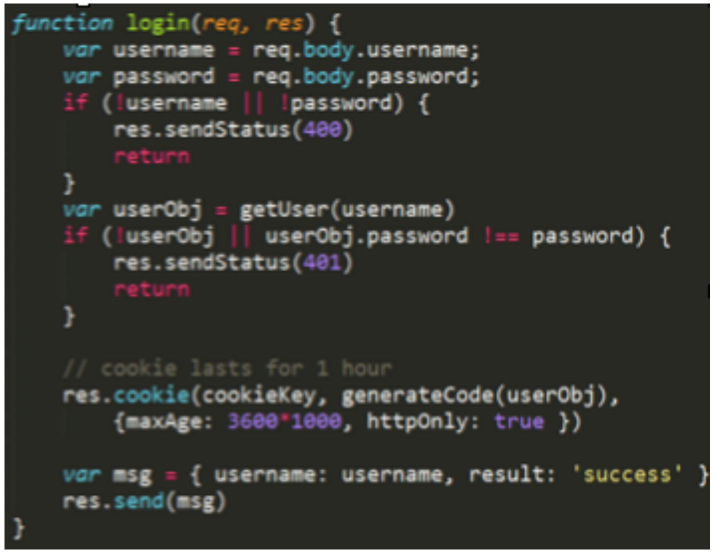
Log In User

- Ensure user supplied username, password (error: Bad Request)

- Get object (e.g. document) given username (error: Unauthorized)

- sendStatus, shortcut for res.status(401).send('Unauthorized')

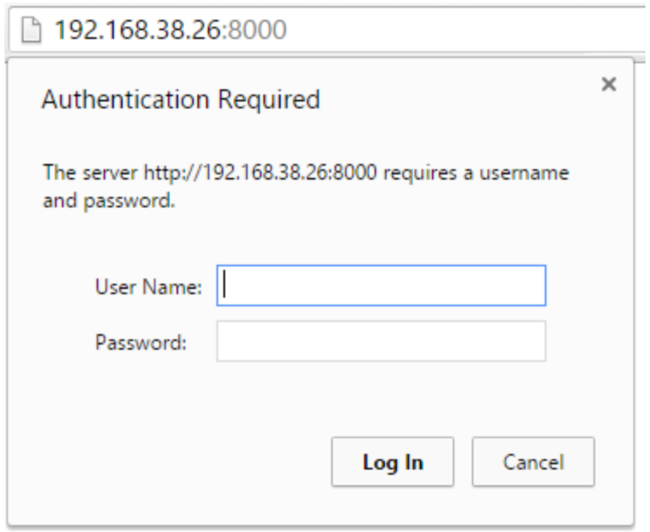
- Create cookie, set in response



```
function login(req, res) {
    var username = req.body.username;
    var password = req.body.password;
    if (!username || !password) {
        res.sendStatus(400)
        return
    }
    var userObj = getUser(username)
    if (!userObj || userObj.password !== password) {
        res.sendStatus(401)
        return
    }

    // cookie lasts for 1 hour
    res.cookie(cookieKey, generateCode(userObj),
        {maxAge: 3600*1000, httpOnly: true })

    var msg = { username: username, result: 'success' }
    res.send(msg)
}
```

Http Auth

- User makes request without Authorization
- Server responds with 401, sets WWW-Authenticate with a "challenge"
- User attempts challenge by filling in username and password
- Server accepts or issues another challenge

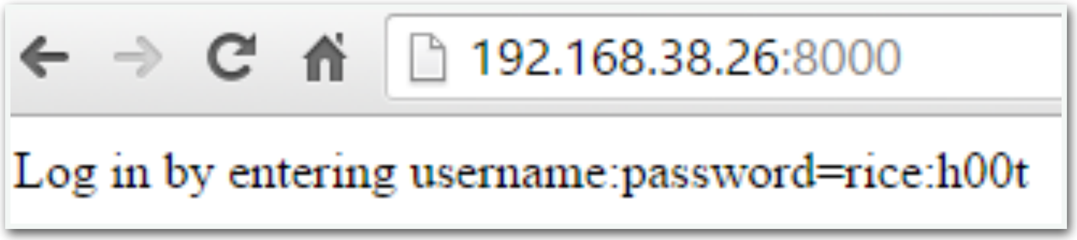

192.168.38.26:8000

Authentication Required

The server http://192.168.38.26:8000 requires a username and password.

User Name:

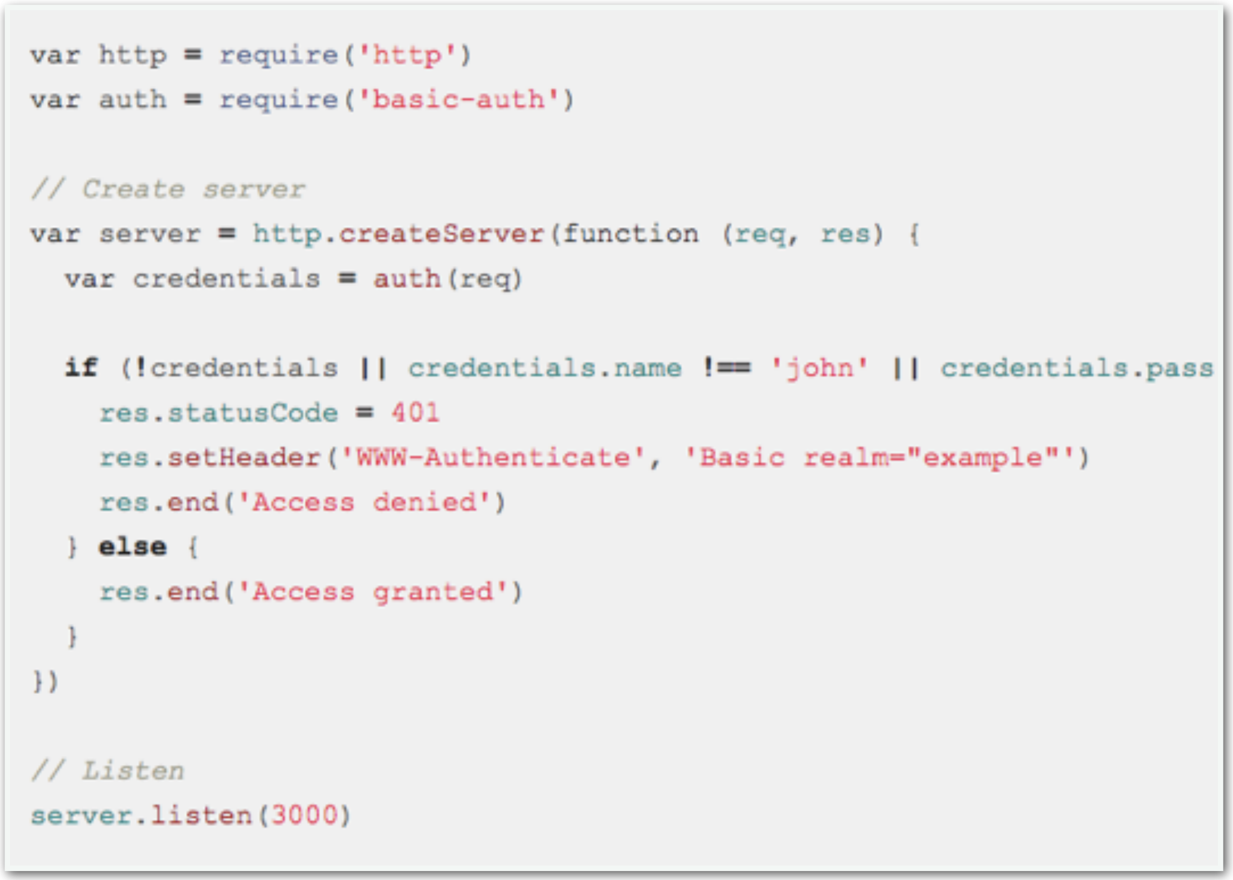
Password:


192.168.38.26:8000

Log in by entering username:password=rice:h00t

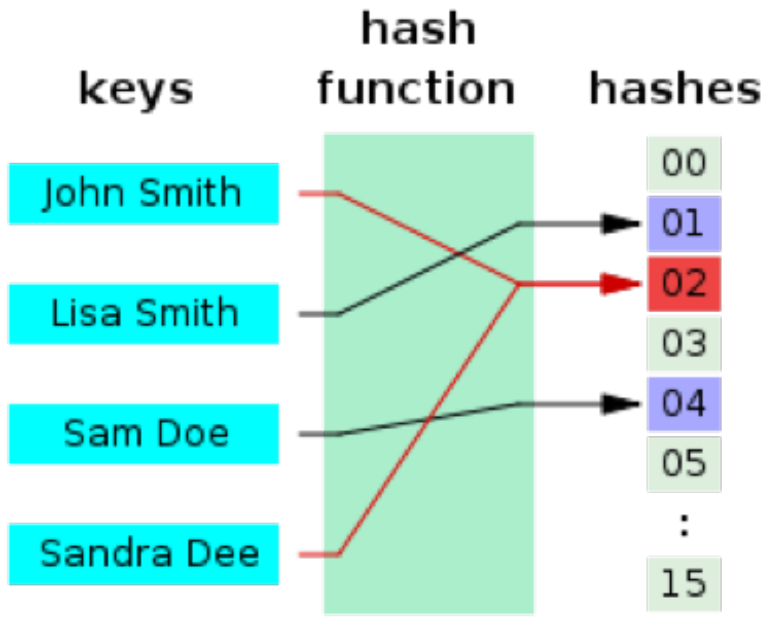
Basic Auth Node Module

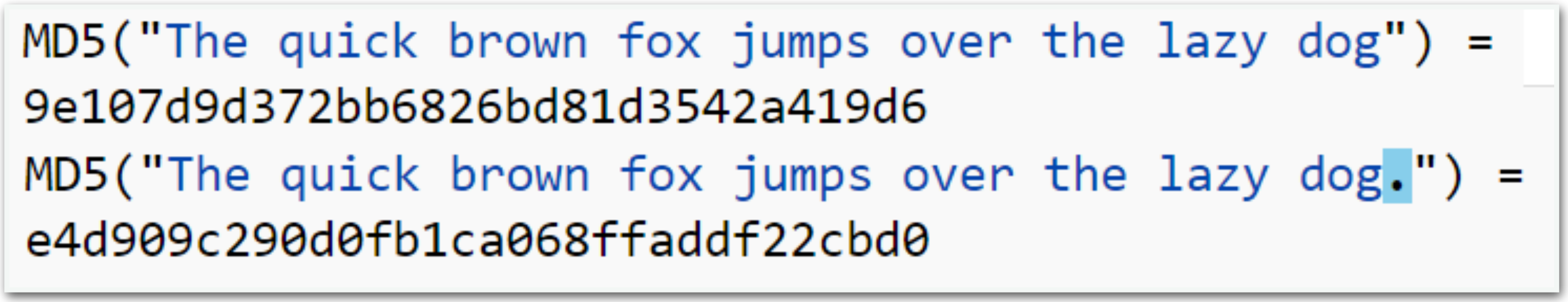
>> npm install basic-auth --save



```
var http = require('http')
var auth = require('basic-auth')

// Create server
var server = http.createServer(function (req, res) {
  var credentials = auth(req)

  if (!credentials || credentials.name !== 'john' || credentials.pass
    res.statusCode = 401
    res.setHeader('WWW-Authenticate', 'Basic realm="example"')
    res.end('Access denied')
  } else {
    res.end('Access granted')
  }
})

// Listen
server.listen(3000)
```

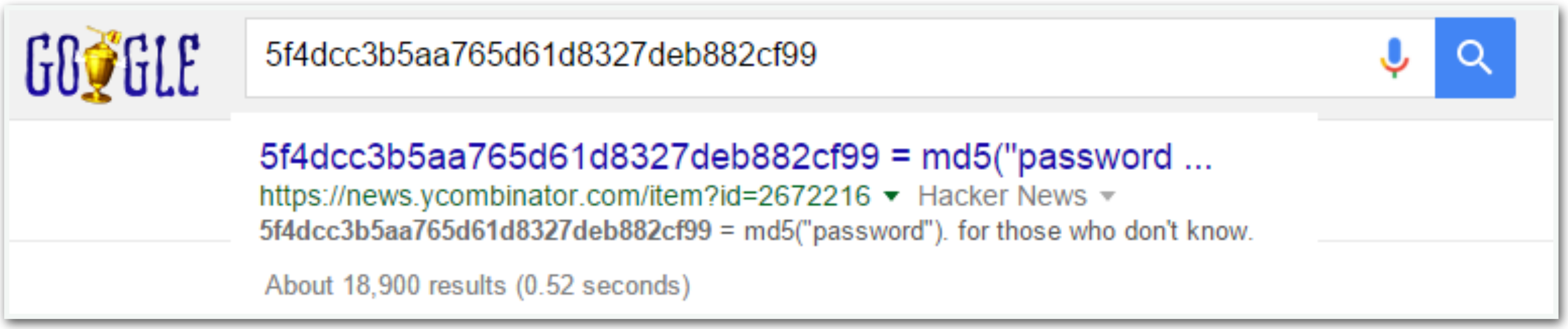
Hashing





```
MD5("The quick brown fox jumps over the lazy dog") =
9e107d9d372bb6826bd81d3542a419d6
MD5("The quick brown fox jumps over the lazy dog.") =
e4d909c290d0fb1ca068ffaddf22cbd0
```

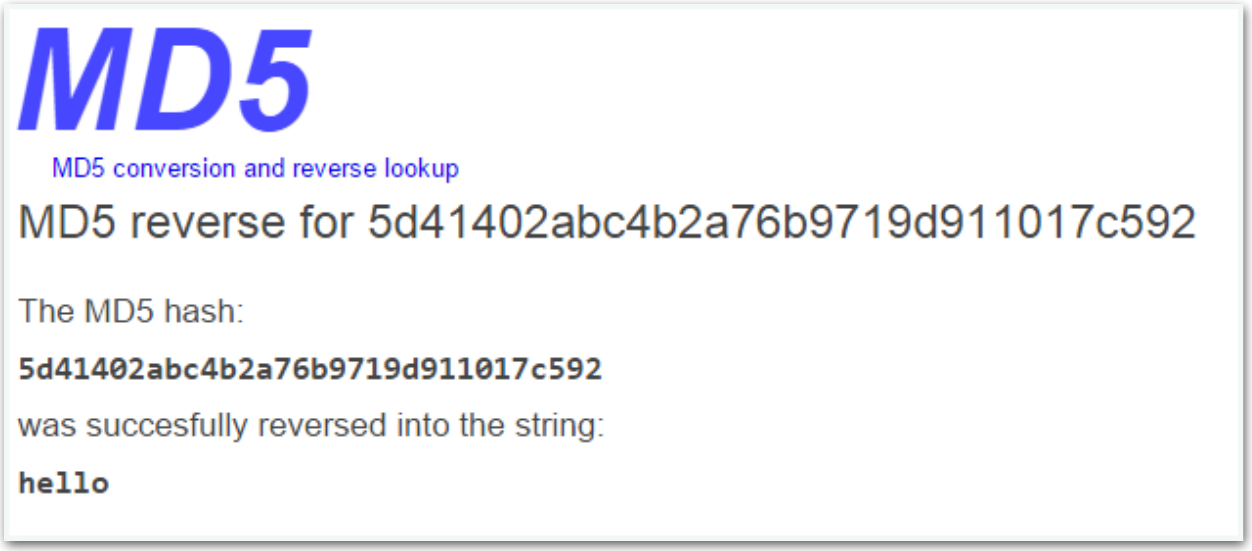
Hash Lookup



5f4dcc3b5aa765d61d8327deb882cf99

5f4dcc3b5aa765d61d8327deb882cf99 = md5("password ...
https://news.ycombinator.com/item?id=2672216 ▾ Hacker News ▾
5f4dcc3b5aa765d61d8327deb882cf99 = md5("password"). for those who don't know.

About 18,900 results (0.52 seconds)


MD5

MD5 conversion and reverse lookup

MD5 reverse for 5d41402abc4b2a76b9719d911017c592

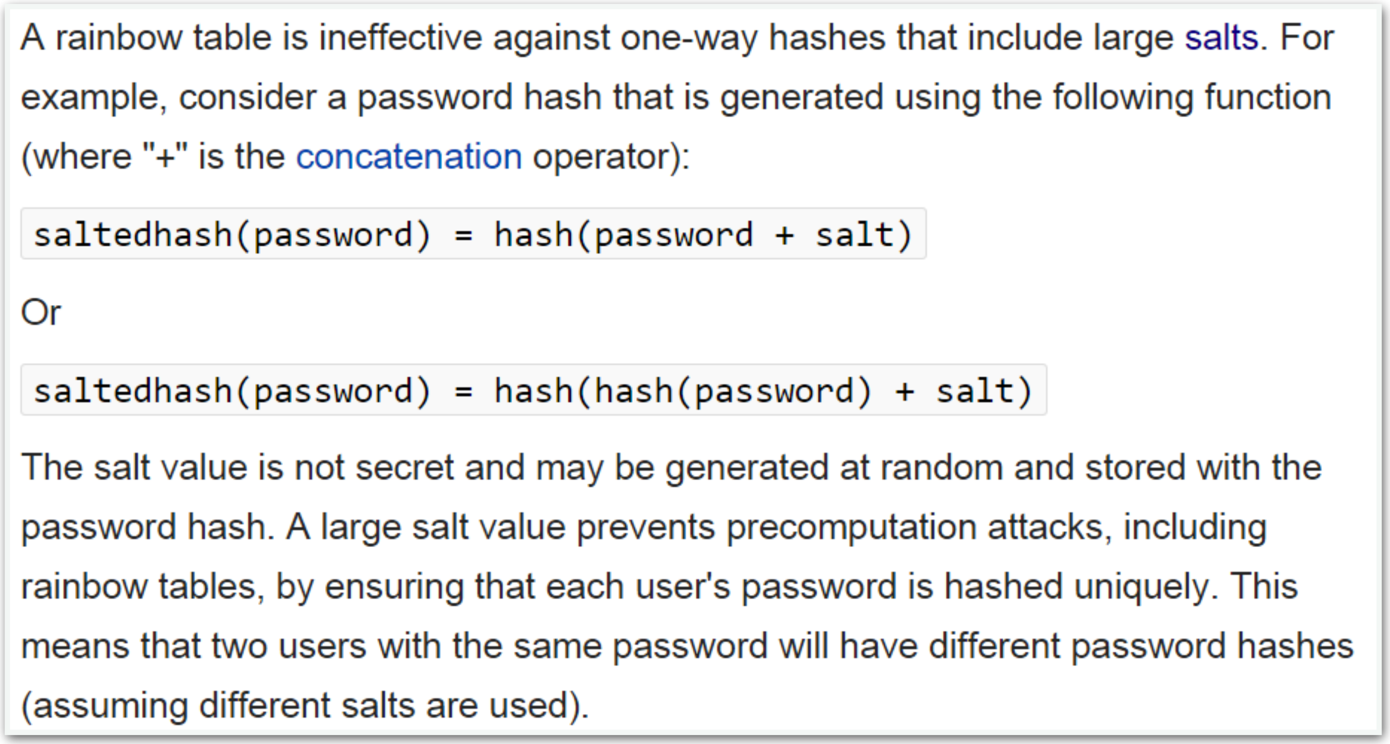
The MD5 hash:

5d41402abc4b2a76b9719d911017c592

was succesfully reversed into the string:

hello

Defense: Salting



A rainbow table is ineffective against one-way hashes that include large salts. For example, consider a password hash that is generated using the following function (where "+" is the concatenation operator):

```
saltedhash(password) = hash(password + salt)
```

Or

```
saltedhash(password) = hash(hash(password) + salt)
```

The salt value is not secret and may be generated at random and stored with the password hash. A large salt value prevents precomputation attacks, including rainbow tables, by ensuring that each user's password is hashed uniquely. This means that two users with the same password will have different password hashes (assuming different salts are used).

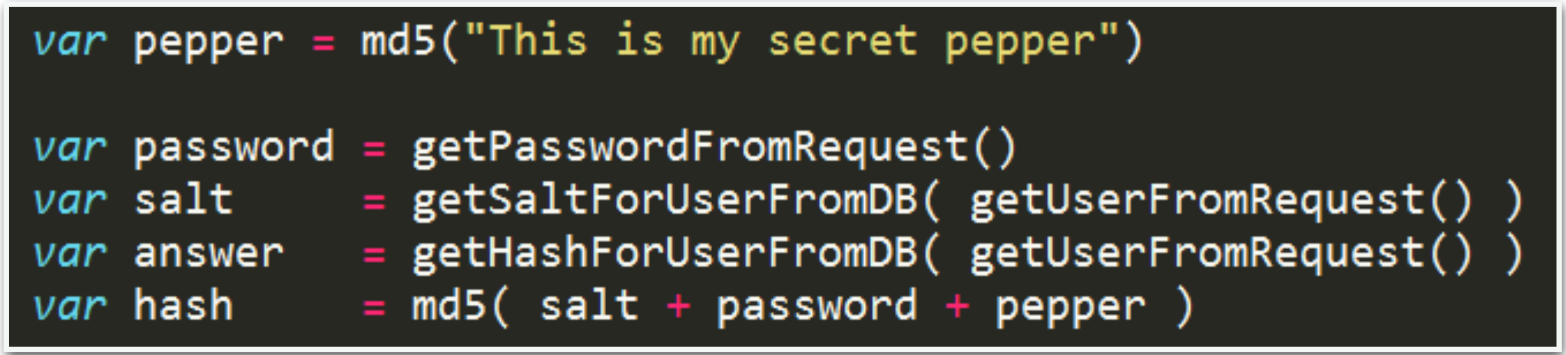
See: https://en.wikipedia.org/wiki/Rainbow_table

Salted Passwords

- Pre-Salt plan of attack
  - Create lookup table of every n-character password to hash (slow)
- The salt is typically public
  - Larger n-character lookup table
- Salted plan of attack
  - Take the salt, generate a table from it (eventually will get in)

Peppering

- Note that there's a different salt for each user
- Salt is in the database
- If database is compromised, attacker can get it by making a lookup table
- Pepper is a secret code on the server, not in the database



```
var pepper = md5("This is my secret pepper")

var password = getPasswordFromRequest()
var salt     = getSaltForUserFromDB( getUserFromRequest() )
var answer   = getHashForUserFromDB( getUserFromRequest() )
var hash     = md5( salt + password + pepper )
```

Security

- Reduce likelihood of hack
  - Hash on the browser? Sure.
  - Hash on the server? Definitely.
- MD5 and SHA-I are now “trivial”, company wouldn’t use in production
  - H(H(H(H(…..H(password + salt) + salt) + salt)….)
- Can use a Key-Derivative Function such as bcrypt/scrypt