CSE 106

Lecture 16 – Web Security Basics

Acknowledgement:

https://www.sohamkamani.com/web-security-basics

https://martinfowler.com/articles/web-security-basics.html

https://www.freecodecamp.org/news/md5-vs-sha-1-vs-sha-2-which-is-the-most-secure-encryption-hash-and-how-to-check-them

https://owasp.org/www-community/attacks/xss

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Web Security Basics

- HTTPS and Transport Layer Security (TLS)
- Session Tokens
- Password Storage
- Cross Site Scripting (XSS)
- Cross Site Request Forgery (CSRF)
- SQL Injection
- Cross Origin Resource Sharing (CORS)
- Brute Force Attack

HTTPS and Transport Layer Security (TLS)

- Most websites are served over HTTPS so that data transfer is secure
- HTTPS protocol uses the TLS protocol to secure communications
  - TLS is the successor to the Secure Sockets Layer (SSL) protocol
- When configured and used correctly it provides:
  - protection against eavesdropping and tampering
  - a reasonable guarantee that a website is the one we intend to be using

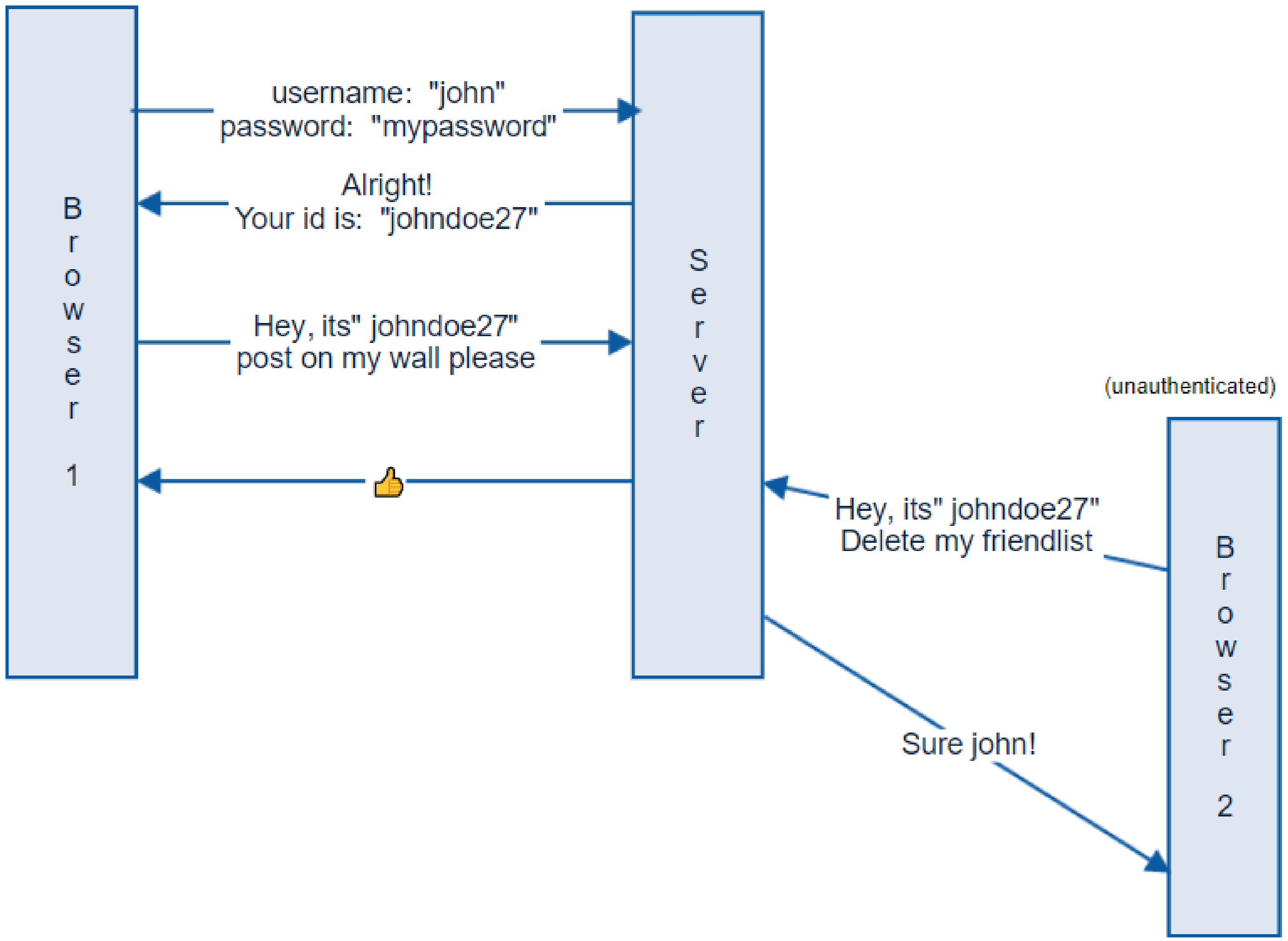
How TLS works

- When using TLS, a site proves its identity using a public key certificate
    - contains information about the site along with a public key that is used to prove that the site is the owner of the certificate
    - Certificates are verified using a trusted third party called a Certificate Authority (CA)
- TLS uses public key encryption to encode and decode data
- Public key encryption explained

Session Tokens

- If authentication is involved, sessions and cookies are needed so the user doesn’t need to reauthenticate with every request
- Because servers are stateless, the browser needs to store something about the user so they remember who they are
- Don’t just store the username, this isn’t enough

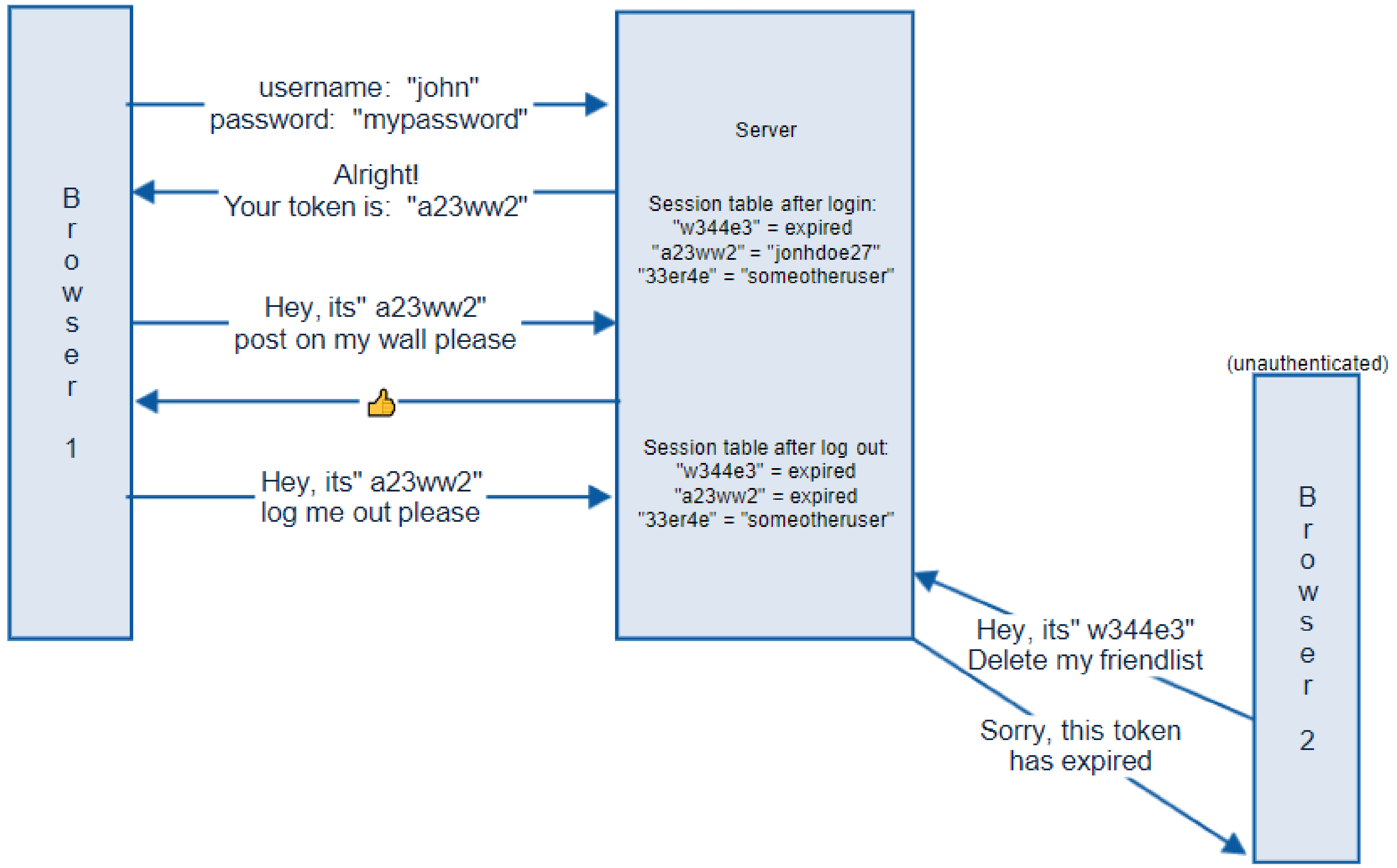
Session Tokens



Session Tokens

- A session token is a unique identifier that the client stores in cookies
- The token is assigned when a user logs in with their credentials
- This token is used when sending requests
- The token expires after the user logs out or when the token expires, after a period of time

Session Tokens



Password Storage

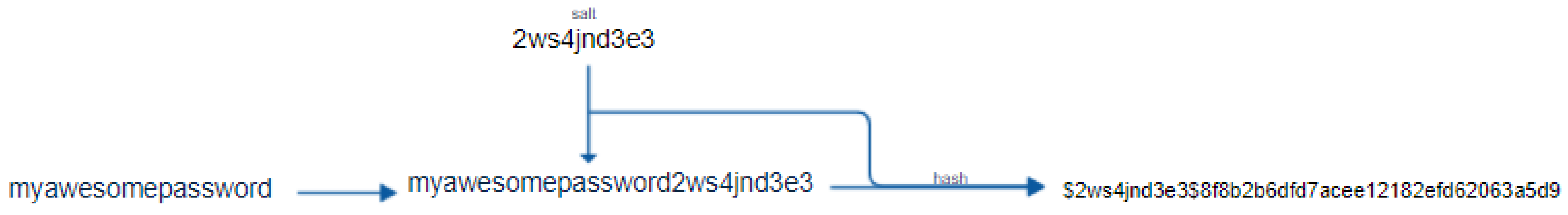
- Never store passwords in plain text! This is not secure.
- Always hash passwords in a database so not even you know what it is
- One way hashing
  - Use an established irreversible encryption algorithm (SHA-256 recommended)
  - The hash values of inputs are spread evenly and unpredictably across the whole range of possible hash values
  - Examples:
    - "The quick brown fox." -> 2e87284d245c2aae1c74fa4c50a74c77
    - "The quick brown fax." -> c17b6e9b160cda0cf583e89ec7b7fc22

One Way Hashing with Salt

- Hashing is not enough because common passwords can be identified with reverse lookup table
- Some common words with md5 hashes
  - lettuce : 8cbd191432b5f52b48497313f966a4f8
  - cat : d077f244def8a70e5ea758bd8352fcd8
  - bottle : 3a385ac07dcec4dde1a4ca47a9802c96
- https://md5.gromweb.com

One Way Hashing with Salt

- “Salting” means to add a random string of letters and numbers to a users password, and then hash it
- This more or less guarantees that the word is unique, and therefore cannot be part of a lookup table


salt
2ws4jnd3e3
myawesomepassword
myawesomepassword2ws4jnd3e3
hash
$2ws4jnd3e3$8f8b2b6dfd7acee12182efd62063a5d9

Password validation

- When a user enters a password on login, we can validate it by following these steps:
  1. Get the hashed + salted password from our data store
  2. Read the salt from the stored data (the part between the $ symbols)
  3. Append the salt to the entered password, and hash the result
  4. Compare the output of this hash to the part on the right of the second $ symbol

Wake-up!

- https://youtu.be/2W0aofVe-70

Cross Site Scripting (XSS)

- A web attacker uses a web app to send malicious code in the form of a browser side script
- Example:
  - The div below shows your status and is updated by you
  - A malicious user changes status from “I am feeling alright” to “\<script>alert(‘You will die tomorrow!’)\</script>”
  - Now it shows an alert with a death threat

```
<div id="status">
  I am feeling alright
</div>
```

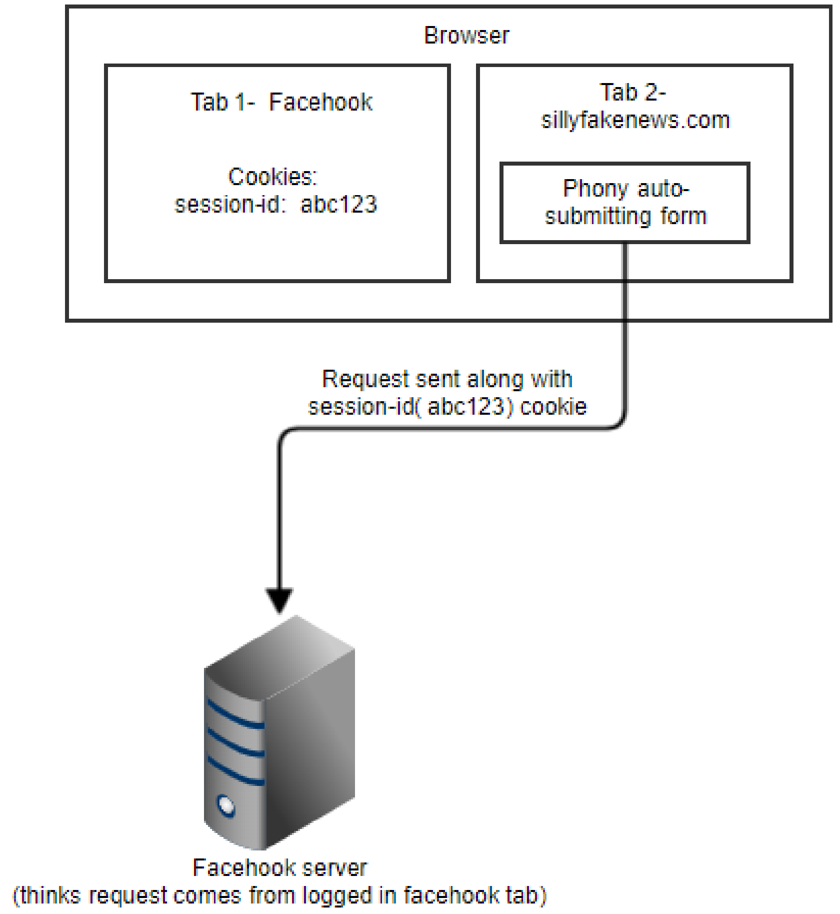
```
<div id="status">
  <script>
    alert(‘You will die tomorrow!’)
  </script>
</div>
```

Cross Site Scripting (XSS)

- Prevent XSS by:
  - Sanitizing your inputs before using them
    - Make sure all info being rendered on the browser is HTML encoded first
    - Remove certain characters or keywords
  - Don’t render input directly
  - Use a frontend framework (React, Angular, Vue)

Cross Site Request Forgery (CSRF)

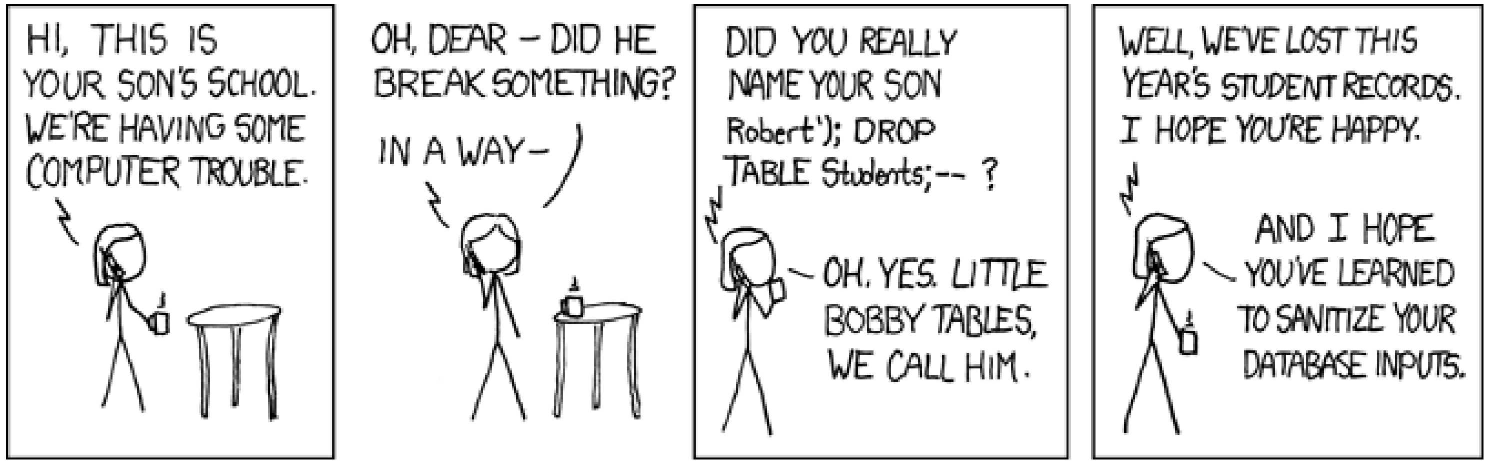
- An attack where a request coming from one website is disguised to seem like it's coming from another
- Leverage session tokens, stored in cookies, which are sent in requests
- **Cross site:** coming from a site other than the one it is intended for
- **Request forgery**: Sending a request which appears to be legitimate but is actually malicious


Browser
Tab 1- Facehook
Cookies: session-id: abc123
Tab 2- sillyfakenews.com
Phony auto-submitting form
Request sent along with session-id(abc123) cookie
Facehook server
(thinks request comes from logged in facehook tab)

Cross Site Request Forgery (CSRF)

- Prevent CSRF by:
  - Block requests from other sites (CORS policy)
  - CSRF Tokens
    - A unique CSRF token is assigned for every user session
    - Not transmitted via a cookie, but received through a response and sent via requests
    - Prevent CSRF because an attacker doesn’t naturally have access to the token and can’t create a valid request without one

SQL Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

```
void addStudent(String lastName, String firstName) {
    String query = "INSERT INTO students (last_name, first_name)
                    VALUES ('" + lastName + "', '" + firstName + “’)”;
    getConnection().createStatement().execute(query); }
```

```
INSERT INTO students (last_name, first_name) VALUES ('Fowler', 'Martin’)
INSERT INTO students (last_name, first_name) VALUES ('XKCD', 'Robert’); DROP TABLE Students;-- ')
```

SQL Injection

- Prevent SQL Injection by:
  - Sanitize all your inputs
    - Escape all your SQL queries - This means replacing special characters like ` , " with their escaped versions (i.e \` and \")
  - Use an ORM instead of writing your own queries
  - Use a NoSQL Database

Cross Origin Resource Sharing (CORS)

- Browsers enforce a same origin policy:
  - It is forbidden to make a request to any origin, other than the one your code is running in, unless otherwise allowed by its server
- Response headers control this access

```
Access-Control-Allow-Origin: https://developer.mozilla.org
Access-Control-Allow-Methods: POST, GET, OPTIONS
```

Cross Origin Resource Sharing (CORS)

- Why can’t these requests be made from the browser, when they can be easily made through third party applications like postman or curl?
  - CORS isn’t actually enforced by the server, but rather the browser
- If a server really does not want other clients to receive a response, it can disable CORS for non-browser clients as well
  - This means that you can only make requests from the same origin, and that tools like postman and curl can’t make requests either

Brute Force Attack

- An attacker attempts to guess a password by submitting it again and again
- An attacker wants to take you down so they create a lot of meaningless requests
- Prevent brute force attacks:
  - Rate limiter on attempts by a given user or ip address – locks account for a few minutes
  - CAPTCHAS – presents a challenge a human can solve, but a computer can’t
  - Combine the two methods