Enterprise Programmering 1

Lesson 10: Security

Prof. Andrea Arcuri

Authentication/Authorization

- **Authentication**:
  - do I know who a user X is?
  - how to distinguish X from a different user Y?
- **Authorization**:
  - once I know that the current user is X, what is X allowed to do?
  - can s/he delete data?
  - can s/he see data of other users?
  - etc.

Authentication/Authorization failures

- If not authenticated, server can:
  - redirect to login page, HTTP status code 3xx
  - error page, HTTP status 401 *Unauthorized*
- If authenticated but not authorized
  - eg user X tries to access data of Y
  - 3xx redirection
  - HTTP status 403 *Forbidden*

Blacklisting vs Whitelisting

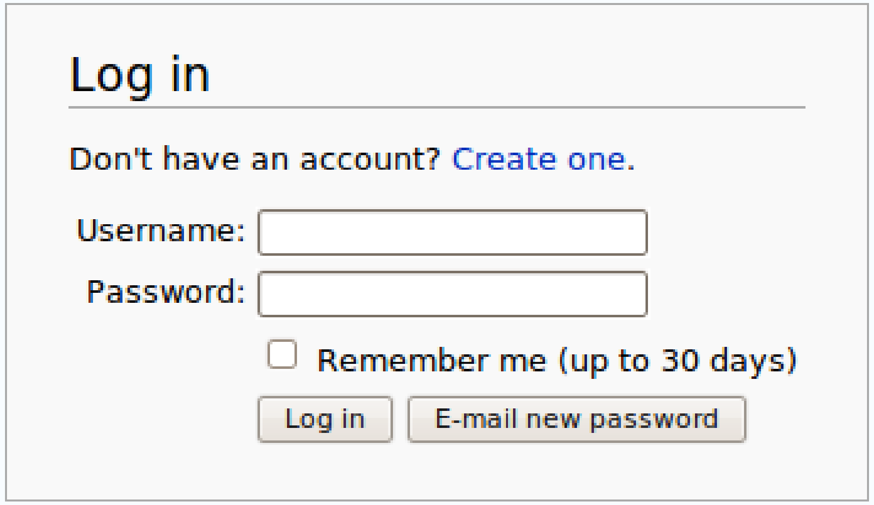
- Authorization is done on the server, and will depend on the language/framework
  - JEE, Spring, .Net, NodeJS, etc.
  - user will just get either a 3xx or 403 response
- *Blacklisting*: everything is allowed by default. What is not allowed for a given user/group has to be explicitly stated
  - Usually not a good idea, as easy to forget to blacklist some critical operation
- *Whitelisting*: nothing is allowed by default. What is allowed has to be explicitly stated
  - “forgetting to allow something” (reduced functionality) is much, much better than “forgetting to forbid something” (security problem)

Authentication: first steps

- Server does not know who the user is
- Server only sees incoming HTTP/S messages
  - not necessarily from a browser… user can do direct TCP connections from scripts
- HTTP/S is stateless
- Need a way to tell that sequence of HTTP/S calls come from same user
- User has to send information of who s/he is at EACH HTTP/S call
- But users can **lie**… (eg, hackers)

Ids and Passwords

- A user will be registered with a *unique* id
- Need also secret password to login
  - Otherwise anyone could login with the ids of other users…
- HTTP/S does not prevent attempts to login to accounts of other users



Log in

Don't have an account? Create one.

Username:

Password:

☐ Remember me (up to 30 days)

How to implement a login mechanism?

- When talking about security and what to implement on the server, think about HTTP/S messages, not necessarily coming from browsers
- Could have endpoint to get *token* from server given userId/password
  - Use such token on each following request as parameter
- GET /login?**userId=x**&**password=y**
  - userId/password as URL parameters to the /login endpoint
  - get back new token Z associated to this user, as HTTP/S response body, no HTML page
- GET /somePageIWantToBrowse**?token=z**
  - pass “token=z” parameter to each HTTP/S request

Awful Solution

- That solution would work with HTTPS, but…
- “*/login?userId=x&password=y*” would be cached in your browser history, even after you logout
- How to handle the adding of “*?token=z*” to all your *<a>* tags in the HTML pages?
  - doable, but quite cumbersome
- How to handle browser bookmarks?
  - tokens would be there, and made the links useless once they expire, eg after a logout

POST and Cookies

- User ids and passwords should never be sent with a GET
  - GET specs do not allow body in the requests
- Should be in HTTP body of a POST
  - This is also default behavior of <*input*> forms in HTML
- Authentication “tokens” should not be in URLs, but in the HTTP Headers
- **Cookie**: special header that will be used to identify the user
- The user does not choose the cookie, it is the server that assigns them
- Recall: user can craft its own HTTP messages, so server needs to know if cookie values are valid

Login with Cookies

- Browser: POST /login
  - Username X and password as HTTP body
- Server: if login is successful, respond to the POST with a “*Set-Cookie*” header, with some unique identifier Y
  - Server needs to remember that cookie Y is associated with user X
  - *Set-Cookie: <cookie-name>=<cookie-value>*
- Browser: from now on, each following HTTP request will have “*Cookie: Y*” in the headers
- Logout: remove association between cookie Y and user X on server.
- Server: HTTP request with no cookie of invalid/expired cookie, do 3xx redirect to login page

Cookies and Sessions

- Servers would usually send a “*Set-Cookie*” regardless of login
  - want to know if requests are coming from same users, regardless if s/he is registered/authenticated
  - ie cookies used to define “sessions”
- After login could create a new session (ie, invalidate old cookie and create a new one) or use the existing session cookie (eg, the one set by the server when login page was retrieved with the first GET)
- Problem with re-using session cookies: make sure all the pages were served with HTTPS and not HTTP
  - ie, use HTTPS for all pages, even the login one
  - do not use HTTP and then switch to HTTPS once login is done

Storing cookies

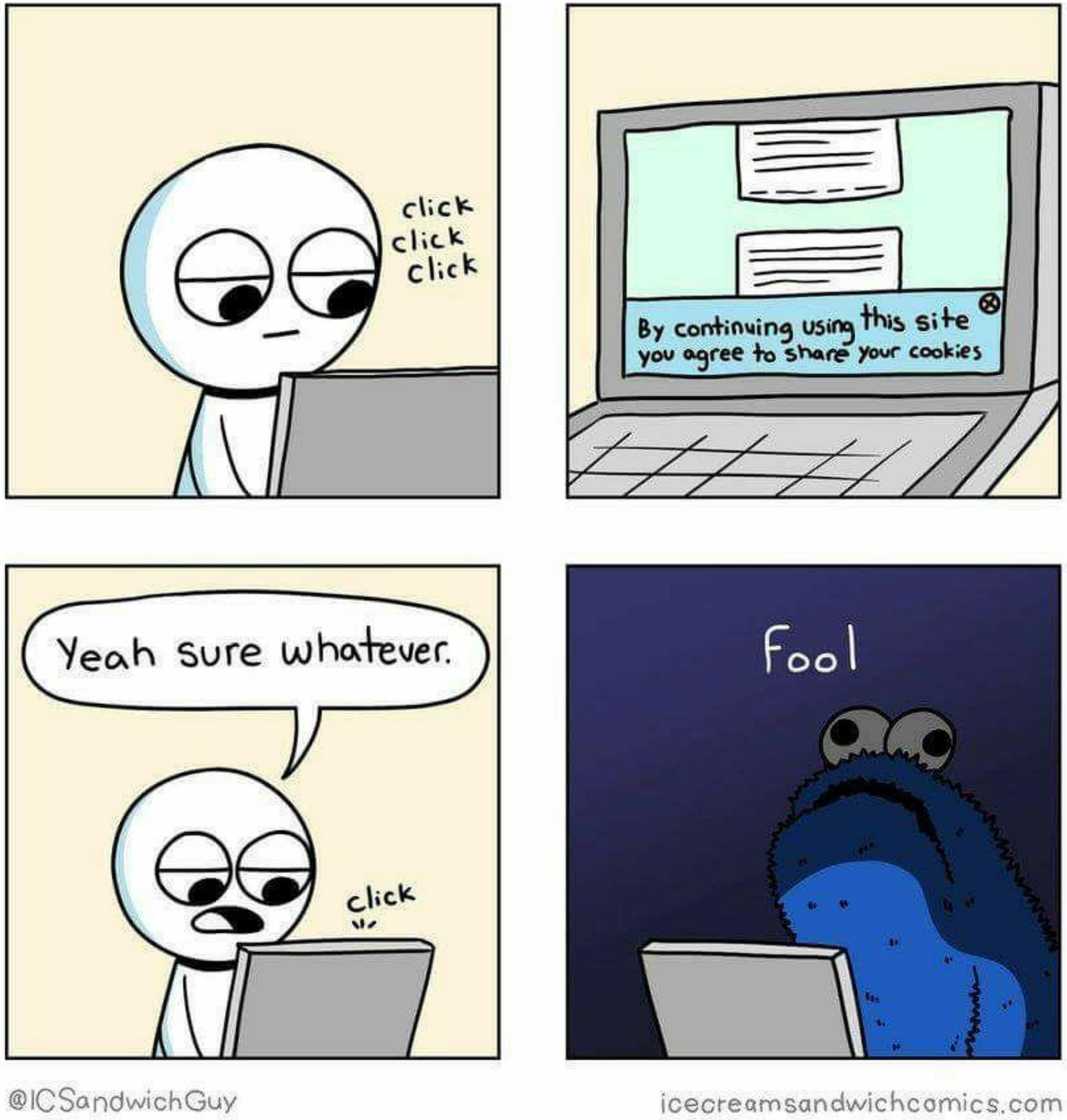
- The browser will store cookie values locally
- At each HTTP/S request, it will send the cookies in the HTTP headers
- Cookies are sent only to same server who asked to set them
    - Eg, cookies set from “foo.com” are not going to be sent when I do GET requests to “bar.org”
- JavaScript can read those cookie values on the browser
- What is the problem with it?
    - You can fabricate a web site with JS that reads all cookies, and then use them to access the user’s Google/Facebook/Bank accounts by doing AJAX calls…
- As cookies are arbitrary strings, they can be used to store data
    - usually up to 4K bytes per domain can be stored in a browser

Expires / Secure / HttpOnly

- **Set-Cookie: <name>=<value>; Expires=<date>; Secure; HttpOnly**
- *Expires*: for how long the cookie should be stored
- *Secure*: browser should send the cookie only over HTTPS, not HTTP
  - There are kinds of attacks to trick a page to make a HTTP toward the same server instead of HTTPS, and so could read authentication cookies in plain text on the network
- *HttpOnly*: do not allow JS in the browser to read such cookie.
  - This is critical for authentication cookies

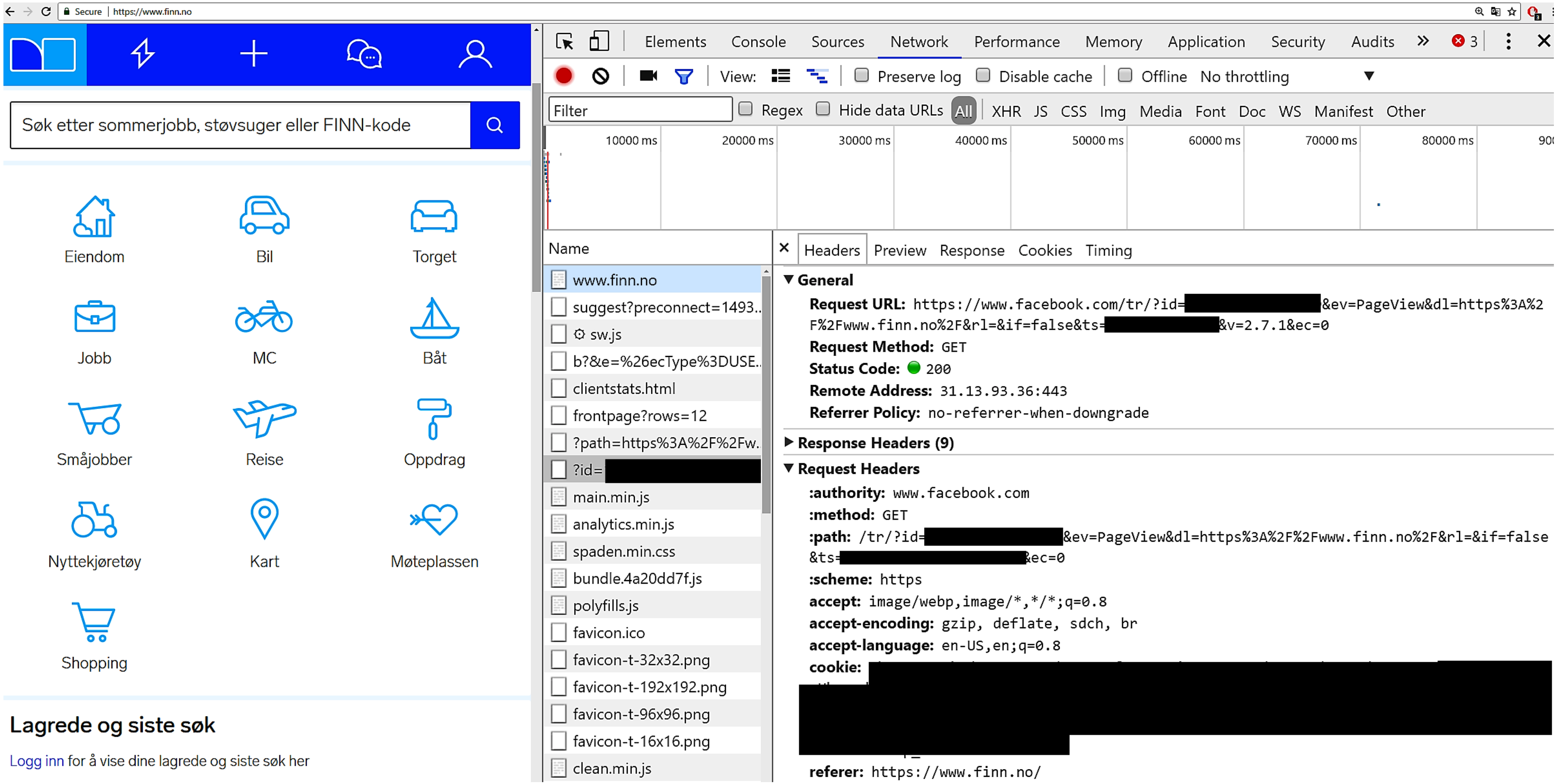
Cookie Tracking

- Besides session/login cookies that have an expiration date, server can setup further cookies (ie Set-Cookie header)
- There are special laws regarding handling of cookies
- Why? Tracking and privacy concerns…


click
click
click
By continuing using this site you agree to share your cookies
Yeah sure whatever.
click
Fool
@ICSandwichGuy
icecreamsandwichcomics.com

Tracking

- Many sites might rely on resources provided by other sites
  - Images, JavaScript files, CSS files, etc.
  - eg, Facebook “Like” button
- When you download a HTML page from domain X (eg *finn.no*) which uses a resource from Y (eg, *facebook.com*), the HTTP GET request for Y will include previous cookies from Y
- So, even if you are logged out from Facebook, FB can know which pages you visit (as long as they do use FB resources)
- Even worse, FB can track your browser even if you have never used FB!!!
- This happens by simply opening the page from X, no need to click anything!!!
- *referer* HTTP header: domain origin of request to Y from page not from Y
  - Eg, “Referer: X” is added when page loaded from X ask for resource in Y



Name

- www.finn.no
- suggest?preconnect=1493...
- ⚙ sw.js
- b?&e=%26ecType%3DUSE...
- clientstats.html
- frontpage?rows=12
- ?path=https%3A%2F%2Fw...
- ?id=[redacted]
- main.min.js
- analytics.min.js
- spaden.min.css
- bundle.4a20dd7f.js
- polyfills.js
- favicon.ico
- favicon-t-32x32.png
- favicon-t-192x192.png
- favicon-t-96x96.png
- favicon-t-16x16.png
- clean.min.js

▼ **General**

Request URL: `https://www.facebook.com/tr/?id=`[redacted]`&ev=PageView&dl=https%3A%2F%2Fwww.finn.no%2F&rl=&if=false&ts=`[redacted]`&v=2.7.1&ec=0`

Request Method: `GET`

Status Code: ● `200`

Remote Address: `31.13.93.36:443`

Referrer Policy: `no-referrer-when-downgrade`

▶ **Response Headers (9)**

▼ **Request Headers**

:authority: `www.facebook.com`

:method: `GET`

:path: `/tr/?id=`[redacted]`&ev=PageView&dl=https%3A%2F%2Fwww.finn.no%2F&rl=&if=false&ts=`[redacted]`ec=0`

:scheme: `https`

accept: `image/webp,image/*,*/*;q=0.8`

accept-encoding: `gzip, deflate, sdch, br`

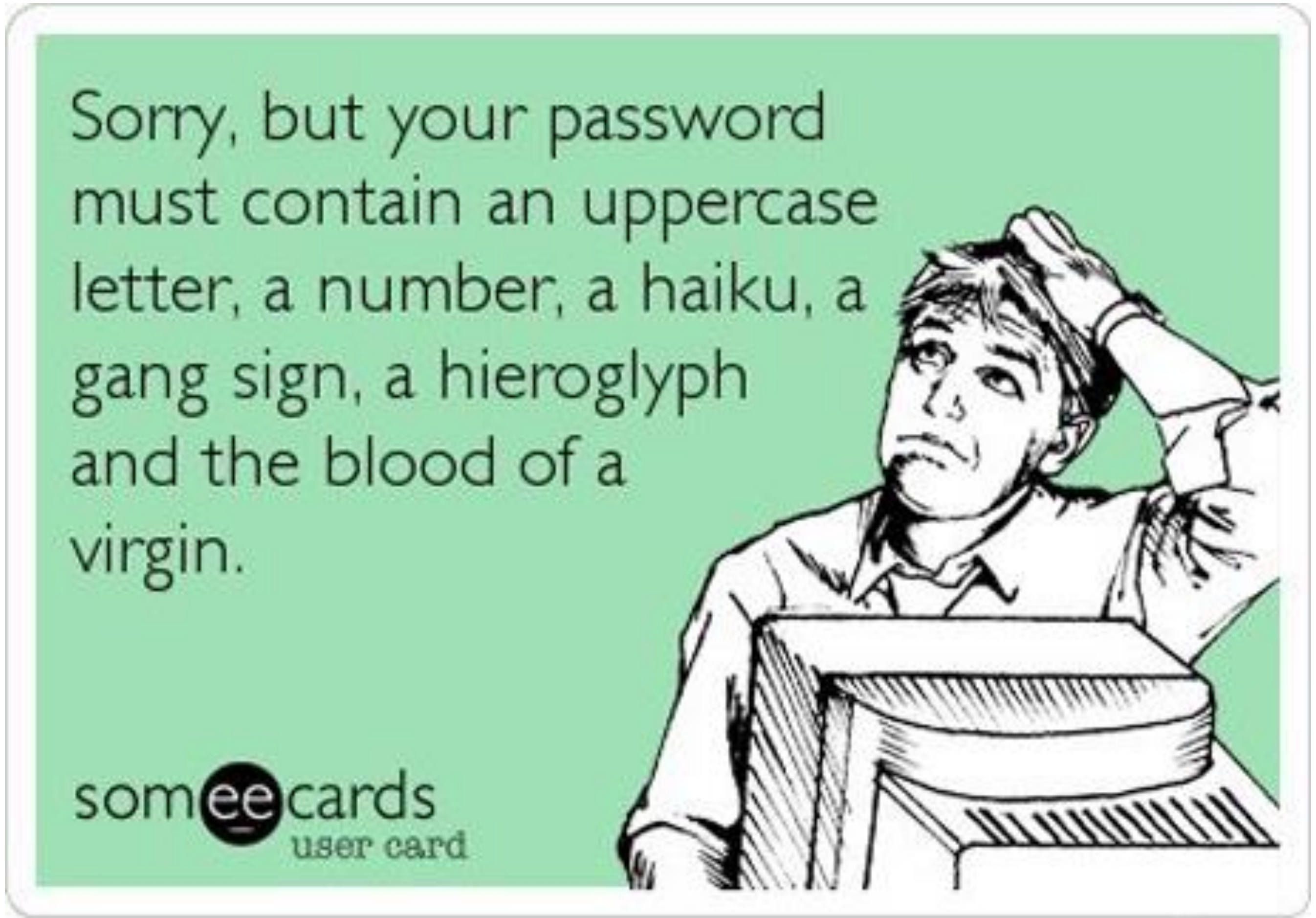
accept-language: `en-US,en;q=0.8`

cookie: [redacted]

referer: `https://www.finn.no/`

Passwords

- Needed to verify identity of a user
- Not too short or simple, otherwise too easy to crack with brute-force
- *Security* vs *Usability:* hard to get a good balance
    - eg, ideally would have different passwords for each different site, and change them often, eg every week… but who the heck is going to do that???

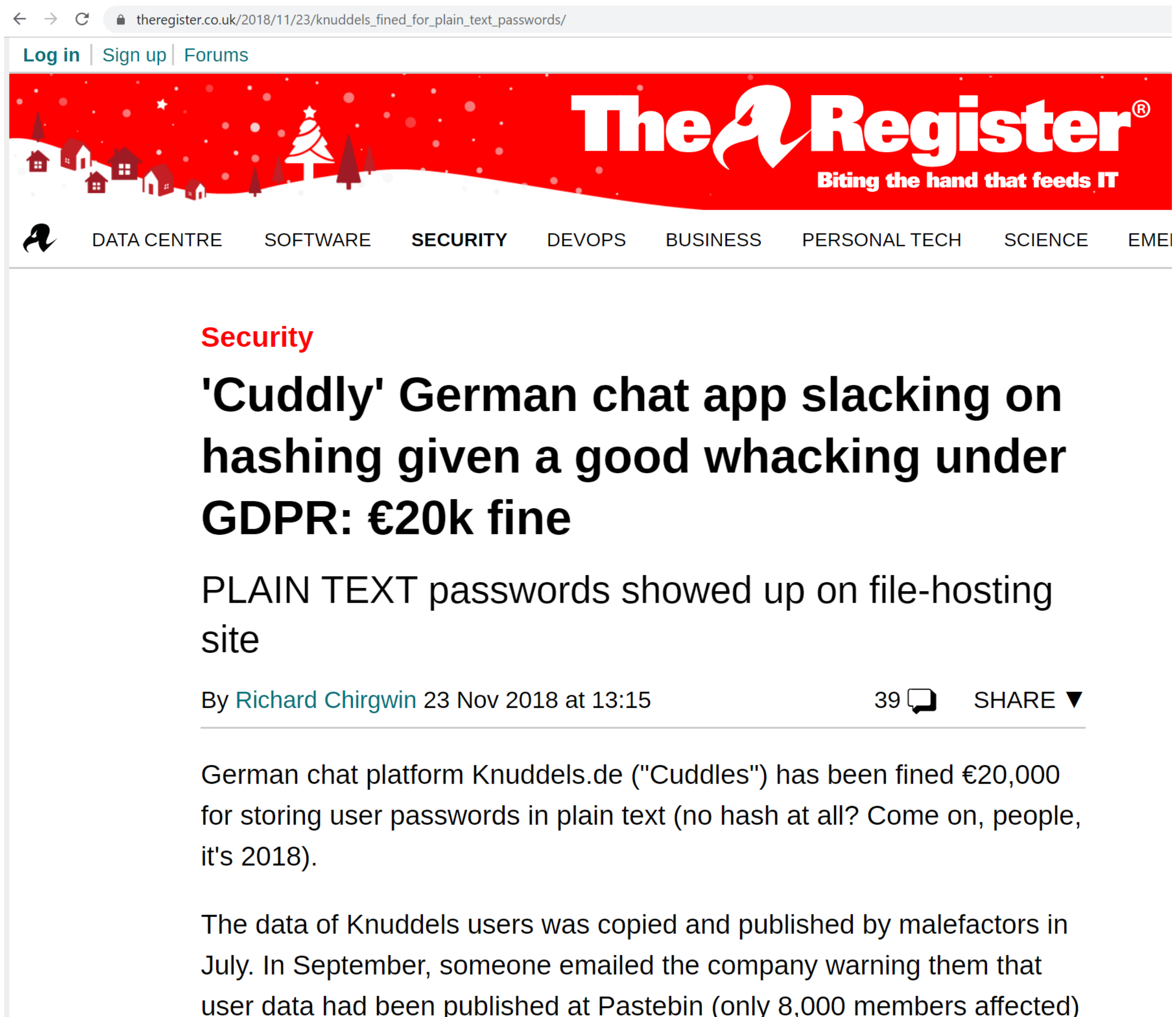


Sorry, but your password must contain an uppercase letter, a number, a haiku, a gang sign, a hieroglyph and the blood of a virgin.

Password Storage

- When creating new user, need to save password somewhere, usually a database
- **NEVER SAVE A PASSWORD IN PLAIN TEXT**
- Passwords need to be *hashed*
- Even if an hacker has full access to database, shouldn’t be able to get the password
  - Typical case is a successful SQL Injection attack
  - But many more cases: eg disgruntled employee, recovery from broken thrown away hard-drive, etc.
- Besides being able to impersonate a user, hacker can try the same password on other sites (Amazon/Facebook/etc)

Plain-text Passwords? GDPR fines…



theregister.co.uk/2018/11/23/knuddels_fined_for_plain_text_passwords/
Log in | Sign up | Forums
The Register®
Biting the hand that feeds IT
DATA CENTRE
SOFTWARE
SECURITY
DEVOPS
BUSINESS
PERSONAL TECH
SCIENCE
EMEI

Security

'Cuddly' German chat app slacking on hashing given a good whacking under GDPR: €20k fine

PLAIN TEXT passwords showed up on file-hosting site

By Richard Chirgwin 23 Nov 2018 at 13:15 39 SHARE ▼

German chat platform Knuddels.de ("Cuddles") has been fined €20,000 for storing user passwords in plain text (no hash at all? Come on, people, it's 2018).

The data of Knuddels users was copied and published by malefactors in July. In September, someone emailed the company warning them that user data had been published at Pastebin (only 8,000 members affected)

Hash Functions

- $h(x) = y$
- It is just a mathematical function from *x* to *y*
  - In our case, *x* is the password, and *y* is its hashed value
- Deterministic: always same *y* from same input *x*
- Shouldn't be able to recover *x* from *y,* even if you have full knowledge of how *h()* is implemented
- Small change *x'* to *x* should lead to big different between *y* and *y'*
  - ie, *y* and *y'* should look uncorrelated, and so cannot say if *x* and *x'* are similar
- No collision: no two values should have same hash, ie $h(x) = y = h(z)$

Login with Hashed Passwords

- How can server verify the login of user *A* with password *X*, if the server does not know the password *X*, but only the hash *Y=h(X)*?
- Server needs to retrieve from database the hash *Y* for given user *A*, recompute the hash *h(X)* from the input password *X*, and then verify that the new hash does match *Y*, ie *Y == h(X)*

Salted Passwords

- Cannot expect users to have long passwords
- If hacker has access to DB, from a hash *Y*, can calculate *h(K)* for all strings K up to certain length N, eg N=8, and check if any *h(K)* does match Y
- For small N, this is doable. Do not even need to run *h(),* as those values can be pre-computed, ie *Rainbow Tables*
- Further issue: two users with same passwords will have same hash Y
- Solution: add a random *salt* S (eg a random long string) to the password before hashing, and store the salt together with the hash in the database
- *h(X+S)=Y*
- Each user will have its own random salt

Pepper

- If hacker has access to the database, s/he can read the *salt* values
- Still non-trivial to break the hash code, but doable
- *Pepper*: yet another random string added before calculating the hash
- NOT stored in the database, just somewhere else
    - files, remote server, hardcoded in the source code, etc
- One single pepper string for whole application (and not per user)
- Help mitigating if hacker gets access to the database (eg via SQL Injection), as would not be able to read the pepper

Hash Function Speed

- You want hash functions that are *slow,* to make it difficult for the hackers to break them
    - but still manageable time on server to do authentication
- *BCrypt* is the most used hash function for passwords
- However, you can make *slow* any hash function (eg SHA256) by using a loop, in which the output is re-hashed N times
    - *eg, N=6 ->   h(h(h(h(h(h(x)))))) = y*

Spring Security

About these slides

- These few slides on Spring Security are just high level overviews of what covered in class
- The details are directly in the code comments on the Git repository

Security Is Hard

- You should NOT roll out your own solutions for security
  - far too easy to make mistakes
- Need to use existing, battle-tested frameworks
- *Spring Security*: the module of Spring that deals with security
- However, still important to understand how they work internally

Configuration

- Need to have a *@Configuration* bean that extends *WebSecurityConfigurerAdapter*
- Furthermore, need to use the annotation *@EnableWebSecurity*
- Then, in such class you can override the methods:
  - *configure(AuthenticationManagerBuilder auth)* for handling *authentication*, ie checking of users on database and password storage
  - *configure(HttpSecurity http)* for handling *authorization*, ie the access policy rules

OWASP

Open Web Application Security Project (OWASP)

- www.owasp.org
- A non-profit organization dedicated to software security
- One of the main resources to learn about software security
- Also produces some open-source tools (eg *ZAP* for penetration testing)
- Maven plugin *dependency-check-maven*
  - Scan your third-party dependency libraries for known vulnerability
  - Automatically connect to an updated database
  - Note: nowadays hosting services like *GitHub* do dependency checks automatically

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **intro/spring/security/manual**
- **intro/spring/security/authorization**
- **intro/spring/security/dependencies**
- Exercises for Lesson 10 (see documentation)