Multimedia im Netz

Online Multimedia

Winter semester 2015/16

Tutorial 10 – Major Subject

Today’s Agenda

- Theory recap: questions from the assignments
- Mash-Ups
- Discussion: Your Questions so far.

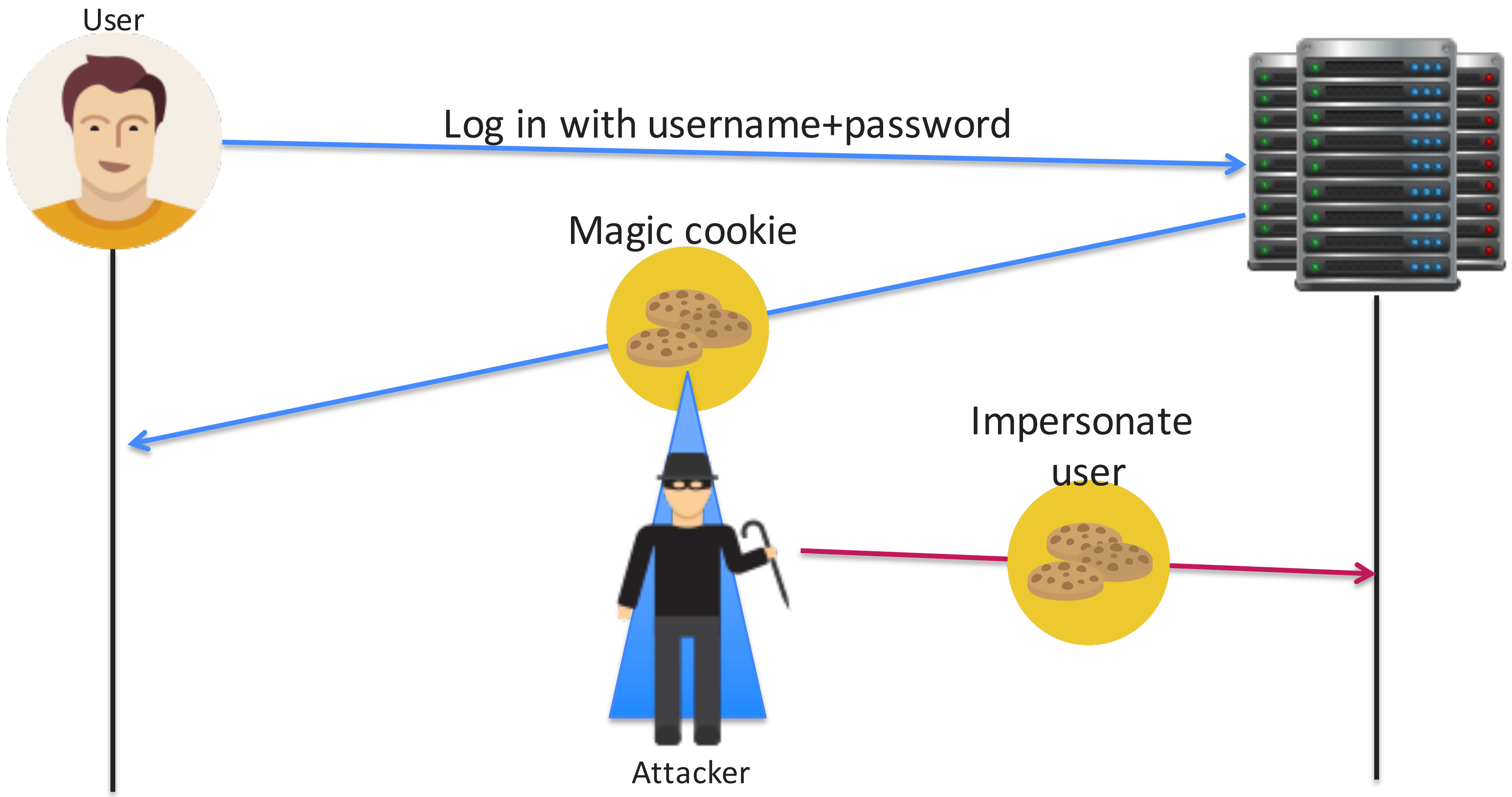
Theory Recap

Cookie Theft

- **Man-in-the-middle** (MITM) attack with the goal to **impersonate another user** through stealing a "Magic-Cookie"
- Also known as Session-Hijacking
- Solution: Encrypted communication channels (SSL)

- Reading material:
  - https://www.techopedia.com/definition/24633/cookie-theft
  - http://www.welivesecurity.com/2010/11/09/cookie-theft-sidejacking-or-session-hijacking-for-normal-people/
  - https://en.wikipedia.org/wiki/Session_hijacking
  - https://en.wikipedia.org/wiki/HTTP_cookie#Cookie_theft_and_session_hijacking

Cookie Theft Example



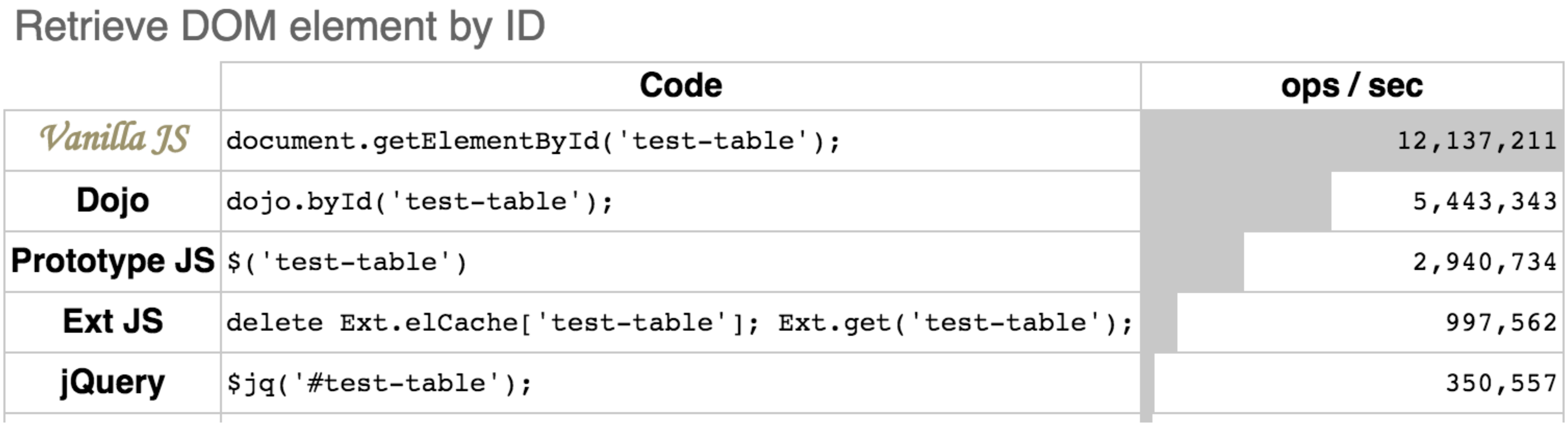
- Take-away: Use SSL/TLS!
- Get your SSL Certificate here: https://letsencrypt.org/

Vanilla JS

- … is not a real framework.
- Vanilla JS = Using JavaScript without any frameworks / libraries
- Pros:
  - Much faster in terms of operations per second
  - Only slightly “uglier”
- Cons:
  - Requires more code
  - Handy methods not always available (cross-browser issue)

Why is jQuery a potential problem?

- DOM selections offer convenience functionality
- Example: You can pass Strings containing selectors or HTML or jQuery objects or genuine DOM-nodes to the `$()` function.
  ```
  $('<div>Hi!</div>').appendTo('div:eq(2)');
  ```
- This results in if-then controls affecting performance



Retrieve DOM element by ID

	Code	ops / sec
Vanilla JS	`document.getElementById('test-table');`	12,137,211
Dojo	`dojo.byId('test-table');`	5,443,343
Prototype JS	`$('test-table')`	2,940,734
Ext JS	`delete Ext.elCache['test-table']; Ext.get('test-table');`	997,562
jQuery	`$jq('#test-table');`	350,557

source (data not verified): vanilla-js.com

Hoisting (1)

- Variable declarations are moved to the top of the current scope → a variable can be used **before** it was declared
- Example

```
<div id="foo"></div>
<script>
    function setContent(){
        content = 'This is a hoisting test.';

        var div = document.getElementById('foo');
        div.innerHTML = content;

        var content;
    }
</script>
```

http://www.w3schools.com/js/js_hoisting.asp

Hoisting (2) - Implications

- Since we can use variables before they were declared, this might lead to bugs very easily.
- Recommendation: Declare **<u>all</u>** your variables at the top of a scope.
- Example:

```
function properSetContent(message){
    var content, div;
    content = message;
    div = document.getElementById('foo');
    div.innerHTML = content;
}
```

http://www.w3schools.com/js/js_hoisting.asp

Style Guides, Tips, and Tricks

- These guides are highly recommended, if you are into extending your knowledge about Front-End coding
- https://github.com/airbnb/javascript
  airbnb's very exhaustive and structured approach to improve the quality of their JavaScript code.
- https://github.com/AllThingsSmitty/css-protips
  https://github.com/AllThingsSmitty/jquery-tips-everyone-should-know
  CSS and jQuery tips by AllThingsSmitty
- https://github.com/bendc/frontend-guidelines
  Front-end markup/code recommendations by D. De Cock

Screen Scraping

- Most commonly “Web Scraping”: Automatic information extraction from web sites
- Screen scraping sometimes also means: taking an automated screenshot and running the image through OCR (optical character recognition)
- Example
  - Flight search engines
  - Data aggregators
  - Mash-ups
- Often, screen scraping violates usage terms!

Static, Dynamic, Duck-Typing

- Static: Every variable is declared with a static, non-changeable type. E.g. `String s = "myString";`
- Dynamic: Variables are declared without an explicit type. E.g.
```
var x = 42;
var s= "Hello";
```
- Duck-Typing:
  - Special form of "dynamic" typing: all that counts is the suitability to perform an action with an object.
    → "does the object have method XYZ?"
  - "When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck." James Whitcomb Riley.

https://en.wikipedia.org/wiki/Duck_typing

Lazy Loading

- Design Pattern (not only on the web)
- Common use-case on the web: placeholders that are replaced with the actual images
- Advantages
  - Web-site content becomes visible/accessible faster
  - Traffic can be reduced
- Disadvantages
  - Number of requests can rapidly increase
  - Difficult to cache / bookmark
- Where do you see lazy loading every day?

https://en.wikipedia.org/wiki/Lazy_loading

Watermarking: Characterization Task

EXIF information in JPEG file

- Visibility: **not in image directly**. EXIF tool necessary.
- Universality: **Depends**. Images from the same camera will all have the same EXIF information regarding the camera model / vendor.
- Detectability: **High**. File explorer shows data with a mouse click.
- Robustness: **Low**. Printing and scanning destroys watermark.
- Capacity: **Medium**. There are many different fields. But difficult to store “rich” information (e.g. logos) in EXIF info
- Security: **Low**. EXIF information can easily be changed.
- Efficiency: **High**. There is little overhead with inserting the data.

Mash-Ups

Mash-Ups

- Aggregation of multimedia content: Single web page that shows content from a lot of other sources.
- One specific topic (e.g. a music band)
- Content originates from external web services
- Usually, mashups gather data from multiple sources and display it nicely

- Get inspired: http://www.programmableweb.com/mashups/

https://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

Mash-Ups: Visually

flickr + last.fm + YouTube

+ + + Play on Spotify

+ ... = Mash-Up

Prerequisite: Authentication

- Opening an API to the public can cause a lot of traffic/stress for the servers (... and their administrators)

- Many services require you to sign up for an **access key** to the application programming interfaces (APIs)
  - Usually sent via a GET/POST parameter to identify the origin
  - Used to monitor requests and quota.
  - Fixed quota of requests for some services (which you probably won‘t exceed in this course)

- Advanced Authorization: OAuth

OAuth

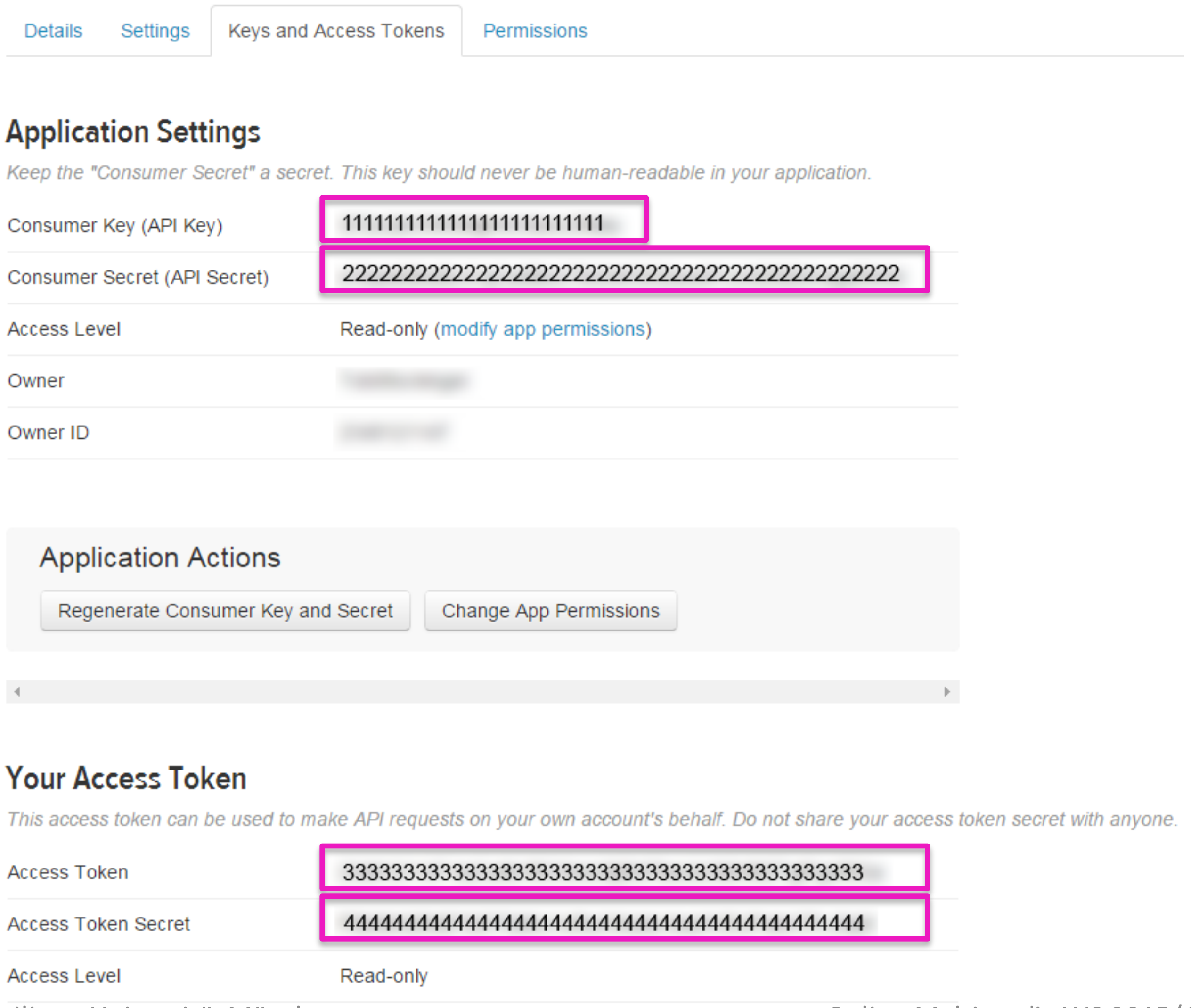
- Motivation: Users want to ensure that web apps can only access what has been approved by the users themselves.
- Solution: OAuth
  Standardized protocol for API authorization
- Providers issue access tokens to apps allowing them to operate in their name
- Many APIs support the OAuth mechanism
- Further readings:
  - http://hueniverse.com/oauth/
  - http://oauth.net/

Example: Twitter & OAuth

- The Twitter API not accessible from client-side JavaScript, because the API secrets would become readable.

- There are two variants in twitter:
  - Application-User authentication:
    - App acts on behalf of user
    - Authentication ensures permissions for each app
  - **Application-only authentication:**
    - **App does not have any user-context (e.g. profile name)**
    - **Only allows access to publicly available information on twitter**

Register a Twitter App



Details | Settings | Keys and Access Tokens | Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	1111111111111111111111111
Consumer Secret (API Secret)	22222222222222222222222222222222222222222222222222222
Access Level	Read-only (modify app permissions)
Owner	[redacted]
Owner ID	[redacted]

Application Actions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	33333333333333333333333333333333333333333333333333
Access Token Secret	44444444444444444444444444444444444444444444444444
Access Level	Read-only

NodeJS and Twitter

- We can use a node app to access the Twitter API
- Libraries make our lives easier!
  - "Passport" provides general access to OAuth providers for user-authentication.
  - For our example, we use the `twitter` package that includes all steps for application-level authentication.
- More info: https://www.npmjs.com/package/twitter
- Use it in your app:

```
npm install twitter --save
```

routes/twitter.js (1)

- Define the Twitter access credentials in config/config.js
- Example usage:

```
var express = require('express');
var router = express.Router();
var config = require('../config/config');
var Twitter = require('twitter');



var twitterClient = new Twitter(config.twitter);
```

routes/twitter.js (2)

```
router.get('/', function (req, res) {
    var searchTerm;
    if (req.query.q && req.query.q.length > 0) {
        searchTerm = req.query.q;    }
    else {        searchTerm = 'MMN'    }
    twitterClient.get('search/tweets', {
        q: searchTerm
    }, function (error, docs) {
        if(!error){
            res.json({
                status : 'success',
                tweets : docs.statuses,
                message : 'fetched Tweets'
            });
        }
        else{
            res.json({
                status : 'error',
                message : error
            })
        }
    })
});
```

https://dev.twitter.com/rest/reference/get/search/tweets

routes/index.js

```
var express = require('express');
var router = express.Router();

var twitterRoute = require('./twitter');

router.use('/twitter',twitterRoute);

module.exports = router;
```

On the front end

- We provided a fully working front end in the examples on GitHub.
- This is where the call to the API is made:

```
function APIHandler() {
    const api = {
        baseURL: '/',
        tweets: {
            "get": 'twitter/'
        }
    };
    this.fetchTweets = function(searchTerm, callback) {
        $.get(api.baseURL + api.tweets.get, {
            q: searchTerm
        }, callback)
    };
}
```

Round-Up

Enjoy the holidays!

Thanks!
What are your questions?