Information Retrieval

WS 2012 / 2013

Lecture 7, Wednesday December 5th, 2012

(PHP, Cross-Site Scripting, Cookies, UTF-8)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

UNI FREIBURG

- Organizational
  - Your experiences with ES#6 (search web application)
- More web app stuff + UTF-8
  - Dynamic contents with **PHP**: very short intro
  - At one point in the last lecture it said in the JS console:
    Origin http://... not allowed by Access-Control-Allow Origin
    What does that mean and how can it be fixed?
  - **Cookies:** ingredients and recipes
  - **UTF-8:** background and specification
  - Exercise Sheet 7: Extend your web app from ES#6 by some tasty cookies + allow for corrupt UTF-8

Experiences with ES#6 (search web app)

UNI FREIBURG

- Summary / excerpts last checked December 5, 15:36
  - Nice / awesome / fascinating exercise + see it all work together
  - But many of you also had a hard time with the web stuff
  - Why was www code from lecture not provided? ... It was !
  - Annoying bugs ate a hole lot of time ... sleep first !
  - "I feel that no-one is reading what I write here" ... I do !
  - Annoying due to wrong formula in the script ... I told you
  - Please send a mail to all if something major is fixed ... OK !
  - C programmers sacrifice usability for useless performance gains
  - Tutor overly strict with points ... I told them please not to be
  - Page numbers on the left please ... impossible, sorry

PHP (**P**HP: **H**ypertext **P**reprocessor)

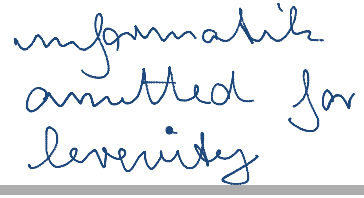
UNI FREIBURG

- A full-featured (interpreted) programming language
  - Especially suited for outputting HTML pages with variable elements (e.g. depending on URL parameters)

```
<!DOCTYPE html>
<html><body><p>
<?php for ($i = 1; i < $GET_["n"]; i++) print "$i, "; ?>
</p></body></html>
```

  - Syntax-wise it's a mixture of Perl and C and C++
  - Quite a "dirty" language: no variable types, weak object-orientation and unit testing (like in Perl), inconsistencies, ...
  - Not recommended for large / complex projects !
  - Still most popular language for tasks like the above though

Same-origin policy 1/2



UNI FREIBURG

- When communicating with a server:
  - Domain of client and server URL must be the same, e.g.

    Web page: http://**stromboli.uni-freiburg.de**/demo.html

    Server script: http://**stromboli.uni-freiburg.de**/demo.php?...
  - Javascript can be loaded from arbitrary locations though, e.g.

    <script src="http://code.jquery.com/jquery-1.8.3.js"></script>
  - This can be used to circumvent the same-origin policy:

    ```
    script = document.createElement("script");
    script.src = "http://etna.uni-freiburg.de/demo.php?...";
    document.body.appendChild(script)
    ```

    This works! And will execute the JS output by demo.php?...

Same-origin policy 2/2

UNI FREIBURG

- Problem: script loading and exec. is asynchronous
  - In our example: we don't know when the demo.php?... has loaded and the produced JS has finished execution
  - **Idea:** let demo.php?... produce JavaScript that calls a function, with the result as argument

    callback([2, 3, 5, 7, 9, 11, 13, 17, 19])
  - Now all we need is a function callback in our original JavaScript, and process the result there
  - This is exactly the mechanism behind **JSONP**, and the kind of code that gets executed in jQuery when writing

    $.ajax({url: "http://...", dataType: "jsonp"})

Cross-site scripting (XSS)

UNI FREIBURG

- Most frequent security vulnerability of web apps
  - **Principle:** inject JavaScript into web page

    Let's look at a simple example in our example code
  - **Example 1:** send someone a mail with a link

    ...index.php?user=guest<script>alert("Got you!")</script>

    or, more sophisticated, with parameters ASCII encoded

    ...index.php?%75%73%65%72%3d%67%75%65%73...
  - **Example 2:** post to forum with some script in it

    I have a question on Exercise Sheet 7.
    <script>... JS code to send me user info by mail ...</script>

    Note: The <script>...</script> will not show on the website, but code will be executed by **any client** viewing the post

Cookies 1/3

UNI FREIBURG

- Specification
  - 1 cup of butter
  - 1 cup of white sugar
  - 1 cup of brown sugar
  - 2 eggs
  - 2 TSPs vanilla extract
  - 3 cups of white flour
  - 1 TSP baking soda
  - 2 cups of chocolate
  - 1 cup of walnuts
- Implementation advice
  - Preheat oven to 450°K
  - Cream butter + sugar
  - Beat in eggs one at a time
  - Add vanilla + baking soda
  - Stir in flour, chocolate, nuts
  - Drop by large spoonfuls onto ungreased pans
  - Bake for ≈ 600K msecs
  - Eat in O(1) time

Cookies 2/3

UNI FREIBURG

- Alternative use in web pages
  - A string stored along with the web page, **but on the client's computer** ... can be different for different clients !
  - String contains an (almost) arbitrary sequence of key-value pairs, separated by semi-colons, for example

    ```
    username=cookie_monster; preference=kekse
    ```

  - Read and set in JavaScript via **document.cookie**

    ```
    var cookies = document.cookie.split(";");
    for (var i = 0; i < cookies.length; i++) {
      var args = cookies.replace(/\s/g,"").split("=");
      if (args[0] == "username") alert("Welcome " + args[1]);
    }
    ```

Cookies 3/3

UNI FREIBURG

- Types of cookies
  - The first type is called **chocolate chip cookie**

    Accidentally developed by Ruth Wakefield in 1930
  - The second type is called **session cookie**

    This lasts as long your browser is open
  - If you specify an expiry date you get a **persistent cookie**

    username=cookie_monster;
    expires=Wed, 05 Dec 2012, 17:45:00 GMT
  - Cookies from other domains are called **third-party cookies**

    Check Resources → Cookies in your JavaScript Console

UTF-8 1/4

UNI FREIBURG

- What is UTF and why do we need it?
  - UTF = Unicode Transformation Format
  - A standard for encoding all the characters of the world
  - Extends the long-standing ASCII / ISO-8859-1

    (which can only differentiate between 256 characters)
- How to encode so many different characters?
  - 1 byte is obviously not enough
  - 2 bytes are also not enough (≤ 65,536 different characters)
  - So take 4 bytes per character → this is what UTF-32 does
  - But the size of strings now quadruples compared to ASCII !
  - And so does the time to process these strings ...

UTF-8 2/4

UNI FREIBURG

- UTF-8 is a variable-byte encoding that realizes all of the following
  - ASCII compatible = a string of characters with ASCII codes < 128 is the same in ASCII as in UTF-8
  - Frequent special characters (like ä, á, å) need two bytes, only very rare characters (old scripts) need four bytes
    - the € symbol needs three bytes though: 226 130 172
  - Easy to decode / convert to UTF-32
  - In particular: no need to decode from left to right, can decode starting from anywhere within a string

UTF-8 3/4

- Here is the encoding Unicode → UTF-8
  - Case 1: Unicode in [0, 127] = xxxxxxx (7 bits)

    → UTF-8 code is 0xxxxxxx (1 byte)
  - Case 2: Unicode in [128, 2047] = yyyxxxxxxxx (11 bits)

    → UTF-8 code is 110yyyxx 10xxxxxx (2 bytes)
  - Case 3: Unicode in [2048, 65535] = yyyyyyyyxxxxxxxx (16 bits)

    → UTF-8 code is 1110yyyy 10yyyyxx 10xxxxxx (3 bytes)
  - Case 4: Unicode in [65536, 2^{21} - 1] = zzzzzyyyyyyyyxxxxxxxx (21)

    → UTF-8 code is 11110zzz 10zzyyyy 10yyyyxx 10xxxxxx
  - Could continue with 5-byte and 6-byte sequences, but UTF-8 stops here, due to RFC 3629

UTF-8 4/4

ü: 11000011 10111100
Code point 00011111100 = 252
=FC

- Some observations
  - In a multi-byte UTF-8 character all bytes are ≥ 128, and vice versa such bytes occur only for multi-byte characters
  - The number of leading 1s in the first byte of a multi-byte character encodes the length of the sequence
  - The concatenation of the remaining bits (except for the 0 that follows the leading 1s) are called the code point
  - For every Unicode in [0, 2^{21} - 1] there is **exactly one** UTF-8 multi-byte sequence
  - But vice versa not all multi-byte sequences are valid UTF-8
  - For example **110**0000x **10**xxxxxx is **not** valid

use 0xxxxxxx instead (one byte is enough)

References

- PHP
  - http://en.wikipedia.org/wiki/PHP
  - http://php.net/manual/en/index.php
- Cross-Site Scripting (XSS)
  - http://en.wikipedia.org/wiki/Cross-site_scripting
- Cookies
  - http://en.wikipedia.org/wiki/HTTP_cookie
  - http://www.w3schools.com/js/js_cookies.asp
- UTF-8
  - http://en.wikipedia.org/wiki/UTF-8
  - http://www.utf8-chartable.de