Web Applications

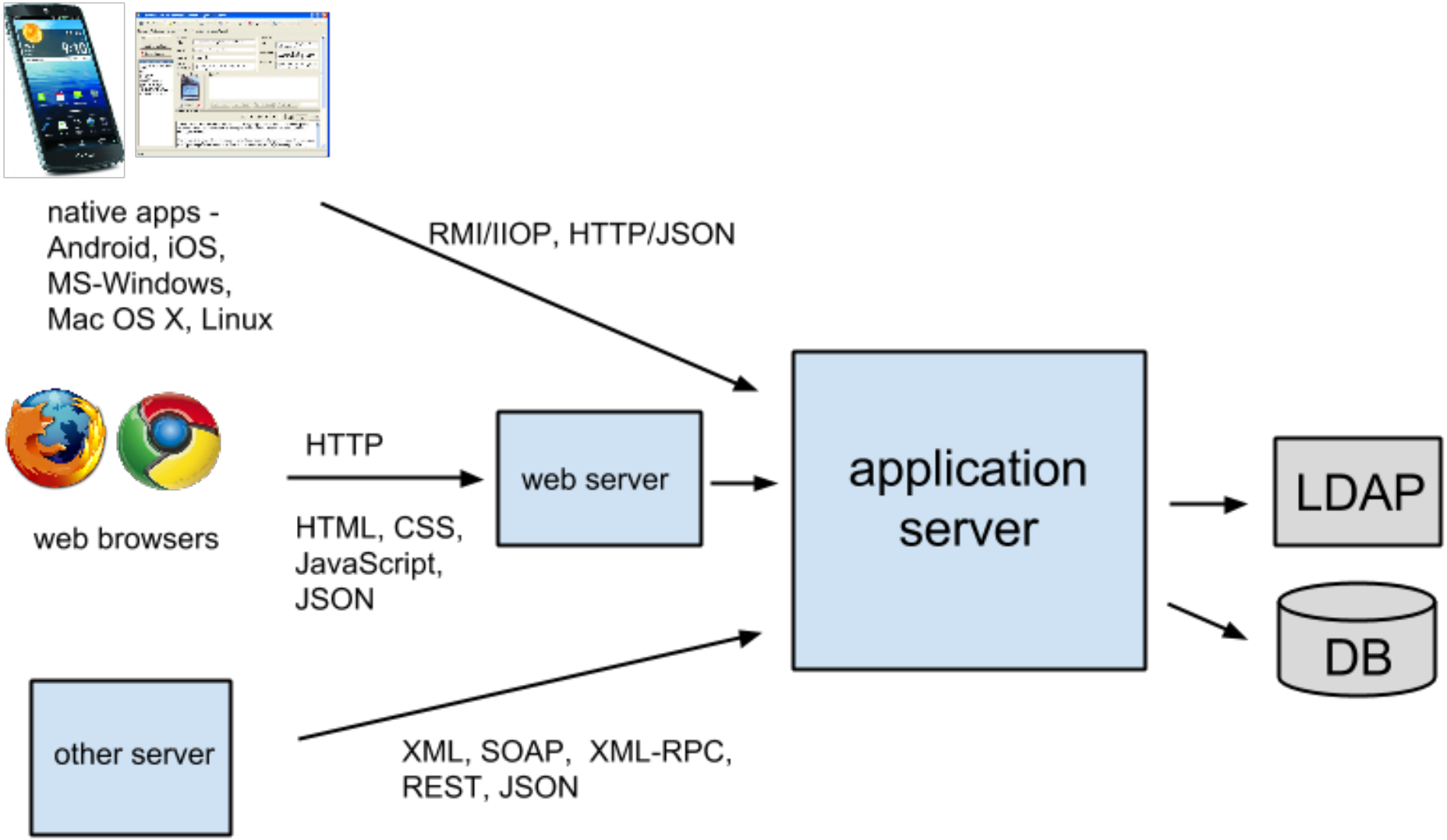
PA 165, Lecture 7

Martin Kuba

Outline

- Architecture of web applications
- Servlet API
- JSP, JSP EL, JSP tags, tag libs, JSTL
- Security (authentication, authorization, main attacks)

Layers in multi-tier application



SaaS Cloud

- web applications are **Software-as-a-Service** type of cloud service
- provide on-demand access to software
- device independence – PC, notebook, tablet, smartphone, smart TV, …
- web mail, messaging, office suites, media libraries, communication tools, business sw …
- Gmail, Facebook, Google Drive, Dropbox, Spotify, Flickr, YouTube, WebEx, NetSuite ...

Deployment

- SaaS services can be deployed
  - into Platform-as-a-Service (PaaS) cloud
    - Google App Engine, Amazon Elastic Beanstalk, Microsoft Azure Websites, RedHat OpenShift, Heroku, ...
  - into Infrastructure-as-a-Service (IaaS) cloud
    - Google Computing Engine, Amazon Elastic Compute Cloud, Microsoft Azure, ...
  - locally
- software is provided as
  - downloadable executable code (i.e. JavaScript, Android app)
  - callable API on provider's servers (e.g. Google Calendar API)

Client side technologies

- HTML, links, forms, CSS
- cookies
- JavaScript, Document Object Model, AJAX
- HTML 5 features - <canvas>, <video>, web storage, web sockets, file API, geolocation API, device orientation, media capture
- WebGL (Web Graphics Library)
- SVG (Scalable Vectr Graphics)
- dead technologies – Java applets, Flash

http://caniuse.com/

WebGL - 3D Canvas graphics - OTHER

Global 54.15% + 27.56% = 81.71%

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			[1] 31					4.1	
8		38	43					4.3	
9		39	44					4.4	
10		40	45	8		8.4		4.4.4	
[1] 11	[1] 12	41	46	9	32	9	8	44	46
	[1] 13	42	47		33				
		43	48		34				
		44	49						

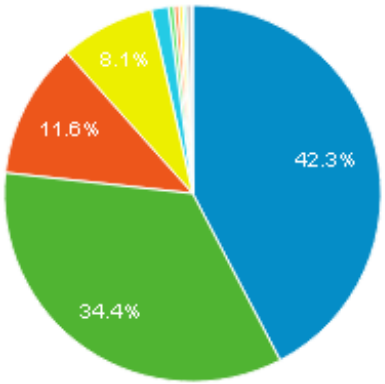
Notes | **Known issues (1)** | **Resources (9)** | **Feedback**

1. Older versions of IE11 have only partial support of the spec, though it's much better with the latest update.

Google Analytics

	Browser	Sessions	Sessions
		13,077 % of Total: 100.00% (13,077)	13,077 % of Total: 100.00% (13,077)
1.	Chrome	5,533	42.31%
2.	Firefox	4,504	34.44%
3.	Safari	1,521	11.63%
4.	Internet Explorer	1,056	8.08%
5.	Opera	193	1.48%
6.	Opera Mini	60	0.46%
7.	Edge	53	0.41%
8.	Android Browser	49	0.37%
9.	UC Browser	33	0.25%
10.	Amazon Silk	12	0.09%
11.	Safari (in-app)	12	0.09%
12.	Mozilla Compatible Agent	9	0.07%
13.	SeaMonkey	8	0.06%
14.	YaBrowser	8	0.06%

Contribution to total: Sessions



Client side libraries and frameworks

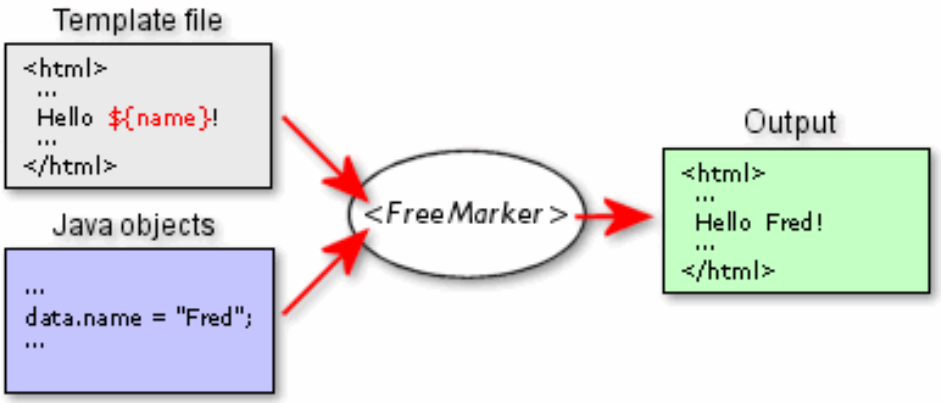
- area of rapid changes
- **jQuery** – JavaScript library
  - HTML document traversal and manipulation, event handling, animation, and Ajax
  - unified API on multiple browsers
- **AngularJS** – framework for declarative programming of user interfaces
  - extends HTML vocabulary for applications
  - suited for single-page applications

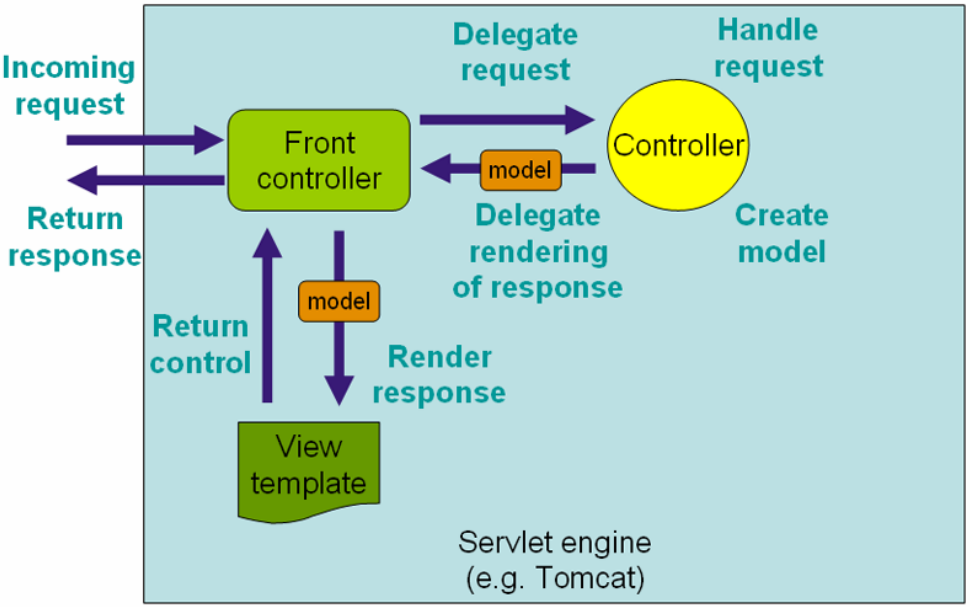
Server side technologies

- PHP, Python, Perl, Ruby on Rails, NodeJS, …
- Java web containers (Apache Tomcat, Jetty, JBoss, IBM WebSphere, …)
- basic Servlet API for handling HTTP
- Java Server Pages (JSP) for page templates
- frameworks on top of Servlet API

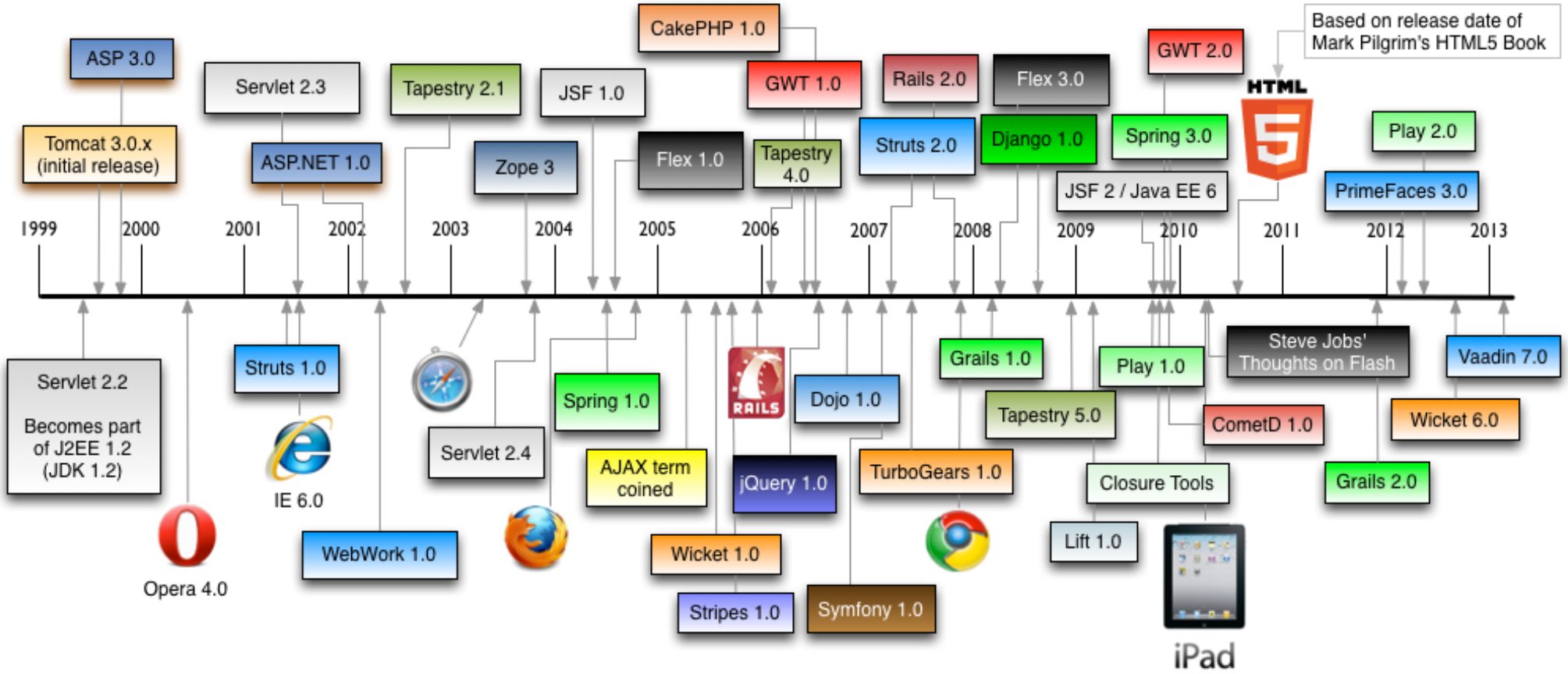
Java server side frameworks

- page templates
  - JSP, tag libraries, JSTL
  - Velocity
  - Freemarker
- Model-View-Controller
  - Spring MVC
  - Stripes
  - Apache Struts
  - ...





Web frameworks history



Servlet API

- **Servlets** for managing HTTP requests
- **Filters** for modifying requests and responses
- **Listeners** for handling events (e.g. app start)
- **HttpSession** for maintaining state
- **RequestDispatcher** and **attributes** for cooperation among multiple servlets

Servlet

```java
@WebServlet("/someurl/*")
public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest r, HttpServletResponse s)
            throws ServletException, IOException {
        r.setAttribute("data", prepareData());
        r.getRequestDispatcher("/some.jsp").forward(r, s);
    }

    @Override
    protected void doPost(HttpServletRequest r, HttpServletResponse s)
            throws ServletException, IOException {
        handleRequestData(r);
        s.sendRedirect(s.encodeRedirectURL(r.getContextPath() + "/someurl"));
    }
```

Filter

```java
@WebFilter("/urls")
public class MyFilter implements Filter {

    @Override
    public void doFilter(ServletRequest r, ServletResponse s, FilterChain ch)
            throws IOException, ServletException {
        doSomething(r,s);
        ch.doFilter(r,s);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException { }

    @Override
    public void destroy() { }
```

Listeners

- ServletContextListener, ServletRequestListener,HttpSessionListener, ...

```
@WebListener
public class MyContextListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent e) {
        ServletContext ctx = e.getServletContext();
        ctx.setAttribute("common_object",getCommonObject());
    }

    @Override
    public void contextDestroyed(ServletContextEvent e) {...}
```

HttpSession

- keeps a Map<String,Object> on server
- assigned to a particular browser
- maintained using cookie or URL rewriting
- timeout after 30 minutes since last request

```
HttpSession session = request.getSession(true);
Boolean authenticated = (Boolean) session.getAttribute("authenticated");
if(!authenticated) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
}
```

Attributes (Scoped variables)

- setAttribute(String name,Object value) on
  - ServletContext (shared by all browsers)
  - HttpSession (shared by all requests from a browser)
  - HttpServletRequest (shared by filters and servlets during a request)
  - JspContext (shared in a single JSP page)
- easily accessed in JSPs as *${name}*

WAR – Web ARchive

- a servlet container can have multiple web applications running, mapped by different context path (first part of URL)
- each web app is deployed in a *.war file
- a ZIP archive containing
  - WEB-INF/classes/**/*.class … classes
  - WEB-INF/lib/*.jar … libraries
  - WEB-INF/web.xml … deployment descriptor
  - WEB-INF/tags/*.tag … custom JSP tags
  - directly accessible files like *.jsp, *.png, *.css, *.js

Java Server Pages

- an HTML file with special directives, converted to a servlet on each change
- scriptlets, EL language, tags, tag libraries

```
<%-- some comment --%>
<%@ page contentType="text/html;charset=UTF-8" session="false" %>
<%@ taglib prefix="my" tagdir="/WEB-INF/tags"  %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
    <%= pageContext.findAttribute("a") %>
    <% out.print(pageContext.findAttribute("a")); %>
    <c:out value="${a}" escapeXml="false"/>
```

JSP Expression Language (EL)

- expressions inside of **${ }**
- value expressions
  - ${attribute.property}
  - ${attribute['property']}
- operators: + - * / div % mod and or not == != < > <= >= empty
- functions defined by tag libraries
  - ${fn:length(orderitems)}

Own JSP tags

- defined in *.tag files with syntax similar to *.jsp
- can be used for common page layout
- attributes can be: simple, dynamic, fragment

```
<%@ tag pageEncoding="utf-8" trimDirectiveWhitespaces="true" %>
<%@ attribute name="title" required="true" %>
<%@ attribute name="body" fragment="true" required="true" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html lang="${pageContext.request.locale}">
<head>
    <title><c:out value="${title}"/></title>
</head>
<body>
<h1><c:out value="${title}"/></h1>
<div id="content">
    <jsp:invoke fragment="body"/>
</div>
```

Java Standard Tag Library (JSTL)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<fmt:setBundle basename="MyTexts" scope="page"/>
<fmt:message key="sometext"><fmt:param value="42"/></fmt:message>
<fmt:formatDate value="${now}" type="both" dateStyle="full" timeStyle="full"/>

<c:set var="a" value="123" scope="request"/>
<c:out value="${a}"/>
<c:if test="${a>1}"> a is bigger than 1 </c:if>
<c:forEach items="${cars}" var="car" varStatus="i"> ${i.count}: ${car.id} </c:forEach>
<c:choose>
    <c:when test="${a<0}"> a is negative </c:when>
    <c:when test="${a==0}"> a is zero </c:when>
    <c:otherwise>a is positive</c:otherwise>
</c:choose>

${fn:substringBefore('foo;bar',';')}
```

Security

- authentication – validation of identity
- authorization - access rights specification

"On the Internet, nobody knows you're a dog."

Authentication in web apps

- form based
- HTTP BASIC - name and password
- SSL client certificate
- federated identity (SAML, OpenID)
- OAuth
- other (HTTP DIGEST, Kerberos, etc.)

Form based authentication

- own form, flag stored in HttpSession, Filter checking the flag
- Servlet API offers “HTTP FORM”, but very limited customization
- how to **not** store passwords:
  - plaintext (can be stolen)
  - hashed (can be reversed using rainbow tables)
  - hashed with salt (can compute specific rainbow table)
- recommended way how to store passwords:
  - BCrypt or PBKDF2 (PBKDF2WithHmacSHA512 in JRE)

HTTP BASIC

- server sends response

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="My secret area"
```

- client asks user for name and password
- browser sends name and password with every request

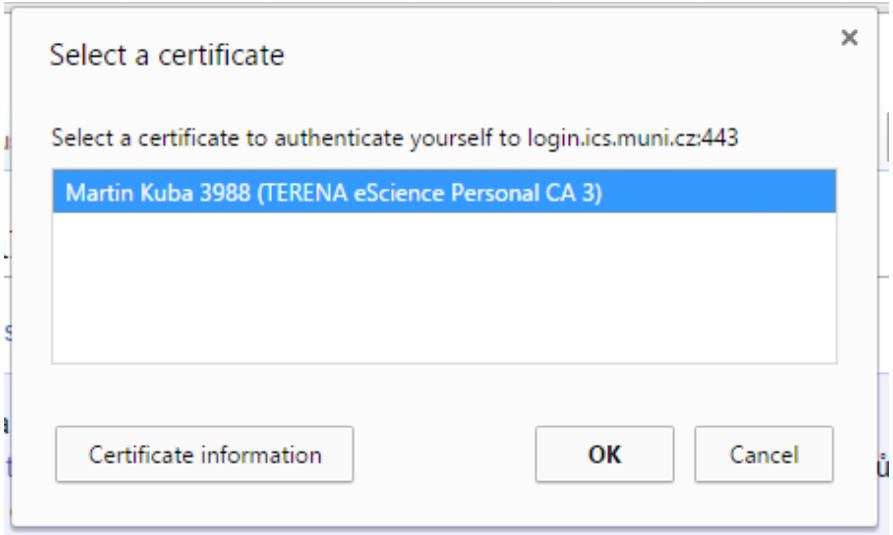
```
GET /protected/ HTTP/1.1
Authorization: Basic bWFrdWI6bWFrdWJpaw==
```

- username:password encoded by base64
- communication must be encrypted using SSL/TLS, otherwise can be easily stolen !

Client X509 certificate

- SSL server **must** present a certificate

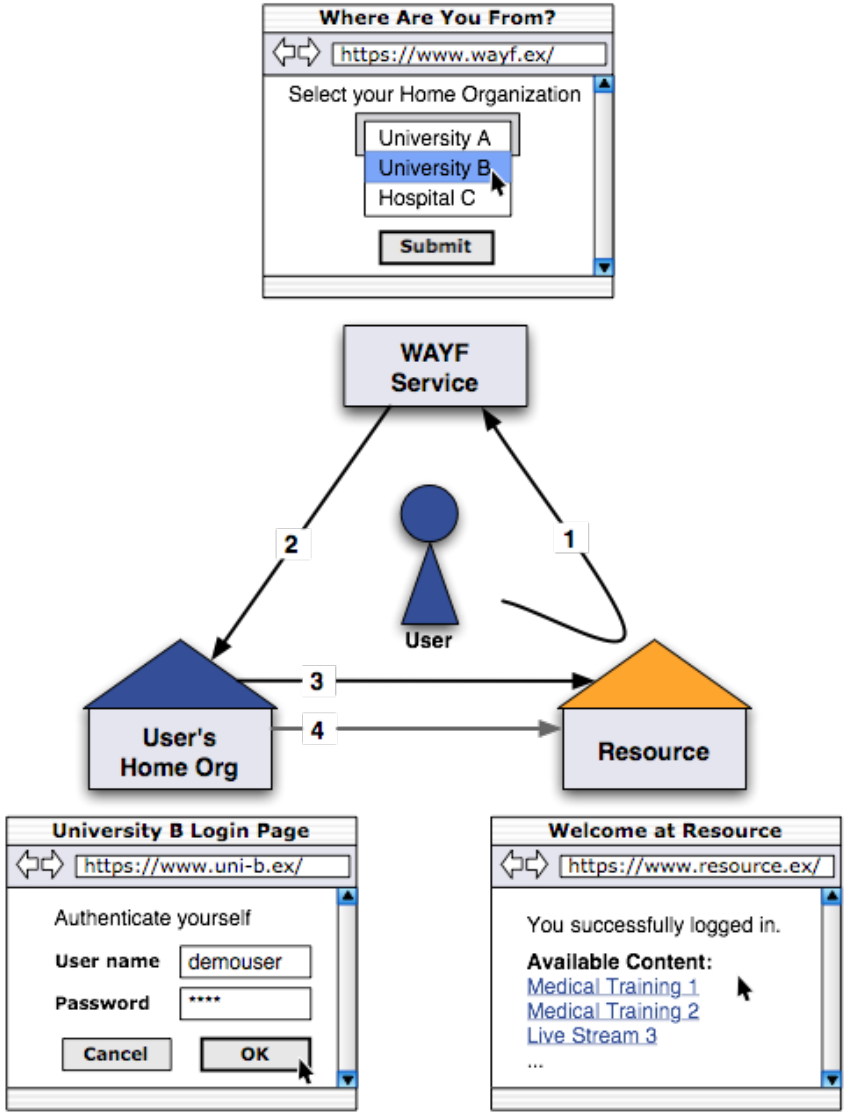
- SSL client **may** present a certificate



- get your certificate at https://tcs.cesnet.cz/

Federated identity

- three parties
  - user
  - Identity Provider
  - Service Provider
- SAML (Security Assertion Markup Language) standard
- Czech **eduId.cz** federation
- worldwide **eduGAIN**



OAuth

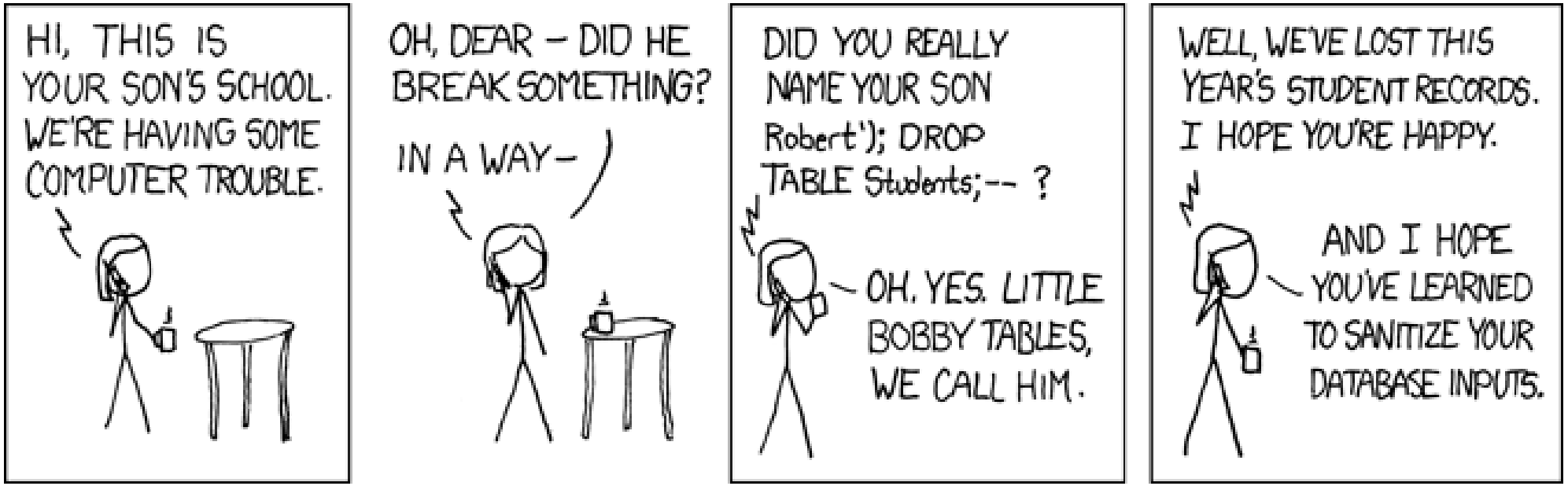
- “open authentication” standard
- used by Google, Facebook, LinkedIn, …
- tree parties
  - user as resource owner
  - resource server (e.g. Facebook)
  - client application (e.g. some third party game)
- user **authorizes** the client to perform a specific set of operations on resource server on his/her behalf
- can be used for **authentication** as authorization for reading user's personal info
- user can revoke the authorization anytime

Attacks

- SQL injection
- session hijacking
- session fixation
- XSS
- XSRF
- clickjacking
- phishing

SQL injection attack

- attacker sends special strings as inputs
- for values, use PreparedStatement in JDBC
- for other (e.g. ORDER BY) check using regex



Session hijacking

- attacker gets the cookie identifying someone else's session and uses it
- defence – session cookies must have attribute “secure”, so can be sent only over SSL/TLS

Session fixation

- attacker sends a URL to the victim containing a session identifier in a parameter
- defence – do not accept session identifiers from URL parameters, use cookies, change session identifier after each login

XSS – Cross site scripting

- attacker injects client-side script (JavaScript) into web pages viewed by other users
- defence
  - replace special characters with HTML entities using <c:out> tag
  - check URLs coming from user for “javascript:”
  - sanitize untrusted HTML input

XSRF – Cross site request forgery

- passive attack, attacker prepares a URL that causes an action on another server
- defence
  - important actions (e.g. money transfers) need to be confirmed
  - web forms contain a hidded randomly generated token

Clickjacking (UI redressing)

- attacker shows a web page in IFRAME in another web page
- tricks user into performing undesired actions by clicking on a concealed link
- defence – X-Frame-Options HTTP header

Phishing

- attacker masquerades as a trustworthy entity
- typically sends an email with link to fake web pages asking user to log in
- defence
  - train users to recognize phishing attempts
  - use Extended Validation server certificates

Penetration testing

- launching a software attack on a computer system looking for security weaknesses
- testing frameworks
  - Metasploit
  - w3af
- automatically performs attacks (SQL injection, XSS, …) on a web application

Thank you for your attention