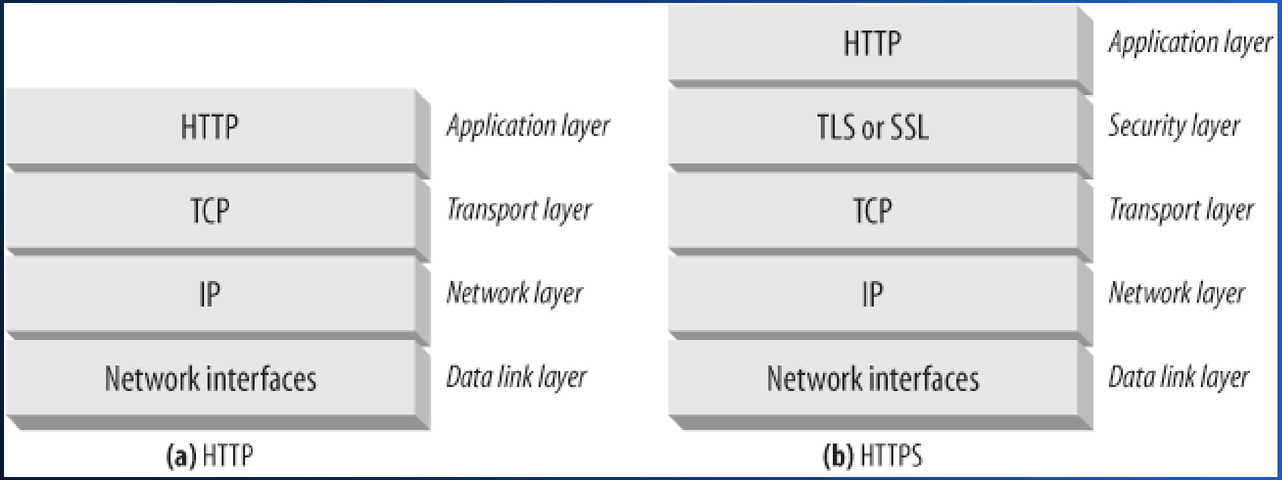
Secure Communication

HTTPS –
On the example of Tomcat webapps

HTTPS

- Hypertext Transfer Protocol Secure
  - Encryption/decryption of data packages
  - Additional Security Layer
    - Transport Layer Security (TLS); old: SSL



https://www.oreilly.com/library/view/http-the-definitive/1565925092/httpatomoreillycomsourceoreillyimages96902.png

HTTPS

- Protects against eavesdropping
- Protects against Man-in-the-middle attacks?
    - Only with certificate authentication!
        - And trustworthy certificate authorities (CA)...
- Encryption through:
    - Asymmetric keys (public & private key pair)
    - Symmetric keys

Asymmetric key encryption

- e.g. RSA (Rivest-Shamir-Adleman)
- Calculate e,d,n such that:
  - For a message m : $(m^e)^d \equiv m \pmod{n}$
  - Then also $(m^d)^e \equiv m \pmod{n}$

    => n & e = public key

    => n & d = private key
- Is secure because prime factorization of large integers (n) takes a lot of time

Webservice Certificate

- To prove that the webservice is who the client believes it is
- Contains:
  - Domain / server name, Location
  - Organizational information, Validity time
  - Public key
  - Digital Signature

Webservice Certificate

- Self-signed Certificate is encrypted with your own private key
    - Others can use your public key to verify that you encrypted the Certificate
    - But noone trusts that this Certificate is actually from the owner of the website

Certificate Authority (CA)

- Trusted agencies that can verify & sign your certificate to build a chain of trust
    - GWDG (https://info.gwdg.de/docs/doku.php?id=de:services:it_security:pki:start)
    - Telecom, Bundesnetzargentur, Globalsign, Let's Encrypt, ...
- The most common ones are pre-stored in your web-browser
    - Root CAs
- What happens if such an agency is hacked?

TLS/SSL Implementation in Tomcat

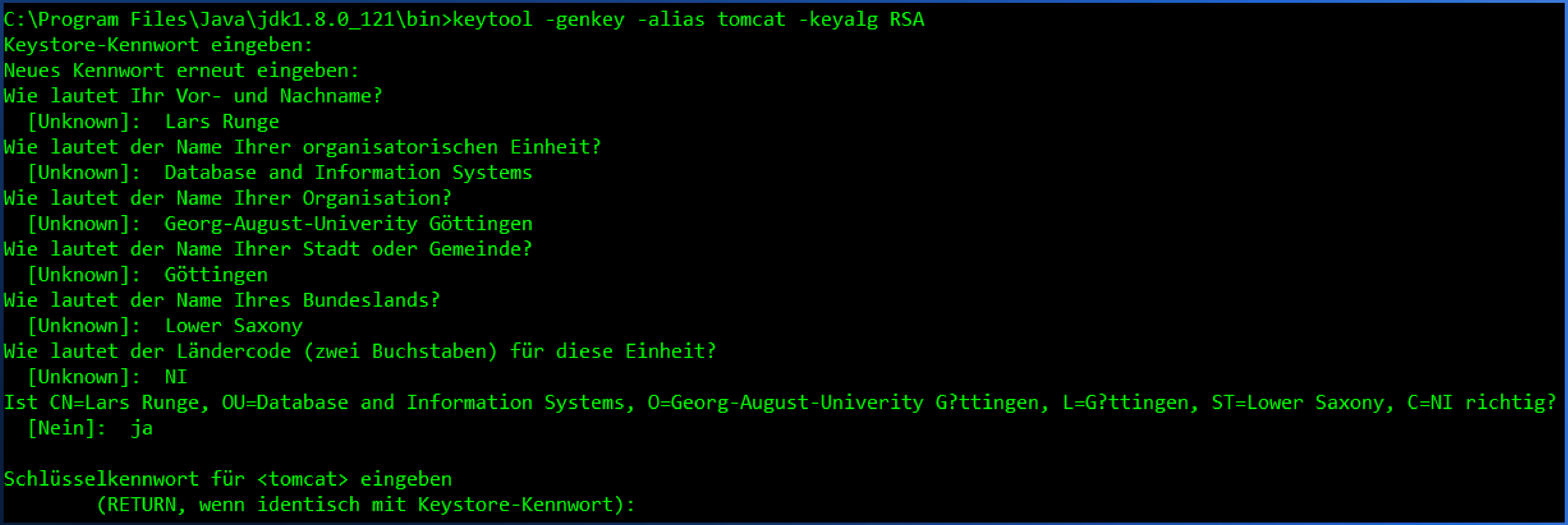
- Only necessary if used as a stand-alone web server
  - Not if used as a Servlet container; e.g. when using in combination with Apache Web Server
- Supported Certificate Keystores
  - JKS (Java Keystore), PKCS11, PKCS12

TLS/SSL Implementation in Tomcat

- Creation of a new JKS keystore
  - cd %JAVA_HOME%\bin
  - keytool -genkey -alias tomcat -keyalg RSA
    - Optional: -keystore \path\to\my\keystore
  - Answer a few questions
    - Tomcat default password: „changeit“
  - Self-signed
    - Not trustworthy, but good enough for a test

Implementation in Tomcat

- Creation of a new JKS keystore



```
C:\Program Files\Java\jdk1.8.0_121\bin>keytool -genkey -alias tomcat -keyalg RSA
Keystore-Kennwort eingeben:
Neues Kennwort erneut eingeben:
Wie lautet Ihr Vor- und Nachname?
  [Unknown]:  Lars Runge
Wie lautet der Name Ihrer organisatorischen Einheit?
  [Unknown]:  Database and Information Systems
Wie lautet der Name Ihrer Organisation?
  [Unknown]:  Georg-August-Univerity Göttingen
Wie lautet der Name Ihrer Stadt oder Gemeinde?
  [Unknown]:  Göttingen
Wie lautet der Name Ihres Bundeslands?
  [Unknown]:  Lower Saxony
Wie lautet der Ländercode (zwei Buchstaben) für diese Einheit?
  [Unknown]:  NI
Ist CN=Lars Runge, OU=Database and Information Systems, O=Georg-August-Univerity G?ttingen, L=G?ttingen, ST=Lower Saxony, C=NI richtig?
  [Nein]:  ja

Schlüsselkennwort für <tomcat> eingeben
        (RETURN, wenn identisch mit Keystore-Kennwort):
```

Java keytool

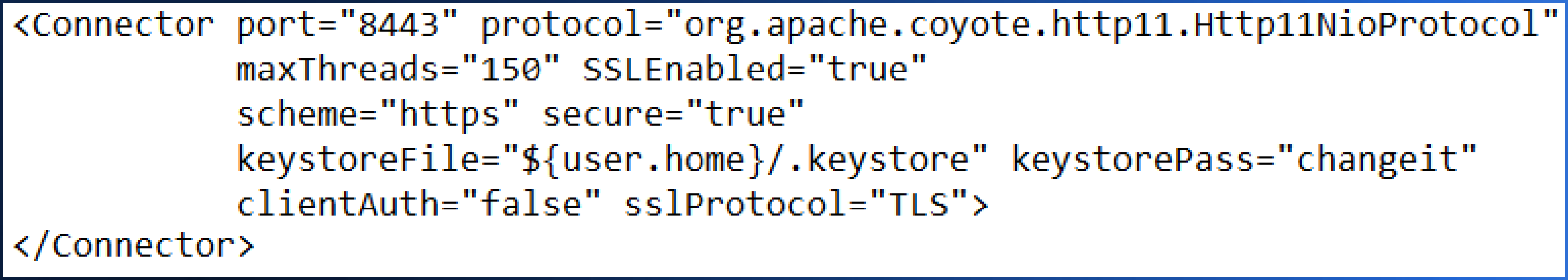
- -genkeypair / -genkey
    - Generates one private & public key pair
    - -alias : The name of the private key
    - -keyalg : Key generation algorithm used
        - RSA, DES, DSA
    - -keystore : Location & name of the keystore file
    - -keysize : Number of bytes used
        - 1024, 2048, ...

Java keytool

- -list
  - Prints the content of the keystore
  - -alias : Only the specified key
  - -v / -rfc : Human-readable output
  - -keystore : Location & name of the keystore
- -certreq
  - Generates a Certificate Signing Request (CSR)
  - -alias / -keystore / -sigalg / etc. as before

TLS/SSL Implementation in Tomcat

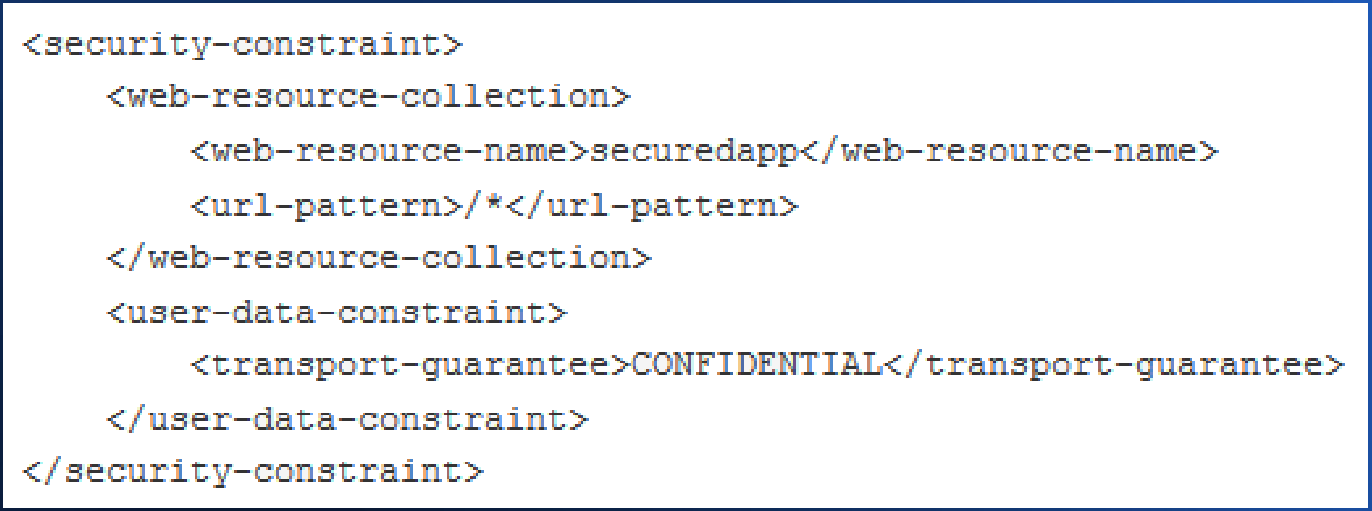
- Find the server.xml in ..\Tomcat\conf
- Find the example connector with
  - <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol" …
- Change it to:



```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true"
           scheme="https" secure="true"
           keystoreFile="${user.home}/.keystore" keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS">
</Connector>
```

TLS/SSL Implementation in Tomcat

- Force your servlet to work with TLS/SSL
- Edit the web.xml and add:



```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>securedapp</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

References

- https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html
- https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html
- https://docs.oracle.com/cd/E19830-01/819-4712/ablrb/index.html

Webservice Multi-user Support

Session management & Authentication -
On the example of Tomcat webapps

Sessions

- Originally the HTTP protocol was meant to be state-less
  - Demand of webservices to have a state specific to each client

- Solution? *
  - Cookies, hidden form fields, URL rewriting

    => Session management

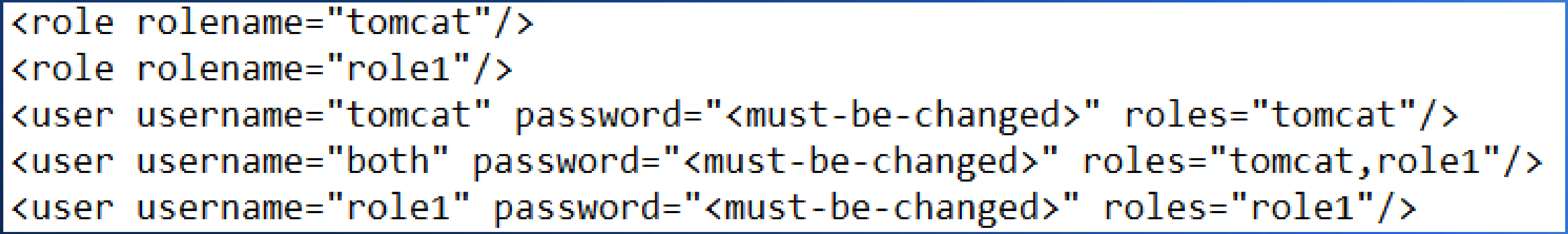
* http://static2.fr.de/storage/image/4/2/3/7/657324_608x342_1oFefH_eH4orl.jpg

Basic Authentication in Tomcat

- Automatic popup for username/password if requesting a webapp
- Users are stored in the tomcat-users.xml
    - Users can be given roles
    - Webapps can be restricted to specific roles in the web.xml
- Does not work with sessions, but the Authorization request header

Basic Authentication in Tomcat

- Tomcat-users.xml

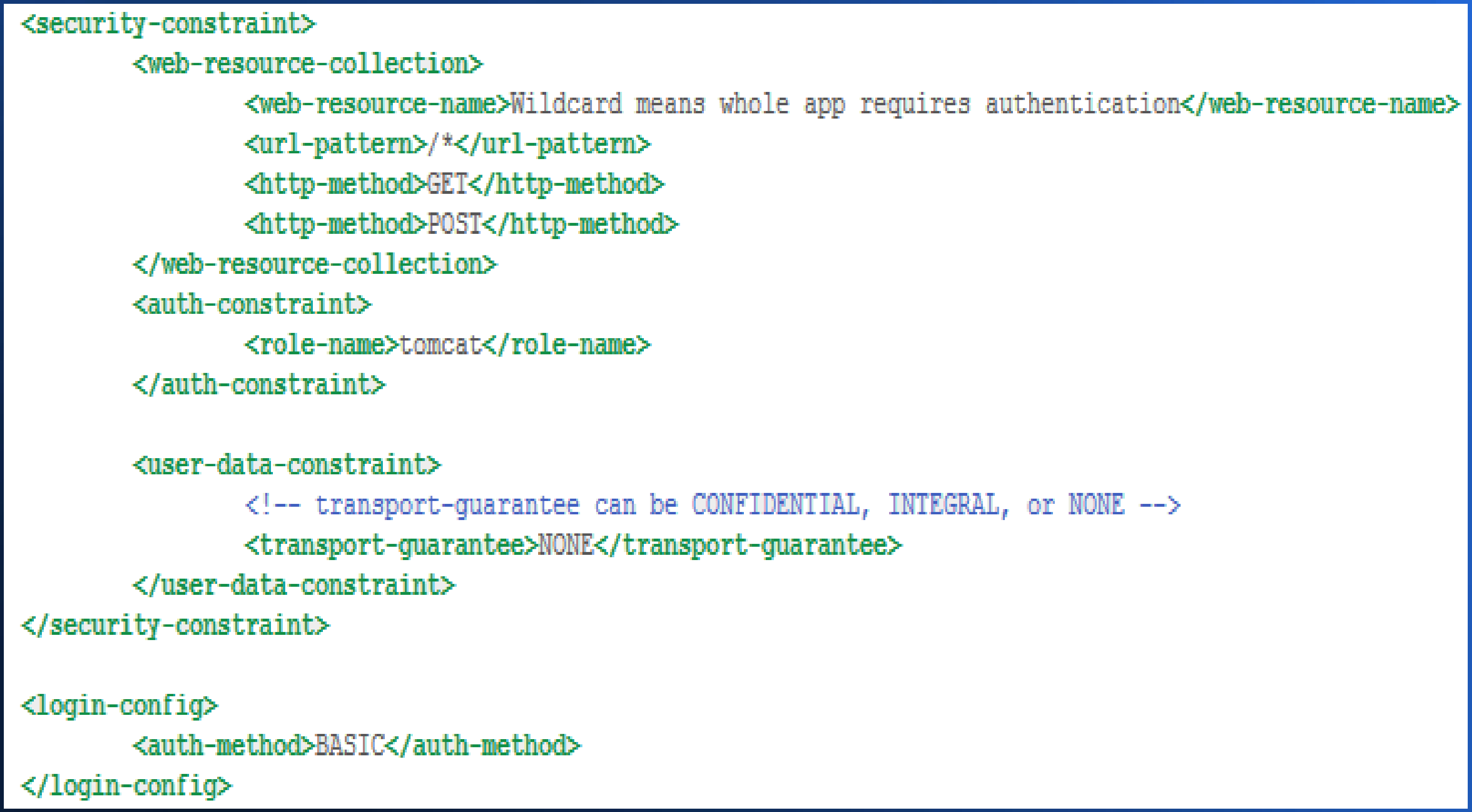


```
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
<user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
<user username="role1" password="<must-be-changed>" roles="role1"/>
```

- Through the server.xml Tomcat can be configured to automatically hash passwords with simple hash functions (e.g. MD5)
  - We already have shown that this is not secure

Basic Authentication in Tomcat

- Web.xml

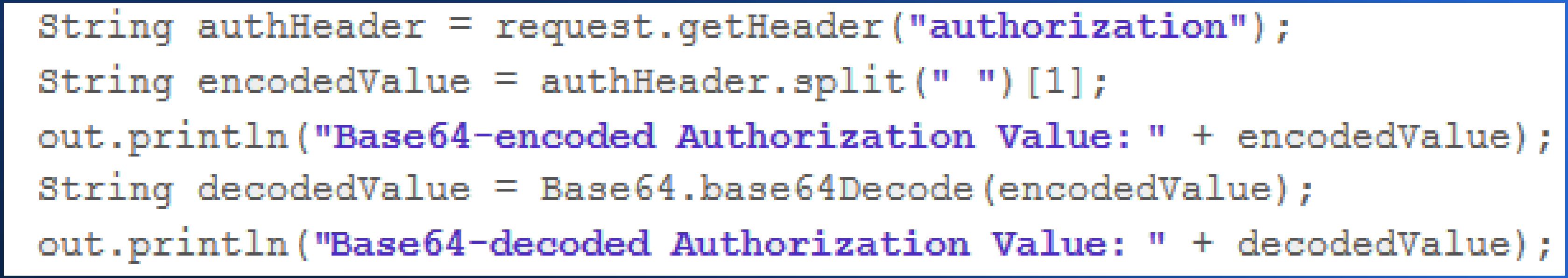


```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Wildcard means whole app requires authentication</web-resource-name>
                <url-pattern>/*</url-pattern>
                <http-method>GET</http-method>
                <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>tomcat</role-name>
        </auth-constraint>

        <user-data-constraint>
                <!-- transport-guarantee can be CONFIDENTIAL, INTEGRAL, or NONE -->
                <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
</security-constraint>

<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

http://www.avajava.com/tutorials/lessons/how-do-i-use-basic-authentication-with-tomcat.html?page=1

Basic Authentication in Tomcat

- The authentication information is also included in the request header (Base64-encoded)



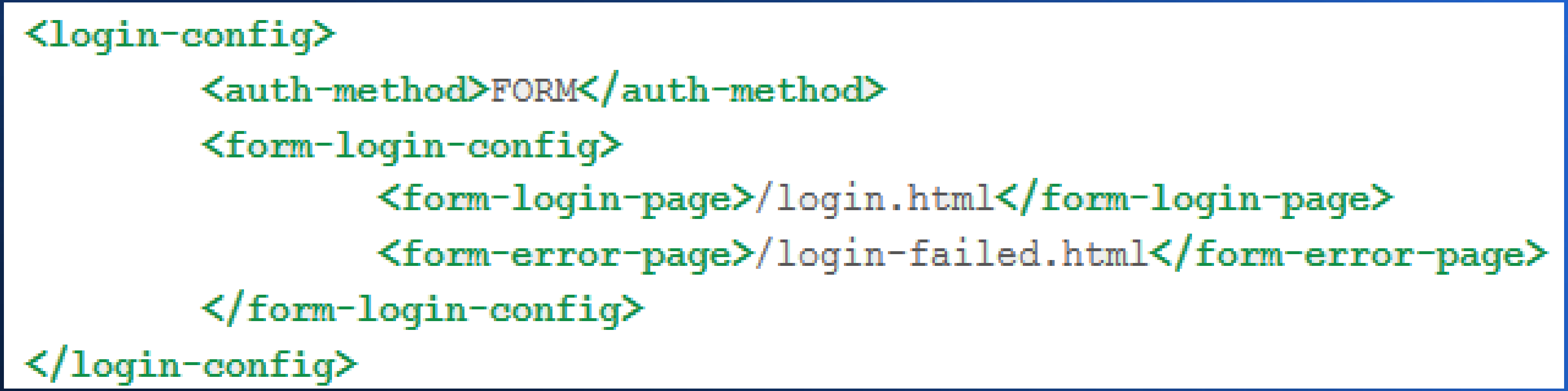
```
String authHeader = request.getHeader("authorization");
String encodedValue = authHeader.split(" ")[1];
out.println("Base64-encoded Authorization Value: " + encodedValue);
String decodedValue = Base64.base64Decode(encodedValue);
out.println("Base64-decoded Authorization Value: " + decodedValue);
```

http://www.avajava.com/tutorials/lessons/how-do-i-use-basic-authentication-with-tomcat.html?page=1

- This authentication should not be done without TLS/SSL
    - Sent username + password are only encoded

Form Authentication in Tomcat

- Creation of a login html page
- Creation of a failed login html page



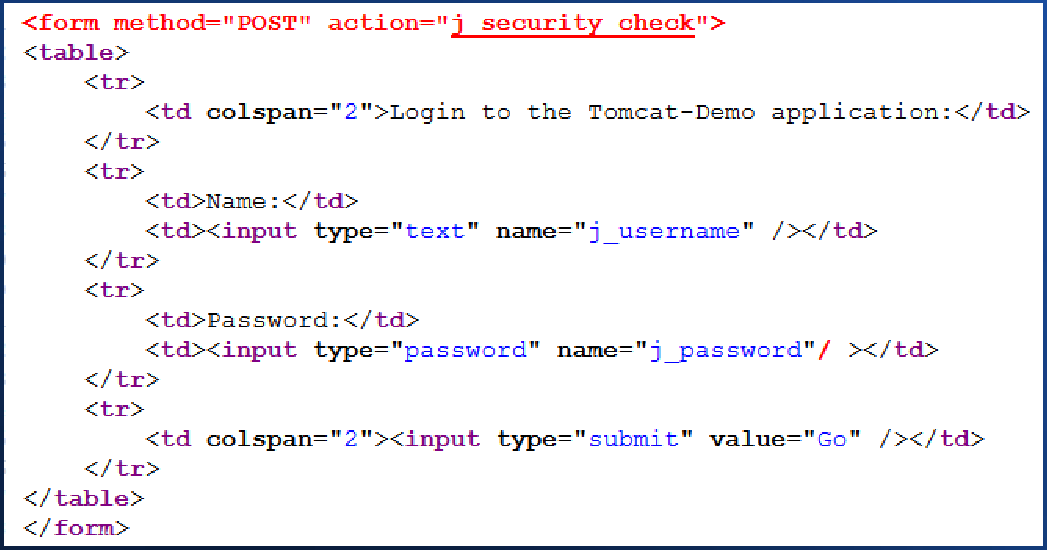
```
<login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
                <form-login-page>/login.html</form-login-page>
                <form-error-page>/login-failed.html</form-error-page>
        </form-login-config>
</login-config>
```

http://www.avajava.com/tutorials/lessons/how-do-i-use-form-authentication-with-tomcat.html?page=1

- Automatically creates sessions

Form Authentication in Tomcat

- Example login.html



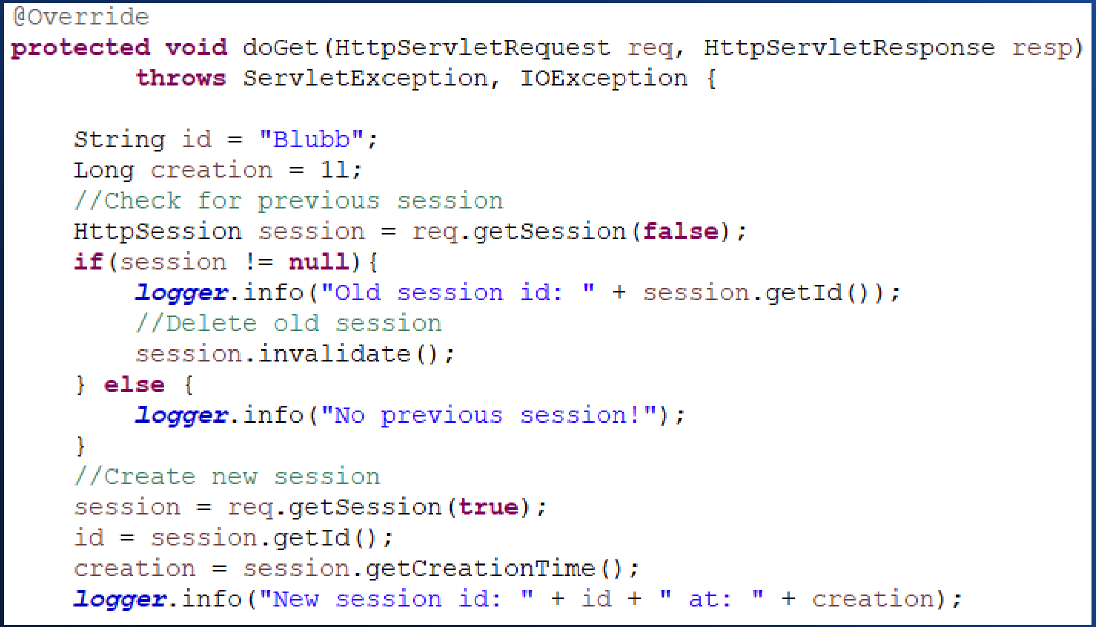
```
<form method="POST" action="j_security_check">
<table>
    <tr>
        <td colspan="2">Login to the Tomcat-Demo application:</td>
    </tr>
    <tr>
        <td>Name:</td>
        <td><input type="text" name="j_username" /></td>
    </tr>
    <tr>
        <td>Password:</td>
        <td><input type="password" name="j_password"/ ></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" value="Go" /></td>
    </tr>
</table>
</form>
```

http://www.avajava.com/tutorials/lessons/how-do-i-use-form-authentication-with-tomcat.html?page=1

- Form with:
  - action=“j_security_check“
  - 2 text fields with "j_username" & "j_password"

Managing Sessions

- HttpClient can manage sessions at each request
  - Session Ids from SecureRandom()
  - HttpSession, SSLSessionManager?



```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    String id = "Blubb";
    Long creation = 1l;
    //Check for previous session
    HttpSession session = req.getSession(false);
    if(session != null){
        logger.info("Old session id: " + session.getId());
        //Delete old session
        session.invalidate();
    } else {
        logger.info("No previous session!");
    }
    //Create new session
    session = req.getSession(true);
    id = session.getId();
    creation = session.getCreationTime();
    logger.info("New session id: " + id + " at: " + creation);
```

Managing Sessions

- .setAttribute(String, Object)
  - Adds information to the session; e.g. user name
- .getAttribute(String)
  - Retrieves the information described by the input
- .setMaxInactiveInterval(int)
  - Sets the maximum time between client requests before the session automatically terminates
- .invalidate() - Manuell terminates session

Managing Sessions

http://static2.fr.de/storage/image/4/2/3/7/657324_608x342_1oFefH_eH4orl.jpg

- Session tracking modes:
  - Cookie
    - The server sends the user a JSESSIONID cookie after authentication
    - The user uses the JSESSIONID in every following header to identify himself
  - SSL
  - URL
    - Rewriting the URL to include the ID

Managing Sessions

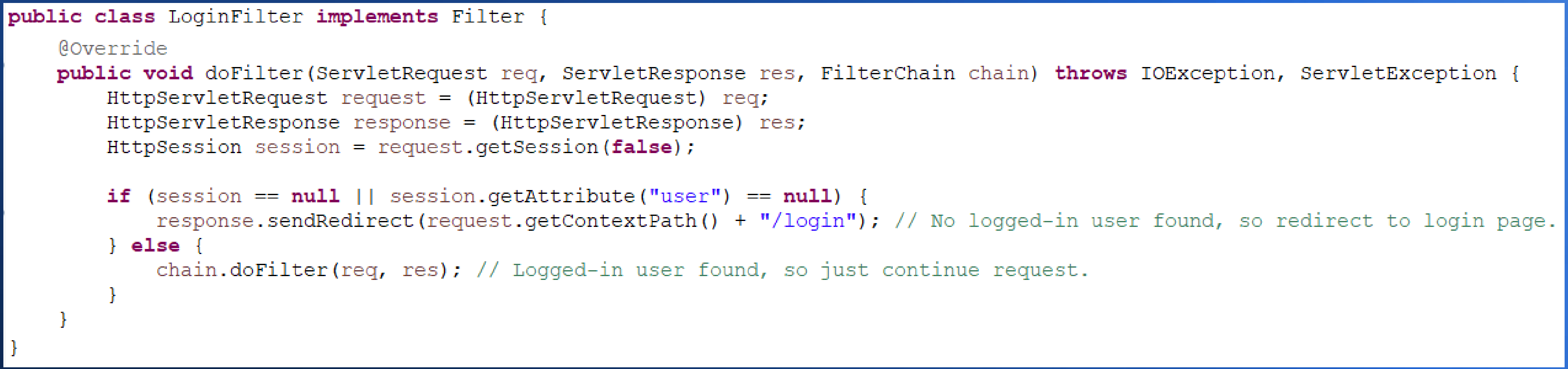
- Session tracking modes:
  - Can be changed at servlet start
    - ServletContextListener
    - Web.xml

```
<session-config>
        <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Authentication Filters

- Servlet Filters are used on every request/response before the actual servlet
  - Block requests / redirect
  - Modify request / response header & data
- E.g. to check if user sending the request is logged-in or not

Authentication Filters



```java
public class LoginFilter implements Filter {
    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) res;
        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("user") == null) {
            response.sendRedirect(request.getContextPath() + "/login"); // No logged-in user found, so redirect to login page.
        } else {
            chain.doFilter(req, res); // Logged-in user found, so just continue request.
        }
    }
}
```

https://stackoverflow.com/tags/servlet-filters/info

- doFilter() is called for every request
- chain.doFilter(req, res) sends the request to the next filter or to the servlet

Authentication Filters

- Add the filter to the web.xml
- Declare which URLs should be filtered
  - Do not filter the login page! -> loop
  - Filter specific servlets : <servlet-name> ... </..>

```
<filter>
  <filter-name>loginFilter</filter-name>
  <filter-class>org.semwebtech.servletdemo.LoginFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>loginFilter</filter-name>
  <url-pattern>/test/*</url-pattern>
</filter-mapping>
```

Aside: Servlet Listener

- Different Listener for different kinds of events
  - ServletContextListener
    - Startup, Shutdown of the servlet
  - HttpSessionListener
    - Session lifecycle events
  - ServletRequestListener
    - Request events

References

- http://www.avajava.com/tutorials/lessons/how-do-i-use-basic-authentication-with-tomcat.html
- http://www.avajava.com/tutorials/lessons/how-do-i-use-form-authentication-with-tomcat.html
- http://www.avajava.com/tutorials/lessons/how-do-i-use-basic-authentication-and-ssl-with-tomcat.html
- https://stackoverflow.com/questions/13274279/authentication-filter-and-servlet-for-login
- https://stackoverflow.com/questions/9965708/how-to-handle-authentication-authorization-with-users-in-a-database