Authentication & HTTPS (with Tomcat)

HTTPS

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

HTTP	HTTPS	Layer
HTTP	HTTP	Application
	TLS or SSL	Security
TCP	TCP	Transport
IP	IP	Network
Network interfaces	Network interfaces	Data link

- Hypertext Transfer Protocol Secure
  - 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

- Calculate two keys as a pair such that a message encrypted with one key can only be decrypted with the other one
- e.g. with RSA (Rivest-Shamir-Adleman)
  - (Simplified) Calculate e, d, (p * q) = N such that:
    - For a message m : $(m^e)^d \equiv m \ (mod \ N)$
    - Then follows: $(m^d)^e \equiv m \ (mod \ N)$

  $\Rightarrow$ (e, N) = public key

  $\Rightarrow$ (d, N) = 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
- A self-signed certificate is encrypted with your own private key
  - Others can use your public key to verify that you encrypted the Certificate
  - But no one should trust that self-signed certificates are 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)
  - Big ones: IdenTrust, Comodo, DigiCert, GoDaddy
  - Germany based: The German Country Signing Certificate Authority (CSCA), Bundesnetzargentur, D-Trust (Bundesdruckerei), ...
  - Free: Let's Encrypt, CAcert, ...

- 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 necessary if used just as a Servlet container; e.g. when using in combination with Apache Web Server

- Supported Certificate Keystores
  - JKS (Java Keystore)
  - PKCS11, PKCS12 (Public-Key Cryptography Standard)

Creation of a new JKS keystore

- Use the Java "keytool" program
  - Located in %JAVA_HOME%\bin
  - Commands
    - -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, ...)

    - -list : Prints the content of the keystore
      - -alias : Only the specified named key
      - -v / -rfc : Human-readable output
      - -keystore : Location & name of the keystore

    - -certreq : Generates a Certificate Signing Request (CSR)
      - -alias / -keystore / -sigalg / etc. as above

- Creating a key pair in a new 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 University of Göttingen
Wie lautet der Name Ihrer Stadt oder Gemeinde?
   [Unknown]: Göttingen
Wie lautet der Name Ihres Bundeslandes?
   [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 University of Göttingen,
    L=Göttingen, ST=Lower Saxony, C=NI richtig?
   [Nein]: ja

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

- Default name of the keystore ".keystore"
- Tomcat default password: "changeit"
- Self-signed certificates are not trustworthy, but good enough for a test

Enabling TLS/SSL in Tomcat

- Create keystore (optional for the test: certify it by sending a CSR to a CA)
- Find file "server.xml" in ..\Tomcat\bin
- Find 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>
```

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

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

- Multiple `<security-constraint>` can be declared to specifying separate security Constraints for different resources
- `<transport-guarantee>` can be
  - CONFIDENTIAL : prevent other entities from observing the content of the transmission
  - INTEGRAL : content can not be changed in transit
  - NONE : any kind of connection

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

- Tomcat-users.xml

```
<role rolename="tomcat"/>
<role rolename="role1"/>

<user username="alice" password="wonderland" roles="tomcat"/>
<user username="bob" password="builder" roles="role1"/>
<user username="trudy" password="vantubb" roles="tomcat,role1"/>
```

- Through the server.xml Tomcat can be configured to automatically hash passwords with simple hash functions (e.g. MD5)
  - Not really secure...

- Enabling Basic Authentication in the web.xml

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>BasicAuth</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>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
```

- The authentication information is also included in the request header
  - Base64-encoded
  - Available in the servlet through the "authorization" attribute

```
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);
```

  - This authentication should not be done without TLS/SSL

  → Sent username + password are only encoded, but not encrypted

Form Authentication

- Creation of a login html page
- Creation of a failed login html page
- Automatically called from Tomcat if the client has no session (is not logged in)
- Automatically creates sessions after login

- Enabling Form Authentication in the web.xml

```
<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>
```

- Creating an example login.html
- Needs a "form"-Element with
  - action="j_security_check"
  - 2 input text fields named "j_username" & "j_password"

```
<!DOCTYPE html>
<html>
    <body>
        <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="text" name="j_password"/></td>
            </tr>
            <tr>
                <td> colspan="2"><input type="submit" value="Go"/></td>
            </tr>
        </table>
        </form>
    </body>
</html>
```

[Filename: Servlet/formAuth.html]

DATABASE SECURITY RISKS & BEST PRACTICES

- Protect valuable & sensitive information not only from outside but inside risks as well
  - Excessive Privileges on Accounts
    - ⋆ Users can do much more than they need to

    → Managing user access rights
    → Query-level access control
      - · Define specific read/write functions
      - · Triggers

  - Privelege Abuse
    - ⋆ Users use their privileges inappropriately or fraudulently

    → Control policies on how data is accessed
    → Time of day, location, volume of data, etc...
    → Audit trails

- Unprotected Backup Data
  - Careless handling of old data devices

→ Archive and encrypt backup data

- Unmaintained database systems
  - No updates possible because of dependencies

→ Patching to fill security holes

→ Disable unused functionality

→ Use Intrusion Prevention Systems (IPS) to monitor and block known exploits

- Weak Authentication
  - Brute-force attacks, etc...

→ Use of modern hashing schemes (Argon2, etc..)

→ 2-way Authentication

- Database injection attacks
  - SQL/NoSQL injection attacks

→ Sanitize user inputs

- **Human error**
  - Social engineering, Reusing passwords, ...

→ Training employees
  - ⋆ Detect phishing attacks
  - ⋆ Internet & E-Mail usage
  - ⋆ Password management

SQL Injection Attacks

- Can happen when a database query is constructed with user input
- The user deliberately formulates the input in such a way that the query is misinterpreted and the database takes unintended actions

- Especially important if the database has a web interface

- Little Bobby Tables: `https://imgs.xkcd.com/comics/exploits_of_a_mom.png`

- Example code

```
String query = "SELECT * FROM Accounts
                WHERE username='" + username + "'
                AND password='" + password + "'";
```

Selecting everything

- User input

```
Username = "' OR '1' = '1"
Password = "' OR '' = '"
```

- Result

```
SELECT * FROM Accounts
WHERE username='' OR '1' = '1'
AND password ='' OR '' = ''
```

Deleting tables

- User input

```
Username = "Joe');DROP TABLE Accounts; --"
```

- Result

```
SELECT * FROM Accounts
WHERE username='Joe');DROP TABLE Accounts;
--AND password ='''";
```

Check your websites for vulnerabilities

- Automated SQL Injection Attack Tools
  - Havij (ITSecTeam, an Iranian security company)
  - SQLmap (open source; sqlmap.org)
  - jSQL (open source; github.com/ron190/jsql-injection)
  - TyrantSQL (open source; GUI version of SQLmap; sourceforge.net/projects/tyrantsql?source=directory)

Countermeasures

- Input sanitization
  - Check the input for dangerous characters
  - E.g. escape ’ or "
  - Careful! Characters can be encoded differently, but still be interpreted by your system
    - Example Login over HTTP GET
    - `Login.html?user=Joe’;Drop Table Accounts;--`
      OR
    - `Login.html?user=Joe%27%3bDrop%20Table%20Accounts%3b--`

- Validation
  - Check if the input data is in the format you want it to be
    - E-mails contain an @
    - ID containing only numbers
    - Length of the input
  - White- or Blacklist characters

- Prepared statements
  - The SQL query is precompiled by the DBMS
    - Also benefit of being faster if the same statement is executed multiple times
  - Allows the use of parameters to change variable values for each execute
    - Represented by markers:

      @ (ASP.NET)
      : (PHP)
      ? (JAVA)
    - Parameter values are added to the query at execution time in a controlled manner
      - "The SQL engine checks each parameter that it is correct for its column and are treated literally, and not as part of the SQL to be executed"
    - **But!** Parameters are not allowed for identifiers (table & column names)

- Prepared statements
  - JAVA example

```
String sql = "SELECT * FROM Accounts WHERE username = ? AND password = ?";

PreparedStatement prepStmnt = con.prepareStatement(sql);

prepStmnt.setString(1, "john");
prepStmnt.setString(2, "doe");

ResultSet result = prepStmnt.executeQuery();
```

  - Set parameters with functions: `set<TYPE>(Index, Value)`
    - For all Java data type: setString, setLong, setDouble, setBytes, ...
    - Index is the number of the parameter placeholder
  - The PreparedStatement is bound to the connection
    - If you close the Connection, the PreparedStatement can not be executed anymore
    - A Connection Pool can be used to handle the Connection if the PreparedStatement is executed periodically over a long time
      - Maintains the Connection, but sets it to sleep mode instead of closing it

- Are prepared statements 100% safe?
  - Not necessarily...
  - The inner workings depend on the driver
    - Some only emulate prepared statements
  - Constructing the query using string concatenation with user input makes it unsafe again
    - Best procedure when an identifier needs to be variable?
  - 2^{nd} order injection attacks
    - 1^{st} order: the user input data is unsafe
    - But what if the database data is also unsafe?
      - It originates from the user most of the time
        Consider that "Joe');DROP TABLE Accounts; - -" is a stored user name

    → Do not even trust the data inside the database

⇒ Proper use of prepared statements prevent most injection attacks, but there are always other attack vectors!