University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Apsi-Lab 2 Sichere Bearbeitung von HTML-Formularen

Jan Fässler & Fabio Oesch

18. Dezember 2013

Zusammenfassung

Im Apsi-Lab 2 sollte eine Webseite mittels Servlets gebaut werden. Die Ziele waren Separation of privilege, Complete mediation, Fail-save defaults und Availability and reliability. Um sicher zu gehen, dass die Rechte auseiander gehalten werden, wurde ein Benutzer erstellt der nur auf die Datenbank schreiben und bearbeiten kann. Jeder Input wird mit einem Regex String verglichen damit diese keine Zeichen beinhalten können welche zu Angriffen benutzt werden können. Falls eine Attacke (SQL Injection oder XSS Attacke) versucht wird. Wird dies verhindert indem die Zeichen, die für die jeweilige Attacke benutzt werden, verändert werden. Die Applikation sollte ohne Probleme 50 Person parallel verarbeiten können. Dies ist jedoch abhängig vom Server.

Inhaltsverzeichnis

Institute of computer science,
www.fhnw.ch

1 Einleitung

1.1 Ziel

Die Bearbeitung von HTML-Formularen bildet das Rückgrat vieler Web-Applikationen. Mit dieser Laborübung sollten Sie eine erste Schutzschicht (im Sinne von Saltzer und Schröder) in Ihrer Web-Applikation sorgfältig planen und entwickeln. Sie sollen Ihre Aufmerksamkeit auf die folgenden Grundprinzipien richten:

1. Separation of privilege: Halten Sie Rechte sorgfältig auseinander. Konkret will heissen, dass die Programmschicht, welche die Eingaben des Benutzers sammelt, nicht auch für den eigentlichen Login des registrierten Benutzers verwendet werden darf.
2. Complete mediation: Jeder Input/Output muss verifiziert werden. Sie sollten im Voraus bestimmen, wie die erlaubte Information aussehen darf, und dann strikt die Kriterien für ihre Validierung formulieren.
3. Fail-safe defaults: Falls der Benutzer unbewusst (oder bewusst) falsche Informationen via HTML-Formular einschleusen will, sollte Ihre Applikation nur mit knappen Hinweisen reagieren und vermeiden, dass der Benutzer die Applikation allzu lange beansprucht (d.h. Sie sollten eine geeignete fail-safe Strategie entwerfen).
4. Availability and reliability: Ihre Applikation soll maximal 50 Benutzer in parallel (ohne nennenswerte Einbussen in der Performanz) bedienen können. Überlegen Sie, welche Implikationen diese Forderung für die Sicherheit Ihrer Applikation beinhaltet.

1.2 Ergebniss

Mit unserer Applikation ist es möglich, dass ein anderer Computer auf den Server zugreift. Um sich einzuloggen muss ein Zertifikat heruntergeladen werden. Es ist auch nicht möglich, eine SQL-Injection oder Cross-Site Scripting zu machen.

2 Klassenbeschreibung

In diesem Abschnitt werden kurz die wichtigsten Klassen diskutiert.

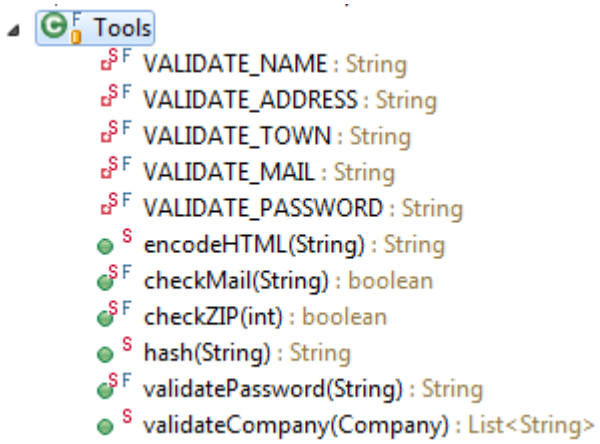
2.1 Company

Die Klasse Company ist für die Verbindung der Datenbank mit den Inputdaten des User verantwortlich. Die Methoden checkLogin(), save(), changePassword und createUsername() greifen auf die Datenbank zu. Der Input wird mit findbugs überprüft um sicher zu gehen, dass wir nicht falsche Informationen in die Datenbank speichern.

2.2 servlet Package

Die Klassen in diesem Package haben alle ein doGet() und doPost() (ausser RattleBitsApplication dieses hat nur eine doGet) welche die GET/POST Anfragen behandeln.

2.3 Tools



Diese Klasse ist für die Validierung der möglichen Eingaben. Wie die Validierung funktioniert wird in einem späteren Kapitel noch genauer erklärt.

3 Robust Programming

3.1 SQL-Injection & XSS Attacken

Damit keine Möglichkeit besteht SQL-Injections und Cross-Site Scripting Attacken zu verwenden wurde die Methode $Tools.encodeHTML()$ erzeugt. Es wird überprüft, ob der eingegebene Text Zeichen beinhaltet, die zu SQL-Injections oder XSS Attacken führen könnten. Diese Zeichen “,< und> wurden aber abgefangen und ersetzt mit mit **”&#“ + (int) c + ”;“**. Wobei c das momentane Zeichen darstellt. So ist sichergestellt, dass keine SQL-Injections oder XSS Attacken durchgeführt werden können.

Listing 1: HTML Encoding

```
/**
 * Encodes html tags
 * @param s string
 * @return encoded string
 */
@Nonnull
public static String encodeHTML(@Nonnull String s) {
    StringBuffer out = new StringBuffer();
    for(int i=0; i<s.length(); i++) {
        char c = s.charAt(i);
        if(c > 127 || c=='"' || c=='<' || c=='>') out.append("&#"+(int)c+";");
        else out.append(c);
    }
    return out.toString();
}
```

3.2 Validierung

Wie schon angetönt wurde ist die Klasse Tools für die Validierung der verschiedenen Felder verantwortlich. Es wird nun noch tiefer auf die verschiedenen Methoden eingegangen.

3.2.1 Variablen

Die finalen Variablen sind Regexausdrücke. Diese werden benutzt um die Eingaben des Benuters auf Richtigkeit zu überprüfen.

Listing 2: Final Variablen

```
private static final String VALIDATE_NAME     = "([ôÔêÊâÂèéÈÉäöüÄÖÜß\\-\\._\\w]+)";
private static final String VALIDATE_ADDRESS  = "([ôÔêÊâÂèéÈÉäöüÄÖÜß\\-\\._\\w\\d ]+)";
private static final String VALIDATE_TOWN     = "([ôÔêÊâÂèéÈÉäöüÄÖÜß\\-\\._\\w\\s\\d]+)";
private static final String VALIDATE_MAIL     = "([^.@]+)(\\.[^.@]+)*@([^.@]+\\.)+([^.@]+)"
    ;
private static final String VALIDATE_PASSWORD = "([ôÔêÊâÂèéÈÉäöüÄÖÜß\\-\\._\\w]+)";
```

3.2.2 E-Mail Adresse überprüfen

Es wird überprüft ob der die E-Mail Adresse die Angegeben wurde auch gültig ist.

Listing 3: E-Mail Adresse überprüfen

```
/**
 * Checks if a mail is valid
 * @param mail email adress
 * @return state if the email is valid
 */
@CheckReturnValue
public static final boolean checkMail(@Nonnull String mail) {
  Hashtable<String, String> env = new Hashtable<String, String>();
  env.put("java.naming.factory.initial", "com.sun.jndi.dns.DnsContextFactory");
  try {
    String[] temp = mail.split("@");
    Attributes attrs = (new InitialDirContext(env)).getAttributes(temp[1], new String[] {"
        MX"});
    return !(attrs.get("MX") == null);
  } catch (NamingException e) { return false; }
}
```

3.2.3 Postleitzahl überprüfen

Diese Methode wird im späteren Verlauf noch präziser erklärt.

3.2.4 Passwort überprüfen

Ein gültiges Passwort muss mindestens aus 8 und maximal aus 64 Zeichen bestehen. Sonst ist das Passwort ungültig und teilt dies dem Benutzer mit.

Listing 4: Passwort überprüfen

```
/**
 * Validates the given password
 * @param pw password
 * @return state if password is valid
 */
public static final String validatePassword(String pw) {
  if (pw != null) {
    String s = pw.trim();
    if (s.length() < 8 || s.length() > 64) {  return "Passwort zu kurz/lang (min. 8 Zeichen
        /max. 64 Zeichen).";  }
    else if (!s.matches(VALIDATE_PASSWORD)) { return "Ungültige Zeichen im Passwort."; }
  } else {                                      return "Passwort eingeben."; }
  return null;
}
```

3.2.5 Company überprüfen

Diese Methode ist für die Verwaltung der Validierung zuständig.

1. Als erstes wird der Name der Firma überprüft. Dieser darf nicht leer sein, mehr als 20 oder ungültige Zeichen beinhalten. Falls einer dieser 3 Bedingungen verletzt wird, wird dem Benutzer mitgeteilt, welche der Bedingungen verletzt ist.
2. Als nächstes wird die Adresse überprüft. Die Adresse darf nicht leer sein und keine ungültigen Zeichen beinhalten.
3. Die Postleitzahl muss im Bereich zwischen $999 < x < 10000$ sein. Zusätzlich muss die Postleitzahl existieren, was wir mit $post.ch$ überprüfen.
4. Die Stadt muss folgende Bedingungen erfüllen. Das Feld darf nicht leer sein oder ungültige Zeichen benutzen.
5. Zuletzt wird validiert ob die eingetragene E-Mail Adresse leer ist oder schon einmal benutzt wurde. Zusätzlich wird mit $checkMail()$ überprüft ob diese gültig ist.

Listing 5: Company überprüfen

```
  /**
2  * Validates the fields
   * @return Error messages of the validation
   */
  @Nonnull
  @CheckReturnValue
7 public static List<String> validateCompany(Company c) {
    List<String> errors = new ArrayList<>();
    String s = c.getName();
      if (s != null) {
        s = s.trim();
12      if (s.isEmpty()) {                    errors.add("Firmenname eingeben."); }
        else if (s.length() > 20) {           errors.add("Firmenname zu lang (max. 20 Zeichen
            )."); }
        else if (!s.matches(VALIDATE_NAME)) { errors.add("Ung&uuml;ltige Zeichen im
            Firmennamen"); }
      } else                     errors.add("Firmenname eingeben.");

17    s = c.getAddress();
      if (s != null) {
        s = s.trim();
        if (s.isEmpty()) {                   errors.add("Keine Adresse."); }
        else if (!s.matches(VALIDATE_ADDRESS)) { errors.add("Ung&uuml;ltige Adresse."); }
22    } else                           errors.add("Keine Adresse.");

      int zip = c.getZip();
      if (zip < 1000 || zip > 9999 || !Tools.checkZIP(zip))
        errors.add("Ung&uuml;ltige Postleitzahl.");
27
      s = c.getTown();
      if (s != null) {
        s = s.trim();
        if (s.isEmpty()) {            errors.add("Keine Stadt."); }
32      else if (!s.matches(VALIDATE_TOWN)) { errors.add("Ung&uuml;ltige Stadt."); }
      } else                     errors.add("Keine Stadt.");

      s = c.getMail();
      if (s != null) {
37      s = s.trim();
        if (s.isEmpty() || !s.matches(VALIDATE_MAIL) || !Tools.checkMail(s)) {
            errors.add("Ung&uuml;ltige Email-Adresse.");
          }
      } else errors.add("Keine Email-Adresse");
42
    return errors;
  }
```

3.2.6 Postleitzahl Validierung

Die Postleitzahl wird mit der Hilfe von post.ch überprüft. Bevor abgefragt wird, ob die Postleitzahl existiert, wird der String in eine Zahl umgewandelt. Dies kann zu einer Exception führen und dann wird zurückgegeben, dass die eingegebene Postleitzahl keine gültige ist.
Falls die Postleitzahl nicht existiert wird auf der Seite “Keine PLZ gefunden” wissen wir, dass die eingegebene Postleitzahl nicht existiert. Es wird hingegen nicht überprüft ob die eingegebene Postleitzahl mit der Stadt übereinstimmt.

Listing 6: Überprüft Postleitzahl

```
/**
 * Validates the zip code with the post.ch website
 * @param zip zip
 * @return state if the zip is valid
 */
@CheckReturnValue
public static final boolean checkZIP(int zip) {
  String line;
  BufferedReader rd = null;
  try {
    HttpURLConnection conn = (HttpURLConnection) (new URL("http://www.post.ch/db/owa/
        pv_plz_pack/pr_check_data?p_language=de&p_nap="+zip)).openConnection();
    conn.setRequestMethod("GET");
    rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    while ((line = rd.readLine()) != null) if(line.contains("Keine PLZ gefunden")) return
        false;
  } catch (IOException e) { System.err.println(e.getMessage()); }
  finally { if (rd != null) try { rd.close(); } catch (IOException e) {} }
  return true;
}
```

3.3 Schutz der Model-Klasse vor schädlichem Missbrauch

Die Schwierigkeit einer korrekten Login-implementation liegt darin, den Ablauf so zu gestalten dass nur dieser Ablauf zu einer validen Änderung führt. Heisst nur beim vorgesehenen Ablauf können nur korrekte Daten geschrieben werden, alles andere darf die Datenbank nicht verändern. So sind beispielsweise setters auf das Passwort problematisch. Zusammen mit der $.save()$ Methode könnte ein Fehlerhafter oder bösartiger Code das Passwort eines beliebigen Users überschreiben. Usernamen und ID sind ebenfalls Felder, auf die nur lesend zugegriffen werden soll.

Um all dies zu verhindern, ist unsere Company Klasse Immutable.

4 SSL

Damit eine SSL-Verbindung hergestellt werden kann muss man im server.xml etwas aus dem Kommentar herausnehmen.

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443
     This connector uses the JSSE configuration, when using APR, the
     connector should be using the OpenSSL style configuration
     described in the APR documentation -->

<Connector port="8443" maxThreads="150" scheme="https" secure="true" SSLEnabled="true"
keystoreFile="conf/RattleBitsKeyStore" keystorePass="apsilab" clientAuth="false"
keyAlias="RattleBits" sslProtocol="TLS"/>
```

Zusätzlich muss man mit dem keystore einen eigenen Schlüssel erstellen. Damit Login auch eine SSL Verbindung herstellen kann wird im web.xml ein Security Constraint eingebaut.

```
  <security-constraint>
    <web-resource-collection>
        <web-resource-name>ApsiUebung2</web-resource-name>
        <url-pattern>/RattleBits/Login</url-pattern>
        <url-pattern>/RattleBits/Overview</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```