  1    
  2  package org.owasp.webgoat.lessons;  
  3    
  4  import java.io.IOException;  
  5  import java.net.URLDecoder;  
  6  import java.net.URLEncoder;  
  7  import java.nio.ByteBuffer;  
  8  import java.nio.CharBuffer;  
  9  import java.nio.charset.Charset;  
 10  import java.nio.charset.CharsetDecoder;  
 11  import java.nio.charset.CharsetEncoder;  
 12  import java.security.MessageDigest;  
 13  import java.security.NoSuchAlgorithmException;  
 14  import java.util.ArrayList;  
 15  import java.util.List;  
 16  import javax.crypto.Cipher;  
 17  import javax.crypto.SecretKey;  
 18  import javax.crypto.SecretKeyFactory;  
 19  import javax.crypto.spec.PBEParameterSpec;  
 20  import org.apache.ecs.Element;  
 21  import org.apache.ecs.ElementContainer;  
 22  import org.apache.ecs.html.A;  
 23  import org.apache.ecs.html.B;  
 24  import org.apache.ecs.html.IMG;  
 25  import org.apache.ecs.html.Input;  
 26  import org.apache.ecs.html.P;  
 27  import org.apache.ecs.html.TD;  
 28  import org.apache.ecs.html.TR;  
 29  import org.apache.ecs.html.Table;  
 30  import org.owasp.webgoat.session.ECSFactory;  
 31  import org.owasp.webgoat.session.WebSession;  
 32  import org.owasp.webgoat.util.HtmlEncoder;  
 33    
 34    
 35  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 36   \*   
 37   \*   
 38   \* This file is part of WebGoat, an Open Web Application Security Project utility. For details,  
 39   \* please see http://www.owasp.org/  
 40   \*   
 41   \* Copyright (c) 2002 - 2007 Bruce Mayhew  
 42   \*   
 43   \* This program is free software; you can redistribute it and/or modify it under the terms of the  
 44   \* GNU General Public License as published by the Free Software Foundation; either version 2 of the  
 45   \* License, or (at your option) any later version.  
 46   \*   
 47   \* This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without  
 48   \* even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  
 49   \* General Public License for more details.  
 50   \*   
 51   \* You should have received a copy of the GNU General Public License along with this program; if  
 52   \* not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  
 53   \* 02111-1307, USA.  
 54   \*   
 55   \* Getting Source ==============  
 56   \*   
 57   \* Source for this application is maintained at code.google.com, a repository for free software  
 58   \* projects.  
 59   \*   
 60   \* For details, please see http://code.google.com/p/webgoat/  
 61   \*   
 62   \* @author Jeff Williams <a href="http://www.aspectsecurity.com">Aspect Security</a>  
 63   \* @created October 28, 2003  
 64   \*/  
 65    
 66  public class Encoding extends LessonAdapter  
 67  {  
 68      public final static A ASPECT\_LOGO = new A().setHref("http://www.aspectsecurity.com")  
 69              .addElement(  
 70                          new IMG("images/logos/aspect.jpg").setAlt("Aspect Security").setBorder(0).setHspace(0)  
 71                                  .setVspace(0));  
 72    
 73      private final static String INPUT = "input";  
 74    
 75      private final static String KEY = "key";  
 76    
 77      // local encoders  
 78    
 79      private static sun.misc.BASE64Decoder decoder = new sun.misc.BASE64Decoder();  
 80    
 81      private static sun.misc.BASE64Encoder encoder = new sun.misc.BASE64Encoder();  
 82    
 83      // encryption constant  
 84    
 85      private static byte[] salt = { (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,  
 86              (byte) 0x00, (byte) 0x00 };  
 87    
 88      /\*\*  
 89       \* Returns the base 64 decoding of a string.  
 90       \*   
 91       \* @param str  
 92       \*            Description of the Parameter  
 93       \* @return Description of the Return Value  
 94       \* @exception IOException  
 95       \*                Description of the Exception  
 96       \*/  
 97    
 98      public static String base64Decode(String str) throws IOException  
 99      {  
100    
101          byte[] b = decoder.decodeBuffer(str);  
102    
103          return (new String(b));  
104      }  
105    
106      /\*\*  
107       \* Description of the Method  
108       \*   
109       \* @param c  
110       \*            Description of the Parameter  
111       \* @return Description of the Return Value  
112       \* @exception IOException  
113       \*                Description of the Exception  
114       \*/  
115    
116      public static String base64Decode(char[] c) throws IOException  
117      {  
118    
119          return base64Decode(new String(c));  
120      }  
121    
122      /\*\*  
123       \* Description of the Method  
124       \*   
125       \* @param c  
126       \*            Description of the Parameter  
127       \* @return Description of the Return Value  
128       \*/  
129    
130      public static String base64Encode(char[] c)  
131      {  
132    
133          return base64Encode(new String(c));  
134      }  
135    
136      /\*\*  
137       \* Returns the base 64 encoding of a string.  
138       \*   
139       \* @param str  
140       \*            Description of the Parameter  
141       \* @return Description of the Return Value  
142       \*/  
143    
144      public static String base64Encode(String str)  
145      {  
146    
147          byte[] b = str.getBytes();  
148    
149          return (encoder.encode(b));  
150      }  
151    
152      /\*\*  
153       \* Description of the Method  
154       \*   
155       \* @param b  
156       \*            Description of the Parameter  
157       \* @return Description of the Return Value  
158       \*/  
159    
160      public static String base64Encode(byte[] b)  
161      {  
162    
163          return (encoder.encode(b));  
164      }  
165    
166      /\*\*  
167       \* Description of the Method  
168       \*   
169       \* @param s  
170       \*            Description of the Parameter  
171       \* @return Description of the Return Value  
172       \*/  
173    
174      protected Element createContent(WebSession s)  
175      {  
176    
177          ElementContainer ec = new ElementContainer();  
178    
179          try  
180          {  
181    
182              String userInput = s.getParser().getRawParameter(INPUT, "");  
183    
184              String userKey = s.getParser().getStringParameter(KEY, "");  
185    
186              Table table = new Table();  
187    
188              TR tr = new TR();  
189    
190              tr.addElement(new TD("Enter a string: "));  
191    
192              Input input = new Input(Input.TEXT, INPUT, userInput);  
193    
194              tr.addElement(new TD().addElement(input));  
195    
196              table.addElement(tr);  
197    
198              tr = new TR();  
199    
200              tr.addElement(new TD("Enter a password (optional): "));  
201    
202              Input key = new Input(Input.TEXT, KEY, userKey);  
203    
204              tr.addElement(new TD().addElement(key));  
205    
206              table.addElement(tr);  
207    
208              tr = new TR();  
209    
210              Element b = ECSFactory.makeButton("Go!");  
211    
212              tr.addElement(new TD().setAlign("center").setColSpan(2).addElement(b));  
213    
214              table.addElement(tr);  
215    
216              ec.addElement(table);  
217    
218              ec.addElement(new P());  
219    
220              Table t = new Table();  
221    
222              t.setWidth("100%");  
223    
224              t.setBorder(0);  
225    
226              t.setCellSpacing(1);  
227    
228              t.setCellPadding(4);  
229    
230              String description;  
231    
232              t.addElement(makeTitleRow("Description", "Encoded", "Decoded"));  
233    
234              description = "Base64 encoding is a simple reversable encoding used to encode bytes into ASCII characters. Useful for making bytes into a printable string, but provides no security.";  
235    
236              // t.addElement( makeDescriptionRow( description ) );  
237              t.addElement(makeRow(description, base64Encode(userInput), base64Decode(userInput)));  
238              // t.addElement( makeSpacerRow() );  
239    
240              description = "Entity encoding uses special sequences like &amp;amp; for special characters. This prevents these characters from being interpreted by most interpreters.";  
241    
242              t.addElement(makeRow(description, HtmlEncoder.encode(userInput), HtmlEncoder.decode(userInput)));  
243    
244              description = "Password based encryption (PBE) is strong encryption with a text password. Cannot be decrypted without the password";  
245    
246              t.addElement(makeRow(description, encryptString(userInput, userKey), decryptString(userInput, userKey)));  
247              description = "MD5 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes. For obscure cryptographic reasons, it is better to use SHA-256 if you have a choice.";  
248    
249              t.addElement(makeRow(description, hashMD5(userInput), "Cannot reverse a hash"));  
250    
251              description = "SHA-256 hash is a checksum that can be used to validate a string or byte array, but cannot be reversed to find the original string or bytes.";  
252    
253              t.addElement(makeRow(description, hashSHA(userInput), "N/A"));  
254    
255              description = "Unicode encoding is...";  
256    
257              t.addElement(makeRow(description, "Not Implemented", "Not Implemented"));  
258    
259              description = "URL encoding is...";  
260    
261              t.addElement(makeRow(description, urlEncode(userInput), urlDecode(userInput)));  
262    
263              description = "Hex encoding simply encodes bytes into %xx format.";  
264    
265              t.addElement(makeRow(description, hexEncode(userInput), hexDecode(userInput)));  
266    
267              description = "Rot13 encoding is a way to make text unreadable, but is easily reversed and provides no security.";  
268    
269              t.addElement(makeRow(description, rot13(userInput), userInput));  
270    
271              description = "XOR with password encoding is a weak encryption scheme that mixes a password into data.";  
272    
273              t.addElement(makeRow(description, xorEncode(userInput, userKey), xorDecode(userInput, userKey)));  
274    
275              description = "Double unicode encoding is...";  
276    
277              t.addElement(makeRow(description, "Not Implemented", "Not Implemented"));  
278    
279              description = "Double URL encoding is...";  
280    
281              t.addElement(makeRow(description, urlEncode(urlEncode(userInput)), urlDecode(urlDecode(userInput))));  
282    
283              ec.addElement(t);  
284    
285          }  
286    
287          catch (Exception e)  
288          {  
289    
290              s.setMessage("Error generating " + this.getClass().getName());  
291    
292              e.printStackTrace();  
293    
294          }  
295    
296          if (getLessonTracker(s).getNumVisits() > 3)  
297          {  
298              makeSuccess(s);  
299          }  
300    
301          return (ec);  
302      }  
303    
304      /\*\*  
305       \* Convenience method for encrypting a string.  
306       \*   
307       \* @param str  
308       \*            Description of the Parameter  
309       \* @param pw  
310       \*            Description of the Parameter  
311       \* @return String the encrypted string.  
312       \*/  
313    
314      public static synchronized String decryptString(String str, String pw)  
315      {  
316    
317          try  
318          {  
319    
320              PBEParameterSpec ps = new javax.crypto.spec.PBEParameterSpec(salt, 20);  
321    
322              SecretKeyFactory kf = SecretKeyFactory.getInstance("PBEWithMD5AndDES");  
323    
324              Cipher passwordDecryptCipher = Cipher.getInstance("PBEWithMD5AndDES/CBC/PKCS5Padding");  
325    
326              char[] pass = pw.toCharArray();  
327    
328              SecretKey k = kf.generateSecret(new javax.crypto.spec.PBEKeySpec(pass));  
329    
330              passwordDecryptCipher.init(Cipher.DECRYPT\_MODE, k, ps);  
331    
332              byte[] dec = decoder.decodeBuffer(str);  
333    
334              byte[] utf8 = passwordDecryptCipher.doFinal(dec);  
335    
336              return new String(utf8, "UTF-8");  
337          }  
338    
339          catch (Exception e)  
340          {  
341    
342              return ("This is not an encrypted string");  
343          }  
344    
345      }  
346    
347      /\*\*  
348       \* Convenience method for encrypting a string.  
349       \*   
350       \* @param str  
351       \*            Description of the Parameter  
352       \* @param pw  
353       \*            Description of the Parameter  
354       \* @return String the encrypted string.  
355       \* @exception SecurityException  
356       \*                Description of the Exception  
357       \*/  
358    
359      public static synchronized String encryptString(String str, String pw) throws SecurityException  
360      {  
361    
362          try  
363          {  
364    
365              PBEParameterSpec ps = new javax.crypto.spec.PBEParameterSpec(salt, 20);  
366    
367              SecretKeyFactory kf = SecretKeyFactory.getInstance("PBEWithMD5AndDES");  
368    
369              Cipher passwordEncryptCipher = Cipher.getInstance("PBEWithMD5AndDES/CBC/PKCS5Padding");  
370    
371              char[] pass = pw.toCharArray();  
372    
373              SecretKey k = kf.generateSecret(new javax.crypto.spec.PBEKeySpec(pass));  
374    
375              passwordEncryptCipher.init(Cipher.ENCRYPT\_MODE, k, ps);  
376    
377              byte[] utf8 = str.getBytes("UTF-8");  
378    
379              byte[] enc = passwordEncryptCipher.doFinal(utf8);  
380    
381              return encoder.encode(enc);  
382          }  
383    
384          catch (Exception e)  
385          {  
386    
387              return ("Encryption error");  
388          }  
389    
390      }  
391    
392      /\*\*  
393       \* Gets the category attribute of the Encoding object  
394       \*   
395       \* @return The category value  
396       \*/  
397    
398      protected Category getDefaultCategory()  
399      {  
400          return Category.INSECURE\_STORAGE;  
401      }  
402    
403      /\*\*  
404       \* Gets the hints attribute of the HelloScreen object  
405       \*   
406       \* @return The hints value  
407       \*/  
408    
409      public List<String> getHints(WebSession s)  
410      {  
411    
412          List<String> hints = new ArrayList<String>();  
413          hints.add("Enter a string and press 'go'");  
414          hints.add("Enter 'abc' and notice the rot13 encoding is 'nop' ( increase each letter by 13 characters ).");  
415          hints.add("Enter 'a c' and notice the url encoding is 'a+c' ( ' ' is converted to '+' ).");  
416          return hints;  
417      }  
418    
419      /\*\*  
420       \* Gets the instructions attribute of the Encoding object  
421       \*   
422       \* @return The instructions value  
423       \*/  
424    
425      public String getInstructions(WebSession s)  
426      {  
427          return "This lesson will familiarize the user with different encoding schemes.  ";  
428      }  
429    
430      private final static Integer DEFAULT\_RANKING = new Integer(15);  
431    
432      protected Integer getDefaultRanking()  
433      {  
434          return DEFAULT\_RANKING;  
435      }  
436    
437      /\*\*  
438       \* Gets the title attribute of the HelloScreen object  
439       \*   
440       \* @return The title value  
441       \*/  
442    
443      public String getTitle()  
444      {  
445          return ("Encoding Basics");  
446      }  
447    
448      /\*\*  
449       \* Returns the MD5 hash of a String.  
450       \*   
451       \* @param str  
452       \*            Description of the Parameter  
453       \* @return Description of the Return Value  
454       \*/  
455    
456      public static String hashMD5(String str)  
457      {  
458    
459          byte[] b = str.getBytes();  
460          MessageDigest md = null;  
461    
462          try  
463          {  
464              md = MessageDigest.getInstance("MD5");  
465              md.update(b);  
466          } catch (NoSuchAlgorithmException e)  
467          {  
468              // it's got to be there  
469              e.printStackTrace();  
470          }  
471          return (base64Encode(md.digest()));  
472      }  
473    
474      /\*\*  
475       \* Returns the SHA hash of a String.  
476       \*   
477       \* @param str  
478       \*            Description of the Parameter  
479       \* @return Description of the Return Value  
480       \*/  
481    
482      public static String hashSHA(String str)  
483      {  
484          byte[] b = str.getBytes();  
485          MessageDigest md = null;  
486          try  
487          {  
488              md = MessageDigest.getInstance("SHA-256");  
489              md.update(b);  
490          } catch (NoSuchAlgorithmException e)  
491          {  
492              // it's got to be there  
493              e.printStackTrace();  
494          }  
495          return (base64Encode(md.digest()));  
496      }  
497    
498      /\*\*  
499       \* Description of the Method  
500       \*   
501       \* @param hexString  
502       \*            Description of the Parameter  
503       \* @return Description of the Return Value  
504       \*/  
505    
506      public static String hexDecode(String hexString)  
507      {  
508          try  
509          {  
510              if ((hexString.length() % 3) != 0) { return ("String not comprised of Hex digit pairs."); }  
511              char[] chars = new char[hexString.length()];  
512              char[] convChars = new char[hexString.length() / 3];  
513              hexString.getChars(0, hexString.length(), chars, 0);  
514              for (int i = 1; i < hexString.length(); i += 3)  
515              {  
516                  String hexToken = new String(chars, i, 2);  
517                  convChars[i / 3] = (char) Integer.parseInt(hexToken, 16);  
518              }  
519              return new String(convChars);  
520          } catch (NumberFormatException nfe)  
521          {  
522              return ("String not comprised of Hex digits");  
523          }  
524      }  
525    
526      /\*\*  
527       \* Description of the Method  
528       \*   
529       \* @param asciiString  
530       \*            Description of the Parameter  
531       \* @return Description of the Return Value  
532       \*/  
533    
534      public static String hexEncode(String asciiString)  
535      {  
536          char[] ascii = new char[asciiString.length()];  
537          asciiString.getChars(0, asciiString.length(), ascii, 0);  
538          StringBuffer hexBuff = new StringBuffer();  
539          for (int i = 0; i < asciiString.length(); i++)  
540          {  
541              hexBuff.append("%");  
542              hexBuff.append(Integer.toHexString(ascii[i]));  
543          }  
544          return hexBuff.toString().toUpperCase();  
545      }  
546    
547      /\*\*  
548       \* The main program for the Encoding class  
549       \*   
550       \* @param args  
551       \*            The command line arguments  
552       \*/  
553    
554      public static void main(String[] args)  
555      {  
556          try  
557          {  
558              String userInput = args[0];  
559              String userKey = args[1];  
560              System.out.println("Working with: " + userInput);  
561              System.out.print("Base64 encoding: ");  
562              System.out.println(base64Encode(userInput) + " : " + base64Decode(userInput));  
563              System.out.print("Entity encoding: ");  
564              System.out.println(HtmlEncoder.encode(userInput) + " : " + HtmlEncoder.decode(userInput));  
565              System.out.print("Password based encryption (PBE): ");  
566              System.out.println(encryptString(userInput, userKey) + " : " + decryptString(userInput, userKey));  
567              System.out.print("MD5 hash: ");  
568              System.out.println(hashMD5(userInput) + " : " + "Cannot reverse a hash");  
569              System.out.print("SHA-256 hash: ");  
570              System.out.println(hashSHA(userInput) + " : " + "Cannot reverse a hash");  
571              System.out.print("Unicode encoding: ");  
572              System.out.println("Not Implemented" + " : " + "Not Implemented");  
573              System.out.print("URL encoding: ");  
574              System.out.println(urlEncode(userInput) + " : " + urlDecode(userInput));  
575              System.out.print("Hex encoding: ");  
576              System.out.println(hexEncode(userInput) + " : " + hexDecode(userInput));  
577              System.out.print("Rot13 encoding: ");  
578              System.out.println(rot13(userInput) + " : " + userInput);  
579              System.out.print("XOR with password: ");  
580              System.out.println(xorEncode(userInput, userKey) + " : " + xorDecode(userInput, userKey));  
581              System.out.print("Double unicode encoding is...");  
582              System.out.println("Not Implemented" + " : " + "Not Implemented");  
583              System.out.print("Double URL encoding: ");  
584              System.out.println(urlEncode(urlEncode(userInput)) + " : " + urlDecode(urlDecode(userInput)));  
585          } catch (Exception e)  
586          {  
587              e.printStackTrace();  
588          }  
589      }  
590    
591      /\*\*  
592       \* Description of the Method  
593       \*   
594       \* @param value1  
595       \*            Description of the Parameter  
596       \* @param value2  
597       \*            Description of the Parameter  
598       \* @param description  
599       \*            Description of the Parameter  
600       \* @return Description of the Return Value  
601       \*/  
602    
603      private TR makeRow(String description, String value1, String value2)  
604      {  
605    
606          TD desc = new TD().addElement(description).setBgColor("#bbbbbb");  
607          TD val1 = new TD().addElement(value1).setBgColor("#dddddd");  
608          TD val2 = new TD().addElement(value2).setBgColor("#dddddd");  
609          TR tr = new TR();  
610    
611          tr.addElement(desc);  
612          tr.addElement(val1);  
613          tr.addElement(val2);  
614    
615          return tr;  
616      }  
617    
618      /\*\*  
619       \* Description of the Method  
620       \*   
621       \* @param value1  
622       \*            Description of the Parameter  
623       \* @param value2  
624       \*            Description of the Parameter  
625       \* @param description  
626       \*            Description of the Parameter  
627       \* @return Description of the Return Value  
628       \*/  
629    
630      private TR makeTitleRow(String description, String value1, String value2)  
631      {  
632          TD desc = new TD().addElement(new B().addElement(description));  
633          TD val1 = new TD().addElement(new B().addElement(value1));  
634          TD val2 = new TD().addElement(new B().addElement(value2));  
635          desc.setAlign("center");  
636          val1.setAlign("center");  
637          val2.setAlign("center");  
638          TR tr = new TR();  
639          tr.addElement(desc);  
640          tr.addElement(val1);  
641          tr.addElement(val2);  
642          return (tr);  
643      }  
644    
645      /\*\*  
646       \* Description of the Method  
647       \*   
648       \* @param input  
649       \*            Description of the Parameter  
650       \* @return Description of the Return Value  
651       \*/  
652    
653      public static synchronized String rot13(String input)  
654      {  
655          StringBuffer output = new StringBuffer();  
656          if (input != null)  
657          {  
658              for (int i = 0; i < input.length(); i++)  
659              {  
660                  char inChar = input.charAt(i);  
661                  if ((inChar >= 'A') & (inChar <= 'Z'))  
662                  {  
663                      inChar += 13;  
664                      if (inChar > 'Z')  
665                      {  
666                          inChar -= 26;  
667                      }  
668                  }  
669                  if ((inChar >= 'a') & (inChar <= 'z'))  
670                  {  
671                      inChar += 13;  
672                      if (inChar > 'z')  
673                      {  
674                          inChar -= 26;  
675                      }  
676                  }  
677                  output.append(inChar);  
678              }  
679          }  
680          return output.toString();  
681      }  
682    
683      /\*\*  
684       \* Description of the Method  
685       \*   
686       \* @param str  
687       \*            Description of the Parameter  
688       \* @return Description of the Return Value  
689       \*/  
690    
691      public static String unicodeDecode(String str)  
692      {  
693          // FIXME: TOTALLY EXPERIMENTAL  
694    
695          try  
696          {  
697              ByteBuffer bbuf = ByteBuffer.allocate(str.length());  
698              bbuf.put(str.getBytes());  
699              Charset charset = Charset.forName("ISO-8859-1");  
700              CharsetDecoder decoder = charset.newDecoder();  
701              CharBuffer cbuf = decoder.decode(bbuf);  
702              return (cbuf.toString());  
703          } catch (Exception e)  
704          {  
705              return ("Encoding problem");  
706          }  
707      }  
708    
709      /\*\*  
710       \* Description of the Method  
711       \*   
712       \* @param str  
713       \*            Description of the Parameter  
714       \* @return Description of the Return Value  
715       \*/  
716    
717      public static String unicodeEncode(String str)  
718      {  
719          // FIXME: TOTALLY EXPERIMENTAL  
720          try  
721          {  
722              Charset charset = Charset.forName("ISO-8859-1");  
723              CharsetEncoder encoder = charset.newEncoder();  
724              ByteBuffer bbuf = encoder.encode(CharBuffer.wrap(str));  
725              return (new String(bbuf.array()));  
726          } catch (Exception e)  
727          {  
728              return ("Encoding problem");  
729          }  
730      }  
731    
732      /\*\*  
733       \* Description of the Method  
734       \*   
735       \* @param str  
736       \*            Description of the Parameter  
737       \* @return Description of the Return Value  
738       \*/  
739    
740      public static String urlDecode(String str)  
741      {  
742          try  
743          {  
744              return (URLDecoder.decode(str, "UTF-8"));  
745          } catch (Exception e)  
746          {  
747              return ("Decoding error");  
748          }  
749      }  
750    
751      /\*\*  
752       \* Description of the Method  
753       \*   
754       \* @param str  
755       \*            Description of the Parameter  
756       \* @return Description of the Return Value  
757       \*/  
758    
759      public static String urlEncode(String str)  
760      {  
761          try  
762          {  
763              return (URLEncoder.encode(str, "UTF-8"));  
764          } catch (Exception e)  
765          {  
766              return ("Encoding error");  
767          }  
768      }  
769    
770      /\*\*  
771       \* Description of the Method  
772       \*   
773       \* @param input  
774       \*            Description of the Parameter  
775       \* @param userKey  
776       \*            Description of the Parameter  
777       \* @return Description of the Return Value  
778       \*/  
779    
780      public static synchronized char[] xor(String input, String userKey)  
781      {  
782          if ((userKey == null) || (userKey.trim().length() == 0))  
783          {  
784              userKey = "Goober";  
785          }  
786          char[] xorChars = userKey.toCharArray();  
787          int keyLen = xorChars.length;  
788          char[] inputChars = null;  
789          char[] outputChars = null;  
790          if (input != null)  
791          {  
792              inputChars = input.toCharArray();  
793              outputChars = new char[inputChars.length];  
794              for (int i = 0; i < inputChars.length; i++)  
795              {  
796                  outputChars[i] = (char) (inputChars[i] ^ xorChars[i % keyLen]);  
797              }  
798          }  
799          return outputChars;  
800      }  
801    
802      /\*\*  
803       \* Description of the Method  
804       \*   
805       \* @param input  
806       \*            Description of the Parameter  
807       \* @param userKey  
808       \*            Description of the Parameter  
809       \* @return Description of the Return Value  
810       \*/  
811    
812      public static synchronized String xorDecode(String input, String userKey)  
813      {  
814          try  
815          {  
816              String decoded = base64Decode(input);  
817              return new String(xor(decoded, userKey));  
818          } catch (Exception e)  
819          {  
820              return "String not XOR encoded.";  
821          }  
822      }  
823    
824      /\*\*  
825       \* Description of the Method  
826       \*   
827       \* @param input  
828       \*            Description of the Parameter  
829       \* @param userKey  
830       \*            Description of the Parameter  
831       \* @return Description of the Return Value  
832       \*/  
833    
834      public static synchronized String xorEncode(String input, String userKey)  
835      {  
836          return base64Encode(xor(input, userKey));  
837      }  
838    
839      public Element getCredits()  
840      {  
841          return super.getCustomCredits("", ASPECT\_LOGO);  
842      }  
843  }