Blatt Nr. 2 (Ausgabe: 14. April 2016, Abgabe: 04. Mai 2016)

Kennwortsicherheit

Übungsaufgabe 1. Sicherheit lokaler Rechner

Aufgabe 1.1 Zugriff auf /etc/passwd und /etc/shadow des Webservers

Überblick über die VM.

- *Blatt2-Admin-PC.vmwarevm* wurde aus */home/vmware* nach */home/ss16g07/vmware* kopiert
- wurde über *File -> Open...* importiert (die virtuelle Festplatte wurde eingelesen)
- VM wurde gestartet; beim Boot wurde danach gefragt, ob die VM kopiert oder verschoben wurde; nach Aufgabe wurde "kopiert" ausgewählt

Booten von der CD

- *grml*-iso wurde aus */home/vmware* nach */home/ss16g07/vmware* kopiert
- Neues Image wurde in den VM-Einstellungen in das CD-Laufwerk eingelegt
- Ebenfalls unter den VM-Einstellungen wurde das CD-Laufwerk verbunden
- Während des Bootens der VM wurde das BIOS aufgerufen, um sicherzustellen, dass von der CD gebootet wird

Einlesen und durchsuchen der Root-Partition

- Wählen des deutschen Tastaturlayouts mit $d \rightarrow$ *Enter*
- mounten der Festplatte mit *mount -r /dev/sda1*
- nach */etc/fstab* Festplatte nun */mnt/sda1* zugreifbar
- Auslesen der Dateien */etc/shadow* und */etc/passwd* mit *cat*
  - *passwd* enthält Einträge der folgenden Form: [1]
    - *<Nutzername>*:x[1]:*<Nutzer ID>*:*<Gruppen ID>*:*<Nutzer ID Info>*:*<home-Verzeichnis>*:*<Shell>*
  - *shadow* enthält Einträge der folgenden Form: [2]
    - *<Nutzername>*:*<verschlüsseltes Password>*:*<Tag der letzen Passwortänderung>*[2]:*<minimaler Zeitabstand zwischen Passwortänderungen>*[3]:*<maximaler Zeitabstand zwischen Passwortänderungen>*[4]:*<Warnungszeitraum für auslaufende Passwörter>*:*<Zeit nach der ein Password ausläuft>*[5]:*<Zeit, die seit der Inaktivität des Accounts vergangen ist>*
- es gibt die Benutzer *webadmin* und *georg*
- Herausfinden der Nutzergruppen mit *cat group | grep <Benutzername>*[6]
  - georg : admin georg
  - webadmin : adm dialout cdrom plugdev lpadmin webadmin sambashare

[1] *x* (bei Ubuntu 14: *) indiziert, dass ein verschlüsseltes Passwort für diesen Nutzer in */etc/shadow* vermerkt ist
[2] in Tagen seit dem 1. Jan 1970
[3] Zeit, bis das Passwort wieder geändert werden kann
[4] Zeitpunkt, an dem das Passwort verfällt
[5] nach Inaktivität des Accounts
[6] Durch die | wird die Ausgabe von *cat group* an den *grep*-Befehl weitergegeben, der alles herausfiltert, was nicht zum ihm angegebenen Parameter passt

Aufgabe 1.2 Auslesen von Kennwörter

- salting: Hinzufügen einer zufälligen Zeichenkette ("salt")
- hashing: Umrechnung der Daten in Hash-Werte[1]

Installieren und Verwendung von *john*

- John wurde installiert mit *apt-get install john*, es konnte jedoch nicht authentifiziert werden
- Einfaches Ausführen von *john* zeigt die Hilfe-Seite
- Eingabe des Befehls *john –incremental –users=webadmin /mnt/sda1/etc/shadow*, um das Passwort von *webadmin* im *incremental*-Mode zu ermitteln
- Nach 5 Minuten wurde eine manuelle Terminierung durchgeführt

Wörterbuchangriff

- es wurde in das home Verzeichnis navigiert damit wieder Schreibzugriff besteht
- mit *wget http:download.openwall.net/pub/wordlists/all.gz* wurde ein Wörterbuch runtergeladen
- durch *gunzip all.gz* wurde das Wörterbuch entpackt
- und mit *john –wordlist=all –users=webadmin /mnt/sda1/etc/shadow* der Angriff gestartet
- nach 21.01 sec war das Passwort herausgefunden: mockingbird

Aufgabe 1.3 Setzen von neuen Kennwörtern

- Das Passwort von georg ist nicht ohne weiteres ermittelbar, weil es wahrscheinlich nicht im Wörterbuch steht
- zum unmounten von sda1 wurde *umount /dev/sda1* eingegeben
- zum erneuten mounten wurde *mount -w /dev/sda1* eingegeben
- zum Ändern des root Verzeichnsises mit shell Wechsel wurde *chroot /mnt/sda1/ /bin/sh* eigegeben
- nun wurde das Passwort von georg auf *1* gesetzt: *passwd georg 1*
- es wurde das sytem durch *exit* gefolgt von *shutdown -r now* neu gestartet und sich als georg eingeloggt

Übungsaufgabe 2. Sichere Speicherung von Kennwörtern

Aufgabe 2.1 Angriffe mit Hashdatenbanken und Rainbow-Tables

- es wurde in das home Verzeichnis von webadmin navigiert durch *cd /home/webadmin/*
- Wechseln in das Unterverzeichnis *Rainbowtables/rcracki*
- Ausführung von *rcracki* mit *./rcracki <table-path> -l <password-file>*
- es konnten nicht alle Passwörter ermittelt werden.Vermutlich weil nicht alle Passwörter in der benutzten RainbowTable codiert waren
- eigene Programme sind immer gut, weil man weiß, was sie können, dementprecehnd dauert es aber auch lange, viel Umfang einzbauen
- diese Speicherung würde (da jedes Passwort einen Hash der Länge 128 Bit[4] generiert) $\Sigma_{i=1}^{7} 128^i$ Bit $\approxeq$ 71 Terabyte verbrauchen, während eine der gegebenen Rainbowtables nur ca. 40 MB groß ist

[1] Werte fester Länge, typischerweise hexadezimal codiert [3]

Aufgabe 2.2 Eigener Passwort-Cracker

Quellcode zu dieser Aufgabe zu finden unter **Anhang I: Erinnerungshilfe**

Passwort: s1v3s

Lösungsweg:

- Darstellung der Passwortstellen wie "Walzen"bei einem Zahlenschloss (Wert von a bis 9)
- Senden eines Ticks an die Walzen
- Erste Walze wird gedreht, bei Überlauf Weitergabe an nächste Walze (rekursiv über alle sechs Walzen)
- Startwert jeder Walze ist " "
- Abfragen des Walzenstandes (Char-Sequenz) nach jedem Tick
- abgefragten Stand mit salt konkatenieren und hashen
- Kombinationen reichen von "a" bis "999999"
- Abbruch, wenn gehashte Kombination mit gegebenem Hash übereinstimmt

Aufgabe 2.3 Eigene Kennwort-Speicherfunktion in Java

Quellcode zu dieser Aufgabe zu finden unter **Anhang II: Useradmin-Klasse**

Funktionalität:

- Speicherformat: `username:salt:password-hash`
- Speicherort: Festgelegt auf `~/Documents/passwoerter.txt`
  - Verbesserungsoption: Benutzer den Speicherpfad angeben lassen
- 5000-maliges hashen des Passwortes
- `checkUser` nimmt das Nutzer-Passwort-Tupel entgegen, hasht das Salt-Passwort-Tupel 5000 mal und vergleicht mit dem Nutzer-Hash-Tupel im Speicher
- *main*-Methode nimmt als zusätzliche Parameter Methodennamen und Nutzernamen entgegen
- Eingabeaufforderung für Passwort (vgl. Zusatzfrage, s.u.)
- Vorhandene Datensätze werden überschrieben

Zusatzfrage. Wenn das Passwort als Konsolenparameter übergeben wird, ist es in der Historie einzusehen und kann über `script` gespeichert werden. Aus diesem Grund wurde sich dazu entschieden, das Passwort über ein `JOPtionPane` im Programm abzufragen.

Übungsaufgabe 3. Forensische Wiederherstellung von Kennwörtern

Aufgabe 3.1

- Speicherorte: Swap, Registry, bash-history

Aufgabe 3.2

Es wurde die bash-Chronik ausgelesen mit `/mnt/sda1/home/user # cat .bash_history`.
Inhalt:

- reboot
- cat /etc/inittab
- man intro
- man ls
- exit
- vi /etc/sudoers
- top
- pico
- cd
- exit
- java
- cd Desktop
- jedit wp-config.php
- scp wp-config.php root@10.1.1.2:/var/www/wordpress/
- rm wp-config.php
- exit

Aufgabe 3.3

Es wurden folgende Passwörter gefunden:

- **user:** bloguser
  password: Flugentenfederkiel/991199
- **user:** blog_user
  password: DUzvAu22cKatsXyV
- **user:** bloguser
  password: Kindergeburtstag/119911

Der letzte Eintrag führte zum erfolgreichen Login.

Übungsaufgabe 4. Unsicherer Umgang mit Passwörtern in Java

- Mögliche Sicherheitslücke in Klasse `transport/HTTPTransport.java`
- entschlüsseltes Passwort wird in String gespeichert
- nach [5] sind Strings im Speicher auslesbar
- alternativ kann Prozess gedumped werden, womit später der Dump ausgelesen werden kann

Literatur

[1] http://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/

[2] http://www.cyberciti.biz/faq/understanding-etcshadow-file/

[3] http://zeitstempel.hauke-laging.de/hashinfo.php

[4] http://de.wikipedia.org/wiki/Message-Digest_Algorithm_5

[5] http://www.foo.be/course/dess-20082009/davidoff-clearmem-linux.pdf

Anhang I: Erinnerungshilfe

Startklasse:

```
1  package zwei.zwei;
2
3  public class Startup {
4
5      private static String password;
6      public static void main (String[] args){
7          Erinnerungshilfe eh = new Erinnerungshilfe();
8          password = eh.get_passwort();
9          System.out.println("Passwort: " + password);
10     }
11 }
```

Eigentliche Erinnerungshilfe:

```
1  package zwei.zwei;
2  import java.security.MessageDigest;
3  import java.security.NoSuchAlgorithmException;
4  import java.util.ArrayList;
5
6  public class Erinnerungshilfe {
7      // Halterung für die Walzen
8      private ArrayList<Integer> _walzen;
9      // Speicher für das aktuelle Passwort
10     private String _passwort;
11     // Halterung für alle gültigen Symbole
12     private ArrayList<String> _charListe;
13     // Liste aller gültigen Symbole
14     private String _symbols;
15     // Halterung für Salt
16     private String _salt;
17     // Walze, die zur Zeit die letzte ist, welche bearbeitet wird.
18     private int _aktuelleWalze;
19
20     /**
21      * Konstruktor. Ohne Angabe eines Salts wird der Salt aus der Aufgabe
22      * verwendet
23      */
24     public Erinnerungshilfe() {
25         this("8kofferradio");
26     }
27
28     /**
29      * Konstruktor. Probiert automatisch alle Passwörter durch
30      */
31     public Erinnerungshilfe(String salt) {
32         _walzen = new ArrayList<Integer>();
33         setupWalzenListe();
34         _passwort = "";
35         _symbols = "abcdefghijklmnopqrstuvwxyz0123456789";
36         _charListe = new ArrayList<String>();
37         setupCharListe();
38         _salt = salt;
39         findePasswort();
40     } // end Konstruktor
41
42     /**
```

```
43      * Iteriert über die Walzen, bis die Abbruchbedingung erfüllt ist oder
44      * alle Werte ausprobiert wurden
45      */
46     private void findePasswort() {
47         _aktuelleWalze = 0;
48         while (_aktuelleWalze != _walzen.size()) {
49             tick(_walzen.get(_aktuelleWalze));
50             String hashPassword = getMD5Hash(_salt.concat(_passwort));
51             System.out.println(_passwort);
52 
53             if (hashPassword.equals("2b2935865b8a6749b0fd31697b467bd7")) {
54                 break;
55             }
56             _passwort = "";
57         }
58     } // end findePasswort()
59 
60     /**
61      * Dreht die gegebene Walze um ein Feld weiter. Bei Überlauf wird die
            Walze
62      * zurückgesetzt und die nächste Walze rekursiv aufgerufen. Anschließ
            end
63      * wird das zum aktuellen Walzenstand gehörende Passwort generiert.
64      *
65      * @param walze
66      *            der Stand der aktuellen Walze
67      * @param walzenIndex
68      *            der Index der aktuellen Walze (wichtig zum Ändern des
            Wertes
69      *            im Walzenarray)
70      */
71     private void tick(int walze) {
72         walze += 1;
73         _walzen.set(_aktuelleWalze, walze); // setzt den Wert der Walze
               auch im
74         // Array
75         if (walze == _symbols.length()) {
76             _aktuelleWalze = rolleWalze(_aktuelleWalze); // setzt die nä
                   chste
77             // Walze weiter
78             // walze = _walzen.get(walzenIndex);
79             if (_aktuelleWalze < _walzen.size()) {
80                 walze = 0; // setzt die aktuelle Walze zurück
81                 _walzen.set(_aktuelleWalze, walze); // setzt den Wert der
                       Walze
82                 // auch im Array
83             }
84         }
85         if (_aktuelleWalze < _walzen.size()) {
86             for (int w : _walzen) {
87                 if (w >= 0) {
88                     _passwort += getSymbol(w);
89                 }
90             }
91         }
92     } // end tick(int)
93 
94     /**
95      * Wird aufgerufen, sobald eine Walte komplett durchgedreht hat. Setzt
```

```
         die
96       * nächste Walze einen Index weiter. Methode: Walze 1 dreht durch.
             Walze 1
97       * wird wieder auf Wert 1 gesetzt. Walze 2 dreht durch. Walze 2 wird
             wieder
98       * auf Wert 1 gesetzt und Walze 1 wird einen Wert weiter gesetzt. Walte
              2
99       * dreht durch. usw.
100      *
101      * @param walzenIndex
102      *            der Index der aktuell durch gedrehten Walze
103      * @return der Index der neuen aktuellen Walze
104      */
105     private int rolleWalze(int walzenIndex) {
106         if (walzenIndex == 0) {
107             if (_walzen.get(walzenIndex) >= 35) {
108                 _walzen.set(walzenIndex, 0);
109                 return walzenIndex + 1; // gehe zur nächsten Walze
110             } else {
111                 _walzen.set(walzenIndex, _walzen.get(walzenIndex) + 1);
112                 return walzenIndex;
113             }
114         } else if (0 < walzenIndex && walzenIndex < _walzen.size()) {
115             if (_walzen.get(walzenIndex) >= 35) {
116                 _walzen.set(walzenIndex, 0);
117                 return rolleWalze(walzenIndex - 1) + 1; // setze letzte
                        Walze +1
118             } else {
119                 _walzen.set(walzenIndex, _walzen.get(walzenIndex) + 1);
120                 return walzenIndex;
121             }
122         }
123         return -1; // Hier kommt das Programm nie an
124     }
125
126     /**
127      * Holt ein Symbol aus der Symbolliste
128      *
129      * @param index
130      *            der Index des Symbols
131      * @return das Symbol
132      */
133     private String getSymbol(int index) {
134         return "" + _symbols.charAt(index);
135     } // end getSymbol()
136
137     /**
138      * setzt die gültigen Symbole in einer Liste auf
139      */
140     private void setupCharListe() {
141         for (int i = 0; i < _symbols.length(); i++) {
142             _charListe.add(i, _symbols.substring(i, i + 1));
143         }
144     } // end setupCharListe()
145
146     /**
147      * setzt die Walzen auf und sortiert sie in einer Liste
148      */
149     private void setupWalzenListe() {
```

```
150         int walze0 = -1;
151         int walze1 = -1;
152         int walze2 = -1;
153         int walze3 = -1;
154         int walze4 = -1;
155         int walze5 = -1;
156         _walzen.add(0, walze0);
157         _walzen.add(1, walze1);
158         _walzen.add(2, walze2);
159         _walzen.add(3, walze3);
160         _walzen.add(4, walze4);
161         _walzen.add(5, walze5);
162     } // end setupWalzenListe()
163 
164     /**
165      * Eine Methode zum einhashen von Srings im md5 Format
166      *
167      * @param yourString
168      *            Der String der gehasht werden soll
169      * @return Der gehashte String
170      */
171     private String getMD5Hash(String yourString) {
172         MessageDigest md = null;
173         try {
174             md = MessageDigest.getInstance("MD5");
175         } catch (NoSuchAlgorithmException e) {
176             e.printStackTrace();
177         }
178         md.update(yourString.getBytes());
179         byte[] digest = md.digest();
180         StringBuffer sb = new StringBuffer();
181         for (byte b : digest) {
182             sb.append(String.format("%02x", b & 0xff));
183         }
184         return sb.toString();
185     } // end getMD5Hash(String)
186 
187     /**
188      * gettermethode für _passwort
189      *
190      * @return
191      *          _passwort
192      */
193     public String get_passwort() {
194         return _passwort;
195     } // end get_passwort()
196 
197 } // end class
```

Anhang II: Useradmin-Klasse

Interface `Useradmin`

```
1  package zwei.drei;
2
3  /**
4   * Kennwortverwalter
5   *
6   * @author georg
7   *
8   */
9  public interface Useradministration {
10
11     /**
12      * füge Nutzer hinzu
13      *
14      * @param username
15      *              der Nutzer
16      * @param password
17      *              sein Passwort
18      */
19     public void addUser(String username, char[] password);
20
21     /**
22      * prüft, ob das Passwort gültig ist
23      *
24      * @param username
25      *              der Nutzer
26      * @param password
27      *              sein Passwort
28      * @return
29      *              ist das Passwort gültig?
30      */
31     public boolean checkUser(String username, char[] password);
32 }
```

Implementierende Klasse:

```
1  package zwei.drei;
2
3  import java.security.MessageDigest;
4  import java.security.NoSuchAlgorithmException;
5  import java.util.ArrayList;
6
7  public class Useradmin implements Useradministration {
8
9      private RandomSalt _rs;
10     private DateiVerwalter _ds;
11
12     public static void main(String[] args) {
13         if (args.length == 3) {
14             Useradmin ua = new Useradmin();
15             String name = args[1];
16             char[] passwort = args[2].toCharArray();
17             if (args[0].equals("addUser"))
18                 ua.addUser(name, passwort);
19             else if (args[0].equals("checkUser"))
20                 System.out.println(ua.checkUser(name, passwort));
21             else
```

```
22                  System.out.println("Nein");
23
24          } else
25              System.out.println("Nein");
26      }
27
28      public Useradmin() {
29          _rs = new RandomSalt();
30          _ds = new DateiVerwalter();
31      }
32
33      @Override
34      public void addUser(String username, char[] password) {
35          String salt = _rs.getRandomSalt();
36          String hash = hash1k(salt.concat(String.copyValueOf(password)));
37          _ds.fuegeInhaltHinzu(username + ":" + salt + ":" +  hash);
38          _ds.write();
39      }
40
41      private String hash1k(String hash) {
42          for (int i = 0; i < 1001; i++) {
43              hash = getSHA512Hash(hash);
44          }
45          return hash;
46      }
47
48      @Override
49      public boolean checkUser(String username, char[] password) {
50          ArrayList<String> list = _ds.read();
51          String[] laufstring;
52          String nutzer;
53          String salt;
54          String hash;
55          for (String s : list) {
56              laufstring = s.split(":");
57              nutzer = laufstring[0];
58              salt = laufstring[1];
59              hash = laufstring[2];
60              if (nutzer.equals(username) && hash.equals(hash1k(salt
61                      .concat(String.copyValueOf(password)))))
62                      return true;
63
64          }
65          return false;
66      }
67
68      /**
69       * Eine Methode zum einhashen von Srings im SHA512 Format
70       *
71       * @param yourString
72       *            Der String der gehasht werden soll
73       * @return Der gehashte String
74       */
75      private String getSHA512Hash(String yourString) {
76          MessageDigest md = null;
77          try {
78              md = MessageDigest.getInstance("SHA-512");
79          } catch (NoSuchAlgorithmException e) {
80              e.printStackTrace();
```

```
81          }
82          md.update(yourString.getBytes());
83          byte[] digest = md.digest();
84          StringBuffer sb = new StringBuffer();
85          for (byte b : digest) {
86              sb.append(Integer.toString((b & 0xff) + 0x100, 16).substring(1)
                    );
87          }
88          return sb.toString();
89      } // end getMD5Hash(String)
90  }
```

Datentyp “DateiVerwalter”:

```
1   package zwei.drei;
2
3   import java.util.ArrayList;
4   import java.util.Random;
5
6   /**
7    * generiert random Salts
8    *
9    * @author georg
10   *
11   */
12  public class RandomSalt {
13
14      // Halterung für die Walzen
15      private ArrayList<Integer> _walzen;
16      // Halterung für alle gültigen Symbole
17      private ArrayList<String> _charListe;
18      // Liste aller gültigen Symbole
19      private String _symbols;
20      // Random Event
21      private Random _rn;
22
23      /**
24       * Konstruktor
25       */
26      public RandomSalt() {
27          _rn = new Random();
28          _walzen = new ArrayList<Integer>();
29          _symbols = "abcdefghijklmnopqrstuvwxyz0123456789";
30          _charListe = new ArrayList<String>();
31          setupCharListe();
32      }
33
34      /**
35       * erstellt einen random String
36       *
37       * @return
38       *              ein random String
39       */
40      public String getRandomSalt() {
41          setupWalzenListe();
42
43          String salt = "";
44          for (int w : _walzen) {
45              if (w >= 0) {
46                  salt += getSymbol(w);
```

```
47              }
48          }
49          _walzen.clear();
50          return salt;
51      }
52
53      /**
54       * Holt ein Symbol aus der Symbolliste
55       *
56       * @param index
57       *            der Index des Symbols
58       * @return das Symbol
59       */
60      private String getSymbol(int index) {
61          return "" + _symbols.charAt(index);
62      } // end getSymbol()
63
64      /**
65       * setzt die gültigen Symbole in einer Liste auf
66       */
67      private void setupCharListe() {
68          for (int i = 0; i < _symbols.length(); i++) {
69              _charListe.add(i, _symbols.substring(i, i + 1));
70          }
71      } // end setupCharListe()
72
73      /**
74       * setzt die Walzen auf und speichert sie in einer Liste,
75       * zusammen mit ihrem Ausgabewert
76       */
77      private void setupWalzenListe() {
78          int anzahl = Zufall(0, 20);
79
80          for (int i = 0; i < anzahl; i++) {
81              _walzen.add(i, Zufall(0, 35));
82          }
83      } // end setupWalzenListe()
84
85      /**
86       * generiert eine Zufallszahl zwischen min und max
87       *
88       * @param min
89       *              Untergrenze
90       * @param max
91       *              Obergrenze
92       * @return
93       *              der Zufallswert
94       */
95      private int Zufall(int min, int max) {
96          return _rn.nextInt(max - min + 1) + min;
97      }
98  }
```

Datentyp “RandomSalt”:

```
1  package zwei.drei;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileWriter;
```

```
6  import java.io.IOException;
7  import java.util.ArrayList;
8  import java.util.Scanner;
9
10 public class DateiVerwalter {
11
12     private ArrayList<String> _passwoerter;
13     private String _filePath;
14
15     public DateiVerwalter() {
16         _passwoerter = new ArrayList<String>();
17         String OS = System.getProperty("os.name");
18         _filePath = System.getProperty("user.home");
19         // Anpassen des Dateipfads entsprechend dem Host-OS
20         if (OS.startsWith("Windows")) {
21             _filePath += "\\Documents\\passwörter.txt";
22         } else if (OS.startsWith("Linux")) {
23             _filePath += "/Documents/passwörter.txt";
24         }
25         _passwoerter = read();
26     }
27
28     /**
29      * fügt einen String an die Ausgabe an
30      *
31      * @param inhalt
32      *             der String
33      *
34      */
35     public void fuegeInhaltHinzu(String inhalt) {
36         String[] laufstring;
37         String[] laufstringInhalt;
38         ArrayList<String> passwoerterBuffer = new ArrayList<String>(
               _passwoerter);
39         for (String s : _passwoerter) {
40             laufstring = s.split(":");
41             laufstringInhalt = inhalt.split(":");
42             if (laufstring[0].equals(laufstringInhalt[0]))
43                 passwoerterBuffer.remove(s);
44         }
45         _passwoerter = passwoerterBuffer;
46         _passwoerter.add(inhalt);
47     } // end fuegeInhaltHinzu(String)
48
49     /**
50      * fügt ein StringAray an die Ausgabe an
51      *
52      * @param inhalt
53      *             der String
54      *
55      */
56     public void fuegeInhaltHinzu(ArrayList<String> inhalt) {
57         _passwoerter.addAll(inhalt);
58     } // end fuegeInhaltHinzu(ArrayList<String>)
59
60     /**
61      * Schreibt die Passwortliste in eine Datei 'passwörter.txt' im
62      * Dokumente-Ordner im Home-Verzeichnis des Nutzers
63      */
```

```
64      public void write() {
65          File file = new File(_filePath);
66          try {
67              if (!file.exists()) {
68                  file.createNewFile();
69              }
70              FileWriter fw = new FileWriter(file);
71              BufferedWriter bw = new BufferedWriter(fw);
72              for (int i = 0; i < _passwoerter.size(); i++) {
73                  bw.write(_passwoerter.get(i));
74              }
75              bw.flush();
76              bw.close();
77          } catch (Exception e) {
78              System.out.println("There was a problem writing your file");
79          }
80      } // end write()
81
82      /**
83       * Liest ein Dokument aus
84       */
85      public ArrayList<String> read() {
86          ArrayList<String> list = new ArrayList<String>();
87          try {
88              Scanner s = new Scanner(new File(_filePath));
89              while (s.hasNextLine()) {
90                  list.add(s.nextLine());
91              }
92              s.close();
93          } catch (IOException e) {
94              e.printStackTrace();
95          }
96          return list;
97      } // end read()
98  }
```