Blatt 2 - Kennwortsicherheit

Kolja Hopfmann, Jonas Sander

3. Mai 2018

---

1 Sicherheit lokaler Rechner

1.1

1.1.1

Die VM wurde mit der grml-CD gebootet und der Festplatteninhalt der eigentlichen VM wurde gemountet.

```
lsblk
mount -r /dev/sda1
```

Der Aufbau von */etc/passwd/* ist:
Name:Passwort:UserID:GroupID:Kommentar:Verzeichnis:Shell , wobei hier das Passwort in der */etc/shadow/* Datei als Hash ausgelagert ist.
Es existieren die User *Georg* und *webadmin*. Der User *Georg* hat Sudo-Berechtigung

1.2

1.2.1

Eine kryptographische Hashfunktion ist eine Funktion welche einen beliebigen Eingabeparameter so verändert, dass es praktisch unmöglich ist, mittels Berechnung auf den ursprünglichen Eingabewert zu schließen. Ein Salt ist ein

randomisierter Eingabewert, welcher mit dem Passwort konkateniert wird und gemeinsam gehasht wird. Dadurch ist der Hash eines bestimmten Klartextes mit Salt nicht immer der selbe. Die bedeutet das man das Ursprüngliche Passwort nur durch mehrfaches Ausprobieren herausfinden kann.

1.2.2

John wurde installiert und die Manual wurde gelesen:

```
sudo apt install john
man john
```

John wurde im Incremental-Mode gestartet:

```
john -incremental /etc/shadow
```

Der Versuch war nicht erfolgreich. Grund: Incremental Mode iteriert von Anfang über alle möglichen Passwort-Strings, für einen Zeitraum von 15 Minuten war das Passwort zu lang um es über Incremental zu ermitteln.

1.2.3

Es wurde eine Wordlist runtergeladen,entpackt und anschließend für einen weiteren Versuch mit John The Ripper benutzt.

```
wget http://download.openwall.net/pub/wordlists/all.gz
gunzip all.gz
john --wordlist=all /etc/shadow
```

Passwort für *webadmin*: *mockingbird*

1.3

Das Passwort für den User *georg* war nicht in der Wordlist enthalten.

Das VM-Image wurde erneut gemountet, diesmal mit Schreibzugriff. Daraufhin wurde mit *chroot* eine neue Bash-Session gestartet mit dem VM-Image root als root. Somit konnte man mit Rootrechten das Passwort von *georg* ändern.

```
mount /dev/sda1
chroot /media/sda1 /bin/bash
passwd georg
exit
```

2 Sichere Speicherung von Kennwörtern

2.1

1. Zuerst wurde in das Verzeichnis von rcracki navigiert.

```
cd webadmin/Rainbowtables/rcracki_mt_0.7.0_Linux:x86_64
```

Hier wurde dann rcracki auf der Datei mit den Passwörtern ausgeführt.

```
./cracki -l [password txt] [path to rainbow table]
```

Hierbei wurden Passwort Nummer 4: ulardi und Passwort Nummer 5: avanti gefunden. 2. Die Restlichen Passwörter konnten mit der Verwendeten Rainbowtable nicht geknackt werden. Dies ist darauf zurück zu führen, dass die Passwörter nicht als Wörter in der Tabelle enthalten sind. Ein erneuter Versuch mit einer anderen Rainbowtable könnte weitere Passwörter knacken, jedoch ist dies keineswegs sicher.
Für das Abspeichern der MD5-Hashes aller alphanumerischen Passwörtern der Länge 1-7 wäre ein Speicher von $\sum_1^7(62^i \cdot 32byte) = 1,1 \cdot 10^{14}Byte = 110TB$ nötig. Dies ist um den Faktor 10^4 größer als die verwendete Rainbowtable und deutlich zu groß zur Speicherung.

2.2

Unsere Lösung war auf dem Alphabet bestehend aus 0-9 und a-z iterativ alle möglichen Strings durch zu probieren. Hierfür haben wird eine Java Klasse geschrieben, die in einer Schleife von 0 bis $36^7 - 1$ iteriert. In Jeder Iteration wird ein String zusammengesetzt, dieser mit dem Salt gehasht und das Ergebnis des Hashes mit dem in der Passwort-Datei gefundenem

Hash verglichen. Hierbei werden erst alle 1-Stelligen, dann alle 2-Stelligen Passwörter durchlaufen etc. Hiermit wurde das Passwort s1v3s gefunden. Die Datei bruteforce.java befindet sich im Anhang.

2.3

Die erstellte Java Datei Useradmin.java befindet sich im Anhang.

3 Forensische Wiederherstellung von Kennwörtern

3.1

Aus den empfohlenen Quellen zu Blatt 2 unter "Zum sicheren Umgang mit Passwörtern im Speicher" geht hervor, dass viele Anwendungen Passwörter als Klartext im Arbeitsspeicher ablegen. Dadurch ist es Möglich das Passwort (für kurze Zeit) selbst nach ausschalten des Systems aus dem Speicher zu extrahieren. Desweiteren Dient eine Swap-Partition als zusätzlicher Arbeitsspeicher. So ist es Möglich aus der Swap-Partition Passwörter zu extrahieren.

3.2

Unter grml wurde der Inhalt der VM gemountet:

```
1 mount /dev/sda1
```

Unter */home/user/.bash_history* befinden sich die letzten commands welcher der user in der letzten bash-session in die shell eingetippt hat.

3.3

Da der Admin zunächst mit Jedit die Server-File editiert hatte haben wir zunächst in den Jedit config-Files gesucht, unter *home/.jedit*. Da war zu erkennen dass das alte Passwort "Flugentenfederkiel/991199" war, mit dem User "bloguser".

Daraufhin wurde versucht mit dem Programm photorec, Teile der Swap-Parition wiederherzustellen:

```
lsblk
photorec /dev/sda5
```

Die wiederhergestellten Dateien wurden aufgrund Platzmangel der grml-CD unter */dev/sda*1 abgelegt. Nun wurde versucht die Dateien nach dem String "PASSWORD" zu untersuchen da dies in der Server-Datei vor dem tatsächlichen Passwort stand.

```
find . -name "*.elf" | xargs grep -E 'PASSWORD'
find . -name "*.txt" | xargs grep -E 'PASSWORD'
find . -name "*.java" | xargs grep -E 'PASSWORD'
grep -a 'PASSWORD' f0402684.elf
grep -a 'PASSWORD' f0853144.elf
grep -a 'PASSWORD' f0932456.elf
grep -a 'PASSWORD' f0052256.elf
```

In der Datei *f*0853144.*elf* befand sich das Passwort als Klartext: "Kindergeburtstag/119911"

4 Unsicherer Umgang mit Passwörtern in Java

Der Passwortmanager verwendet sowohl für Encryption als auch Decryption die selbe Funktion. Dieser wird entweder das Klartextpasswort oder das Verschlüsselte Passwort übergeben. Kommt man in den Besitz der Verschlüsselten Passwörter, kann man sie also einfach über den Passwortmanager entschlüsseln.

5 Quellen

- 0. Empfohlene Quellen
- `https://unix.stackexchange.com/`
- man command der Kommandozeile

6 Anhang

6.1

Code des Bruteforcers

```
import javax.xml.bind.DatatypeConverter;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class bruteforce
{

  public static void main(String[] args)
  {
    final String hash = "2b2935865b8a6749b0fd31697b467bd7";
    final String salt = "8kofferradio";
    final String account = "testaccount";
    final int cardinality = 6;
    String result = "";

    final char[] alphabet = {'0', '1', '2', '3', '4', '5', '6
        ', '7', '8', '9',
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'
            , 'l',
        'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w'
            , 'x', 'y', 'z'};


    for (long i = 0L; i < Math.pow(alphabet.length,
        cardinality + 1); i++)
    {
      //first arity
      result = alphabet[(int) (i % 36)] + ""; //36^0
      if (i >= Math.pow(36, 1))
      {
        //second arity
        result = alphabet[(int) ((i / 36) - 1) % 36] + result
            ; //36^1
        if (i >= Math.pow(36, 2))
        {
          //third arity
          result = alphabet[(int) ((i / Math.pow(36, 2) - 1)
              % 36)] + result;
          if (i >= Math.pow(36, 3))
```

```
34            {
35              //fourth arity
36              result = alphabet[(int) ((i / Math.pow(36, 3)) -
                    1) % 36] + result;
37              if (i >= Math.pow(36, 4))
38              {
39                //fifth arity
40                result = alphabet[(int) ((i / Math.pow(36, 4))
                      - 1) % 36] + result;
41                if (i >= Math.pow(36, 5))
42                {
43                  //sixth arity
44                  result = alphabet[(int) ((i / Math.pow(36, 5)
                        ) - 1) % 36] + result;
45                }
46              }
47            }
48          }
49        }
50
51        String saltAndPasswd = salt + result;
52        String hashedPasswd = "";
53        try
54        {
55          hashedPasswd = md5Hash(saltAndPasswd);
56        }
57        catch (NoSuchAlgorithmException ex)
58        {
59          System.out.println("Error when hashing: ");
60          ex.printStackTrace();
61        }
62
63
64        if (hash.equals(hashedPasswd))
65        {
66
67          System.out.println("Password found! :");
68          System.out.println("User: " + account + " Password: "
                + result);
69          break;
70        }
71        else
72        {
73          if(i%100000 == 0)
74          {
```

```
75            System.out.println(i + ". at " + result + "-- Retry
                 ");
76          }
77        }
78      }
79      System.out.println("
           ################################################ --- *
           hackervoice* I'M IN ---
           ################################################");
80    }
81
82    private static String md5Hash(String plain) throws
         NoSuchAlgorithmException
83    {
84      MessageDigest md5 = MessageDigest.getInstance("MD5");
85      md5.update(plain.getBytes());
86      byte[] byteHash = md5.digest();
87      return DatatypeConverter.printHexBinary(byteHash).
           toLowerCase();
88    }
89  }
```

6.2

Code der Webadmin Klasse

```
1  package com.hoppix;
2
3  import javax.xml.bind.DatatypeConverter;
4  import java.io.*;
5  import java.security.MessageDigest;
6  import java.security.NoSuchAlgorithmException;
7  import java.util.Random;
8  import java.util.Scanner;
9
10 public class Useradmin implements Useradministration
11 {
12
13   private static final String fileName = "passwords.txt";
14   private static final int hashIterator = 1337;
15
16   public static void main(String[] args)
17   {
```

```
18      String command = args[0];
19      String user = args[1];
20
21      Useradmin admin = new Useradmin();
22
23      System.out.println("Password: ");
24      Scanner scanner = new Scanner(System.in);
25      String password = scanner.next();
26      scanner.close();
27
28
29
30      switch (command)
31      {
32        case "addUser":
33          admin.addUser(user, password.toCharArray());
34          break;
35
36        case "checkUser":
37          boolean found = admin.checkUser(user, password.
                toCharArray());
38          if (found)
39          {
40            System.out.println("User correct");
41          }
42          break;
43
44        default:
45          System.out.println("Wrong input");
46
47      }
48
49    }
50
51    public void addUser(String username, char[] password)
52    {
53      String passwordString = String.copyValueOf(password);
54      String hash = "";
55
56      Random rand = new Random();
57      int salt = rand.nextInt(1234567) + 100000;
58
59      try
60      {
61        hash = md5Hash(salt + passwordString);
```

```
62        for (int i = 0; i < hashIterator; i++)
63        {
64          hash = md5Hash(hash);
65        }
66
67      }
68      catch (NoSuchAlgorithmException e)
69      {
70        e.printStackTrace();
71      }
72      finally
73      {
74        writeLine(fileName, username, salt, hash);
75      }
76
77
78    }
79
80    public boolean checkUser(String username, char[] password)
81    {
82      String[] content = checkPasswordLine(fileName, username);
83      if (content.equals(null))
84      {
85        System.out.println("User not found!");
86        return false;
87      }
88
89      String passwordString = String.copyValueOf(password);
90      String salt = content[1];
91      String hashSave = content[2];
92
93      try
94      {
95        String hash = md5Hash(salt + passwordString);
96        for (int i = 0; i < hashIterator; i++)
97        {
98          hash = md5Hash(hash);
99        }
100
101       return hash.equals(hashSave);
102     }
103     catch (NoSuchAlgorithmException e)
104     {
105       System.out.println("Wrong password!");
106       e.printStackTrace();
```

```
107         return false;
108       }
109
110     }
111
112     private static String md5Hash(String plain) throws
            NoSuchAlgorithmException
113     {
114       MessageDigest md5 = MessageDigest.getInstance("MD5");
115       md5.update(plain.getBytes());
116
117       byte[] byteHash = md5.digest();
118       return DatatypeConverter.printHexBinary(byteHash).
              toLowerCase();
119     }
120
121     public void writeLine(String fileName, String username, int
            salt, String saltHashedPassword)
122     {
123       try
124       {
125         FileWriter fileWriter = new FileWriter(fileName);
126         BufferedWriter bufferedWriter = new BufferedWriter(
              fileWriter);
127         FileReader fileReader = new FileReader(fileName);
128         BufferedReader bufferedReader = new BufferedReader(
              fileReader);
129
130         if (bufferedReader.readLine() == null)
131         {
132           bufferedWriter.write(username + " " + salt + " " +
                saltHashedPassword);
133           bufferedWriter.newLine();
134         }
135
136         bufferedWriter.close();
137         bufferedReader.close();
138       }
139       catch (IOException e)
140       {
141         e.printStackTrace();
142         System.out.println("Schreibfehler");
143       }
144     }
145
```

```
146   public String[] checkPasswordLine(String fileName, String
          username)
147   {
148     String line = null;
149
150     try
151     {
152       FileReader fileReader = new FileReader(fileName);
153       BufferedReader bufferedReader = new BufferedReader(
              fileReader);
154
155       while ((line = bufferedReader.readLine()) != null)
156       {
157         String[] content = line.split(" ");
158         if (username.equals(content[0]))
159         {
160           return content;
161         }
162       }
163       bufferedReader.close();
164       System.out.println("Read " + fileName);
165     }
166     catch (FileNotFoundException e)
167     {
168       e.printStackTrace();
169       System.out.println("Datei nicht gefunden");
170     }
171     catch (IOException e)
172     {
173       e.printStackTrace();
174       System.out.println("Lesefehler");
175     }
176     return null;
177   }
178
179 }
180
181 interface Useradministration
182 {
183   void addUser(String username, char[] password);
184
185   boolean checkUser(String username, char[] password);
186 }
```