Labreport 02

Tronje Krabbe, Julian Deinert

4. Mai 2016

Inhaltsverzeichnis

Aufgabe 1

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

- Wir haben den Ordner mit dem **File Browser** in unseren vmware-Ordner kopiert.
- Wir änderen die Einstellungen unserer VM so, dass das grml-Abbild im virtuellen CD/DVD-Laufwerk liegt und setzen den Haken bei *connect on power on*. Letzteres sorgt dafür, dass das image geladen wird, bevor Ubuntu gestartet wird.
- Nach dem Starten von *grml* mounten wir die *Root-Partition* nach `/mnt`. Mit `lsblk` ergibt sich aus dem output

  ```
  NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
  fd0      2:0    1     4K  0 disk
  sda      8:0    0    20G  0 disk
  |-sda1   8:1    0  19.1G  0 part
  |-sda2   8:2    0     1K  0 part
  |-sda5   8:3    0   895M  0 part
  ```

  dass die *Root-Partition* auf `sda1` liegt. Wir mounten diese also mit `sudo mount /dev/sda1 /mnt` in den Ordner `/mnt`.
- Die Datei *passwd* enthält für jeden User eine Zeile, die aus folgenden Feldern besteht:
  - Dem Login-Namen
  - Dem verschlüsseltem Password des Users (optional)
  - Der User-ID
  - Der Gruppen-ID
  - Dem User-Namen oder einem Kommentar
  - Dem *home directory* des Users
  - Der *shell* des Users (optional)
- Die Datei *shadow* speichert die gehashten Passwörter der User im folgenden Format:
  <login name>:<encrypted password>:<date of last password change>:<minimum password age>:<maximum password age>:<password warning period>:<password inactivity period>:<account expiration date>:<reserved field>

  In beiden Datien gibt es neben diversen Usern für Services und Programme die User `root`, `georg` und `webadmin`.
- Der User `georg` gehört der Gruppe `admin` an. `webadmin` gehört den Gruppen `adm, dialout, cdrom, plugdev, lpadmin`
  sowie `sambashare` an.

1.2 Auslesen von Kennwörtern

- Ein gesaltetes, gehashtes Passwort ist ein Passwort, das erst mit einem sogenannten Salt kombiniert und danach mit einer Hash-Funktion gehasht wurde. Eine Hash-Funktion verändert ihren Input deterministisch so, dass das Ergebnis nicht (ohne Weiteres) zurück zum Input transformiert werden kann. Der Salt kann eine beliebige Zeichenkombination sein, dessen Nutzung die Anwendung von Rainbow-Tables verhindern soll. Insbesondere erschwert der Hash das Reversieren des gehashten Wertes zum Original.
- Die `--incremental` Option führt einen Brute-Force-Angriff auf das Passwort auf, was sehr, sehr langsam ist aufgrund der vielen möglichen Kombinationen von Zeichen.
- Der Wörterbuch-Angriff funktioniert sehr viel schneller, und das Passwort lautet *mockingbird*.

1.3 Setzen von neuen Kennwörtern

Das Passwort von *georg* konnte nicht mit `john` ermittelt werden, da es nicht in unserem (und daher wahrscheinlich auch in keinem anderen) Wörterbuch vorkommt.

Aufgabe 2

2.1 Angriffe mit Hashdatenbanken und Rainbow-Tables

Die ermittelten Kennwörter sind:

- ente
- ball
- borkeni
- ulardi
- avanti

Zur Idee von Black Hat: es gibt 62 alphanumerische Zeichen (26 Klein- und 26 Großbuchstaben, sowie 10 Ziffern). Das ergibt

$$\sum_{1}^{n=7} 62^i = 3.579.345.993.194$$

mögliche Hashes, jeder mit einer Größe von 32 Byte, bzw. 33 Byte inklusive des Separators, was insgesamt $118,1$ TB belegt. Wer hier pedantisch sein möchte, könnte bemerken, dass wir ein Byte zu viel eingerechnet haben, da nach dem letzten Eintrag kein Separator benötigt wird. Trotzdem ist dies ein exorbitanter Speicherplatzbedarf, der uns auch nur weiterhilft, wenn das Passwort tatsächlich nur aus alphanumerischen Zeichen besteht. Die Rainbow-Tables dagegen belegen lediglich $12,2$ GB.

2.2 Eigener Passwort-Cracker

Wir haben unseren Passwort-Cracker in Python geschrieben; er benutzt `hashlib`, für die *md5*-Funktion, und `itertools`, um alle Permutationen des Alphabets bis zur Länge 6 zu finden. Das Ergebnis ist: s1v3s. Der Source-Code befindet sich im Appendix.

2.3 Eigene Kennwort-Speicherfunktion in Java

Wir haben die Speicherfunktion mit der Hilfe von JBCrypt implementiert. Der Username wird mit dem gesalteten und gehashten Password, durch eine Pipe getrennt, in die Datei `passwords.txt` gespeichert.

Aufgabe 3

Im Home-Verzeichnis des Users */home/user* haben wir in der `.bash_history` entdeckt, dass offenbar eine Konfigurationsdatei für Worpress mit *jedit* editiert wurde. *jedit* speichert diverse hilfreiche Informationen im Verzeichnis *.jedit*. Sehen wir uns die Datei `history` in diesem Verzeichnis an, erkennen wir, dass eine Datenbank-Konfiguration vom clipboard eingefügt wurde. Dort steht ein User `bloguser` und ein Passwort drin, welches `Flugentenfederkiel/991199` lautet, und nicht funktioniert.
Glücklicherweise gibt es zwei User auf dem System, der andere ist natürlich *root*. Auch dieser besitzt ein *.jedit* Verzeichnis, welches eine `history` enthält. In dieser sind zwei Einträge, die vom clipboard eingefügt wurden, zu finden. Sie enthalten zwei User/Passwort Paare, einmal `bloguser` und `Flugentenfederkiel&10`, sowie `blog_user` und `DUzvAu22cKatsXyV`. Beide dieser Kombinationen funktionieren ebenfalls nicht. Die beiden *.jedit* Verzeichnisse enthalten ebenfalls jeweils eine `killring.xml`, wo gelöschte Zeichenketten gespeichert wurden. In der von *root* stehen interessante Kandidaten für Passwörter, aber von denen funktionieren ebenfalls keine.

Appendix

Eigener Passwort-Cracker

pw-crack.py

```
#!/usr/bin/env python

import sys
import string
import hashlib
import itertools

target = b'2b2935865b8a6749b0fd31697b467bd7'
salt = b'8kofferradio'

symbols = string.ascii_lowercase + '0123456789'

for i in range(1, 7):
    for elem in itertools.product(symbols, repeat=i):
        elem_bytes = "".join(elem).encode()
        m = hashlib.md5()
        m.update(salt + elem_bytes)
        if m.hexdigest() == target.decode():
            print("success; found match!")
            print(elem_bytes)
            sys.stdout.flush()
            sys.exit(0)

print("failed; couldn't find a match!")
sys.exit(1)
```

Eigene Kennwort-Speicherfunktion in Java

Useradmin.java

```
import java.io.*;
import java.util.Scanner;

public class Useradmin implements Useradministration
{
    String file = "passwords.txt";

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Useradmin useradmin = new Useradmin();
        if (args[0].equals("addUser") && args[1].length() > 0) {
            Scanner sc=new Scanner(System.in);
            System.out.println("Please enter password...");
```

```
            char[] password=sc.next().toCharArray();
            useradmin.addUser(args[1], password);
        } else if(args[0].equals("checkUser") && args[1].length() > 0) {
            Scanner sc=new Scanner(System.in);
            System.out.println("Please enter password...");
            char[] password=sc.next().toCharArray();
            System.out.println(useradmin.checkUser(args[1], password));
        }
    }


    public void addUser(String username, char[] password) {
        BCrypt bcryptObj = new BCrypt();

        String passwordString = new String(password);
        String pw_hash = BCrypt.hashpw(passwordString, BCrypt.gensalt());
        String line = username + ' ' + pw_hash;
        try {
            PrintWriter writer = new PrintWriter(
            new FileOutputStream (
            new File(file),
            true /* append = true */));
            writer.println(line);
            writer.close();
        } catch (IOException e) {

        }

    }

    public boolean checkUser(String username, char[] password) {
        try(BufferedReader br = new BufferedReader(new FileReader(file))) {
            for(String line; (line = br.readLine()) != null; ) {
                String[] parts = line.split(" ");
                String passwordString = new String(password);

                if (parts[0].equals(username)){
                    return BCrypt.checkpw(passwordString, parts[1]);
                }
            }
        } catch (FileNotFoundException e) {

        } catch (IOException e) {

        }
        return false;
    }
}
```

Useradministraion.java

```
public interface Useradministration
{
    public void addUser(String username, char[] password);
    public boolean checkUser(String username, char[] password);
}
```

BCrypt.java

http://www.mindrot.org/projects/jBCrypt/