

Aufgabe 5: Passwortsicherheit

5.1.

Eigenschaften kryptographischer Hashfunktionen (optional) – Erläutern Sie drei Eigenschaften kryptographischer Hashfunktionen!

Antwort:

1. Zeichenfolge beliebiger Länge (Eingabewert) wird auf eine Zeichenfolge mit fester Länge (Hashwert) abbildet.
2. Einwegfunktion
3. Kollisionsresistenz

5.2.

Einfaches Hash-Verfahren (optional) – Durch kryptographische Hash-Funktionen können Kennwörter sicherer als im Klartext hinterlegt werden. Nennen Sie zwei Gründe, warum die Kennwörter in einem IT-System nicht im Klartext abgespeichert werden sollten. Wie funktionieren Kennwortspeicherung und -prüfung unter Verwendung einer Hash-Funktion im einfachsten Fall? Warum ist dieses Verfahren sicherer als die Abspeicherung im Klartext?

Antwort:

1. Wenn jemand Zugriff auf das System hat, kann er auch das Passwort nutzen.
 1. Ggf. wird das gleiche Passwort auch in anderen Systemen verwendet.
 2. Oder jemand kann sich ggü des Systems als Jemand anderes Identifizieren.
2. Um nicht auf <http://plaintextoffenders.com/> bloßgestellt zu werden.
 - a) Aus dem Passwort des Users wird (am besten lokal auf dem Gerät des Users) ein Hash des Passworts erzeugt, dieser wird mit dem Hash in der Passwortdatenbank verglichen.
 - b) Siehe 1.1 und 1.2

5.3.

Brute-Force-Angriff (Pflicht, 6 Punkte) – Bei vielen Unix-Betriebssystemen wurden früher lediglich die ersten acht Stellen eines Kennwortes verwendet. Wie lange benötigt ein Passwort-Cracking-Tool in diesem Fall maximal, das eine Million Passwörter pro Sekunde prüfen kann, wenn bekannt ist, dass das Passwort lediglich aus alphanumerischen Zeichen besteht. Wäre es im Vergleich dazu für das Passwort-Cracking-Tool aufwendiger, wenn das Betriebssystem keine Beschränkung der Kennwortlänge hat und bekannt ist, dass das Passwort nur aus Zahlen und maximal 16 Stellen besteht?

Antwort:

- a) alphanumerischen Zeichen, 8 Stellen:
52 Buchstaben + 10 Ziffern = 62 Zeichen
 62^8 Möglichkeiten für das Passwort
 $(62^8 / 1000000) / 60 / 60 / 24 = 2527$ Tage
- b) Zahlen und maximal 16 Stellen
10 Zeichen
 10^{16} Möglichkeiten
 $(10^{16} / 1000000) / 60 / 60 / 24 = 11570$ Tage

5.4.

Time-Memory-Trade-Off (optional) – Eine leistungsfähige Technik, mit der auf Basis eines Hashwerts ein dazu passendes Passwort ermittelt werden kann, stellen sogenannte Rainbow Tables dar. Zur Beantwortung können Sie <http://www.h-online.com/security/features/>

Cheap-Cracks-Of-dictionaries-and-rainbows-746217.html heranziehen.

Was versteht man in diesem Zusammenhang unter dem Begriff Time-Memory-Trade-Off ?

Was ist die grundsätzliche Idee von Rainbow Tables bzw. den Vorgänger-Verfahren? Was haben Rainbow Tables mit dem Regenbogen zu tun?

Antwort:

Die grundlegende Idee ist bei Passworthashes ohne Salt eine Datenstruktur möglicher Passwörter und Hashes vorher zu erstellen. Um beim Entschlüsseln von den Hashes auf die Passwörter schließen zu können. Die schnellste Version wäre eine Tabelle direkt im Arbeitsspeicher, diese Umsetzung benötigt jedoch viel von diesem. Andere Umsetzungen wären langsamer, jedoch weniger Arbeitsspeicher hungrig (Time-Memory-Trade-Off). Eine solche Datenstruktur sind Rainbow-Tables.

5.5.

Salting (optional) – Einen wirksamen Schutz gegen Rainbow Tables stellt das Hinzufügen einer zufälligen Zeichenkette (auch „Salt“ genannt) vor dem Anwenden der Hash-Funktion h auf ein Kennwort dar, was als $h(\text{SALT}|\text{PASSWORD})$ ausgedrückt werden kann. Warum?

Antwort:

Eine weitere Methode, die Generierung von Rainbow Tables unwirtschaftlich zu machen, ist der Einsatz von Salt. Dabei wird an das Passwort vor dem Hashen ein – im Idealfall – zufällig generierter Wert, das Salt, angehängt. Das Salt wird zusammen mit dem Hashwert gespeichert, um das Passwort später überprüfen zu können, es ist also kein Geheimnis. Der Salt vergrößert entsprechend die Menge der möglichen Passwörter.

5.6

Dictionary-Attack (Pflicht, 10 Punkte) – Schreiben Sie ein Programm (z. B. in Java, Ruby, Python oder Perl) zum Ermitteln eines Kennworts anhand eines Hashwerts mit einem Wörterbuch-Angriff. Besorgen Sie sich dazu ein deutsches Wörterbuch aus dem Internet. Testen Sie Ihre Implementierung mit den unten angegebenen Daten. Über das gespeicherte Kennwort sind die folgenden Fakten bekannt: Es ist ein deutsches Wort, steht im Wörterbuch, ist kleingeschrieben und nicht länger als 6 Zeichen. Der Salt wird beim Hashen vor das Passwort gestellt. Bei der Hash-Funktion handelt es sich um MD5.

```
# user:salt:hash
```

```
berta;xohth4dew5p8:14146888a9cb5e924987691876fb4252
```

Welches Kennwort konnten Sie ermitteln? Skizzieren Sie die Funktionsweise Ihres Programms anhand der wichtigsten Stellen Ihres Programms (bitte nicht separat abgeben, sondern ins PDF einfügen). Achten Sie dabei auf Verständlichkeit und Übersichtlichkeit.

Wie müsste das Programm erweitert werden, wenn der Salt im Vorfeld nicht bekannt wäre?

Antwort:

Gefundenes Passwort: sterne

```
#!/bin/bash
```

```
## user:salt:hash
```

```
#berta;xohth4dew5p8:14146888a9cb5e924987691876fb4252
```

```
#PW="1234"
```

```
salt="xohth4dew5p8"
```

```
#hash=$(printf '%s' $PW$salt | md5sum | cut -d ' ' -f 1)
```

```
hash=14146888a9cb5e924987691876fb4252
```

```
#numofcpu=4
```

```
numofcpu=$(cat /proc/cpuinfo | grep processor | wc -l)
```

```
# Anzahl der CPU-Kerne = Anzahl der Threads.
```

```
for ((z=1;z<=$numofcpu;z++)); do
```

```
for word in $(split --number=l/$z/$numofcpu ./deutsch.txt); do
```

```
length=$(echo $word | wc -m) #Berechnen der Wortlänge
```

```
if [ $length -lt 8 ] ; then #wenn Wort + EOL < 8 ...
```

```
word=$(echo $word | sed 's/\([A-Z]\)/\L\1/g') #Wort aus Wörterbuch alle Zeichen klein
```

```
var=$(printf '%s' $salt$word | md5sum | cut -d ' ' -f 1)
```

```
if [ $var = $hash ] ; then
```

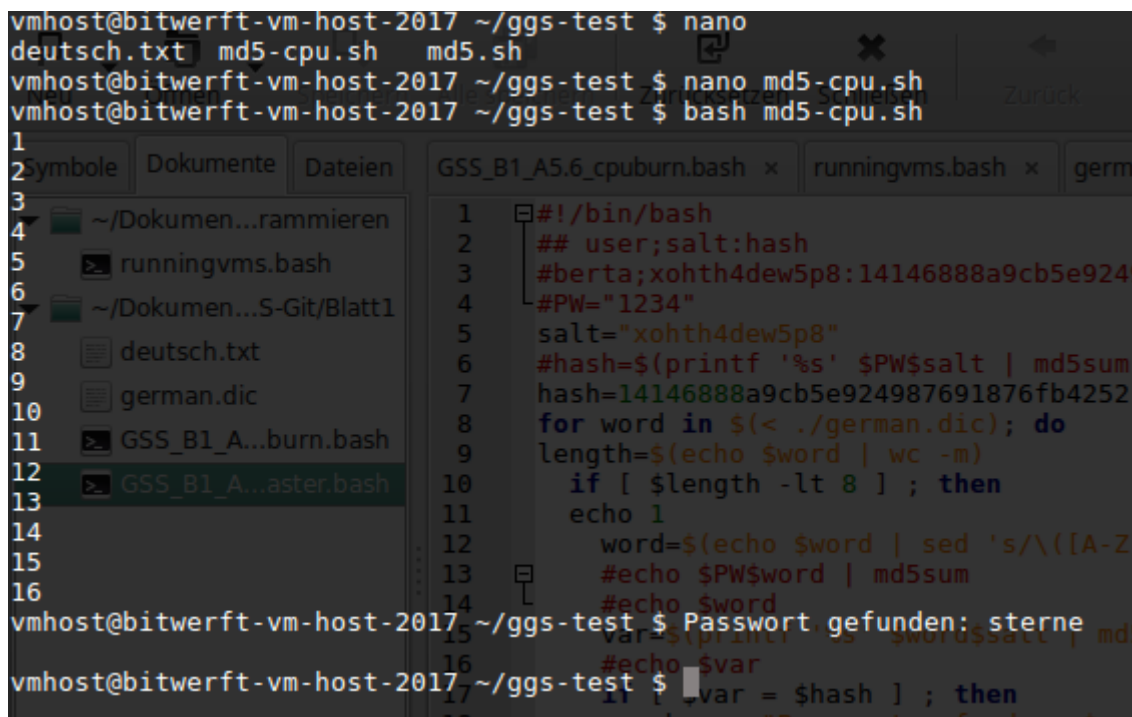
```
echo -e "Passwort gefunden: $word"
```

```
fi
```

```
fi
```

```
done & echo -e "Thread $z of $numofcpu started"
```

```
done
```



```
vmhost@bitwerft-vm-host-2017 ~/ggs-test $ nano
deutsch.txt md5-cpu.sh md5.sh
vmhost@bitwerft-vm-host-2017 ~/ggs-test $ nano md5-cpu.sh
vmhost@bitwerft-vm-host-2017 ~/ggs-test $ bash md5-cpu.sh
1
2 symbole Dokumente Dateien GSS_B1_A5.6_cpuburn.bash x runningvms.bash x germ
3
4 ~/Dokumen...rammieren
5 runningvms.bash
6 ~/Dokumen...S-Git/Blatt1
7
8 deutsch.txt
9 german.dic
10
11 GSS_B1_A...burn.bash
12 GSS_B1_A...aster.bash
13
14
15
16
vmhost@bitwerft-vm-host-2017 ~/ggs-test $ Passwort gefunden: sterne
vmhost@bitwerft-vm-host-2017 ~/ggs-test $
```

Wenn der Salt nicht bekannt wäre müsste nach einem Passwort in der maximalen Länge von Passwortlänge + Saltlänge gesucht werden.