Kryptografie

Immer wieder hörst du davon, dass bei großen Anbietern Datenbankinhalte gestohlen werden. In dem Artikel steht allerdings nichts davon, dass Passwörter gestohlen werden, sondern Hashes. Heute wirst du lernen, dass du jetzt schon ganz einfach viel besser sein kannst als LinkedIn und das mit nur ganz wenigen Codezeilen in Python.

Um alles möglichst gut zu verstehen, musst du auf jeden Fall die beiden oben verlinkten Artikel lesen oder wenigstens überfliegen.

Aufgabe 1:

Nimm eines deiner Passwörter und lasse es auf einer offenen Jupyter-Installation im Web laufen.

```
# wir weisen Python an, Kryptografiefunktionen zu laden
import hashlib
# wir fragen nach einem Passwort
crypted_phrase = input()
# Und geben den Hash des Passworts als MD5-Hash aus
print("Hash is:")
print(hashlib.md5(crypted_phrase.encode('utf-8')).hexdigest())
```

Nimm jetzt den Hash und kopiere ihn auf diese Seite. Nach Eingabe der Sicherheitsabfrage (reCaptcha) kannst du schauen, ob der Hash deines Passworts bereits bekannt ist (Wenn du das Verfahren mit dem Passwort „12345678“ durchführst, wirst du sehen, dass das „geknackt“ wird.

Wiederhole das Verfahren mit einem deiner Passwörter und folgendem Programm (mit „12345678“ klappt es! - auch mit deinem Passwort?):

```
# wir weisen Python an, Kryptografiefunktionen zu laden
import hashlib
# wir fragen nach einem Passwort
crypted_phrase = input()
# Und geben den Hash des Passworts als MD5-Hash aus
print("Hash is:")
# wir nutzen hier einen anderen Hashalogorithmus "hashlib.sha512" statt
"hashlib.md5"
print(hashlib.sha512(crypted_phrase.encode('utf-8')).hexdigest())
```

Wenn dein Passwort auch im zweiten Fall „geknackt“ wurde, hast du ein Problem, wenn du Opfer eines Datenbankdiebstahls wirst. Auch dein Anbieter wird nur Hashes in einer Datenbank speichern.

Informiere dich jetzt über den Unterschied zwischen dem md5- und dem sha512-Verschlüsselungsverfahren. Python kann folgende Verfahren „von Natur aus“: md5, sha1, sha224, sha256, sha384, sha512.

Aufgabe 2

Das Problem ist schon lange gelöst - mit nur wenigen Codezeilen mehr. Informiere dich über den Begriff „Salt“ in Verbindung mit Hashes.

```
import hashlib
# Statt "somestring" kannst bzw. solltest du möglichst wirres Zeug hier reinschreiben
salt = "somestring"
crypted_phrase = input()
salted_password = crypted_phrase + salt
print(hashlib.md5(salted_password.encode('utf-8')).hexdigest())
```

Wenn du jetzt versuchst, den Hash cracken zu lassen, klappt das nicht mehr, weil ein sogenanntest „salt“ (Salz) zum Passwort hinzugefügt wird. Bei unserem Programm verwendet jedes Passwort jedoch den gleichen Salt.

Auch dafür gibt es eine Lösung:

```
import hashlib, uuid
# Python schreibt nun für dich wirres Zeug hier hinein
salt = uuid.uuid4().hex
crypted_phrase = input()
salted_password = crypted_phrase + salt
print(hashlib.md5(salted_password.encode('utf-8')).hexdigest())
```

Hätten LinkedIn und andere die Benutzerpasswörter mit einem sicheren Algorithmus (z.B. sha512) gehasht und mit einem Salt versehen, wäre der Diebstahl der Datenbanken nicht so ein großes Problem, da es sehr lange dauern würde, die Passwörter aus den Hashes zu errechnen.

In der Praxis speichert man die Salts im Klartext zusammen mit den Hashes, meist durch ein Trennzeichen abgesetzt. Du kannst ja einmal überlegen, warum das **kein Problem** darstellt.

Aufgabe 3:

Schreibe folgende Programme:

1. Es wird zweimal ein Passwort abgefragt und dazu ein Hash berechnet. Stimmen beide Hashes (und damit die Passwörter) überein, soll das Programm die Ausgabe „Access granted!“ machen, ansonsten „Access denied!“ ausgeben.
2. Ein Programm fragt nach einem „Masterpasswort“ (password) und einem Domainnamen (salt). Es berechnet daraus einen Hash, den man als Passwort für die betreffende Webseite benutzen kann - wenn man immer das richtige Masterpasswort und den gleichen Domainnamen eingibt - quasi ein ganz einfacher Passwortmanager!

Ergänzender Kommentar

Das vorliegende Beispiel zeigt – finde ich – ganz gut, dass Medienkompetenz und Informatik sich sehr gut ergänzen können, teilweise vielleicht sogar einander bedingen. Das Problem der Passwortlänge und dem Passwortaufbau wird hier bewusst nicht angesprochen, weil das programmiertechnisch etwas anspruchsvoller ist. Das kommt dann in der Folgestunde. Weiterhin ist natürlich auch das sha-2-Verschlüsselungsverfahren moderneren Entwicklungen wie z.B. pbkdf2 weit unterlegen, aber auch programmiertechnisch wesentlich beherrschbarer. sha512 ist schon ganz ok, auch wenn heutige Grafikkarten ca. 200 Millionen Schlüssel pro Sekunde berechnen.