Präsentation “Secure Password Management”

Draft: Struktur

Note: Die angeführte Struktur ist die Langfassung von Punkten, die ich in irgendeiner Form erwähnen wollen würde - die Präsentation selbst hat aber natürlich weit weniger Fließtext auf den Folien

Intro

- Ziel: Ein User soll glaubhaft machen können, dass er der Besitzer eines Accounts ist
  - “Proof-by-ownership”: OTP/TOTP *(Präs B5 erwähnen)*, Hardware-Keys (U2F Standard), Biometrics (bissl hergeholt)
  - “Proof-by-knowledge”: Security Questions, Passwörter
- Problematik: Ich muss das Passwort speichern, um es abgleichen zu können - und wenn meine Application darauf zugreifen kann, kann es auch ein Angreifer[1]
  - *“Warum so schlimm? Eine komprimierte Datenbank ist doch eh schon ein worst-case Szenario …”* Weil Menschen dumm sind und ihre Passwörter wiederverwenden → ein Angreifer kann sich plötzlich in alle[2] Accounts aller User zugreifen, sofern die Passwörter wiederverwendet wurden. (Folie: Link zu https://haveibeenpwned.com/).
  - Paar Punkte zu
  - Ansatz 1: Passwort DB verschlüsseln → Problem: Schlüssel muss auch gespeichert sein → Loop zu Problem
  - Ansatz 2: Wenn es doch nur eine Einbahnfunktion gebe …

Hashing

- Eine Hashfunktion gibt bei selben Input auch wieder einen identen Output → anstelle von pw mit pw zu vergleichen, kann hash mit hash verglichen werden, aber aus hash kann nicht das passwort geschlossen werden (Hier die Hashfunktion mit einem hübschen Pfeil auf der Tafel verdeutlichen)
- Algorithmen Beispiele aufzählen
- Wenn ein input immer zum selben output führt, kann für jedes passwort der hash precomputed werden -> Rainbow Tables Demo

[1] Some effort required
[2] …nicht durch MFA geschützten…

Entropy

- Rainbow tables können praktisch nur beschränkt Daten speichern -> kein standard passwort verwenden

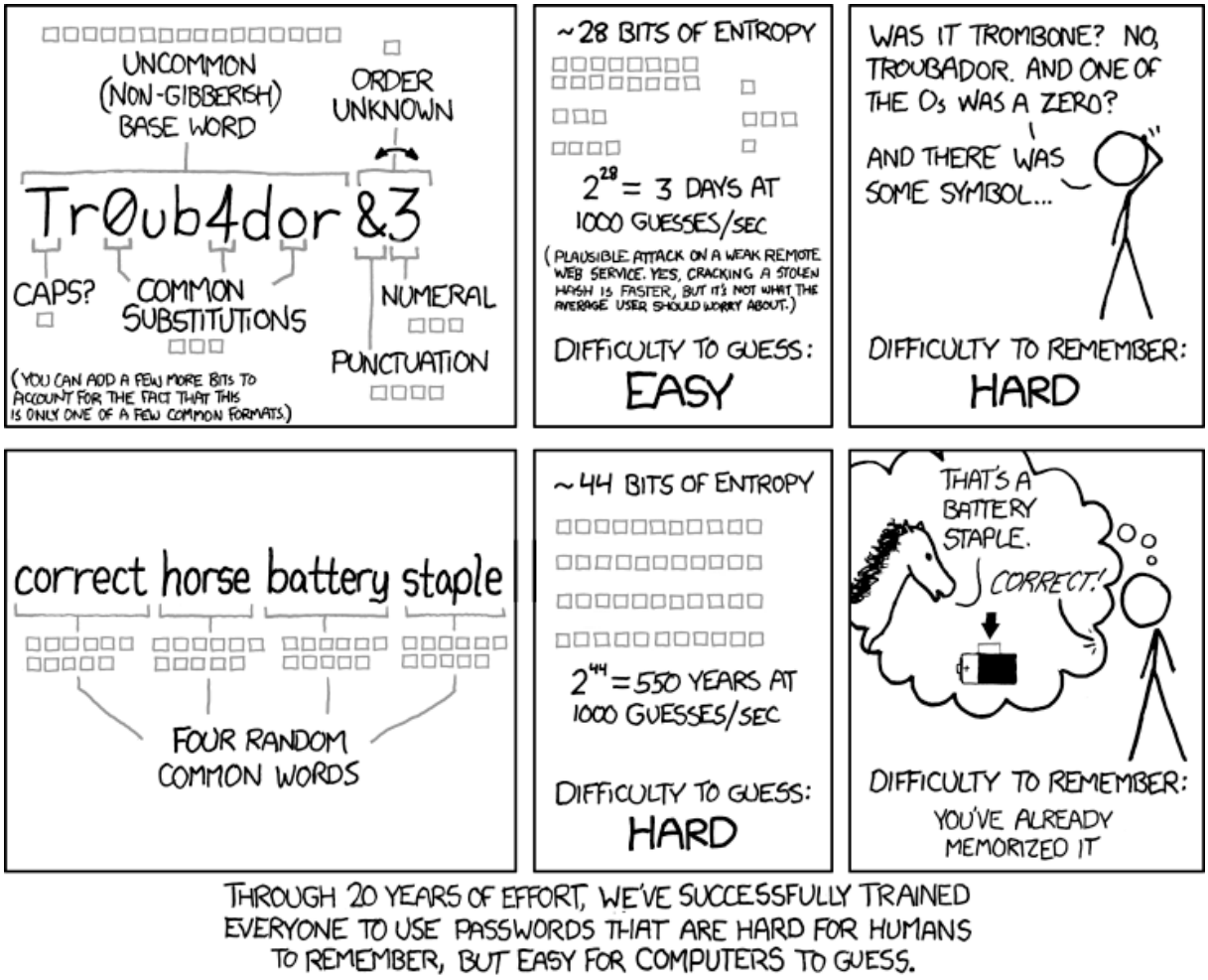


Figure 1: XKCD 936

- nis2 empfehlungen
- Entropy = Mögliche Passwörter aufgrund von Kriterien:
  - Länge
  - Character-Set (Buchstaben (klein/groß), Zahlen, (welche) Sonderzeichen?)
  - Wörter & Namen? → Wenn ja, Tauschregeln von Buchstaben (0 ↔ O, 1 ↔ l ↔ I ↔ !)

Salt & Pepper

- “Gratis” Entropy, am Server angehängt
- Salt: Unique pro Nutzer
- Pepper: Global für Application

Demo

https://gitea.nanopenguin.at/nanopenguin/itse-presentation (WIP)

TODOs:

- Mehr Password-Encoders, ggf. eigens geschrieben
- DB Anbindung (zur besseren Veranschaulichung)

Bonus 1: Encoding Migration

Falls die Zeit es zulässt

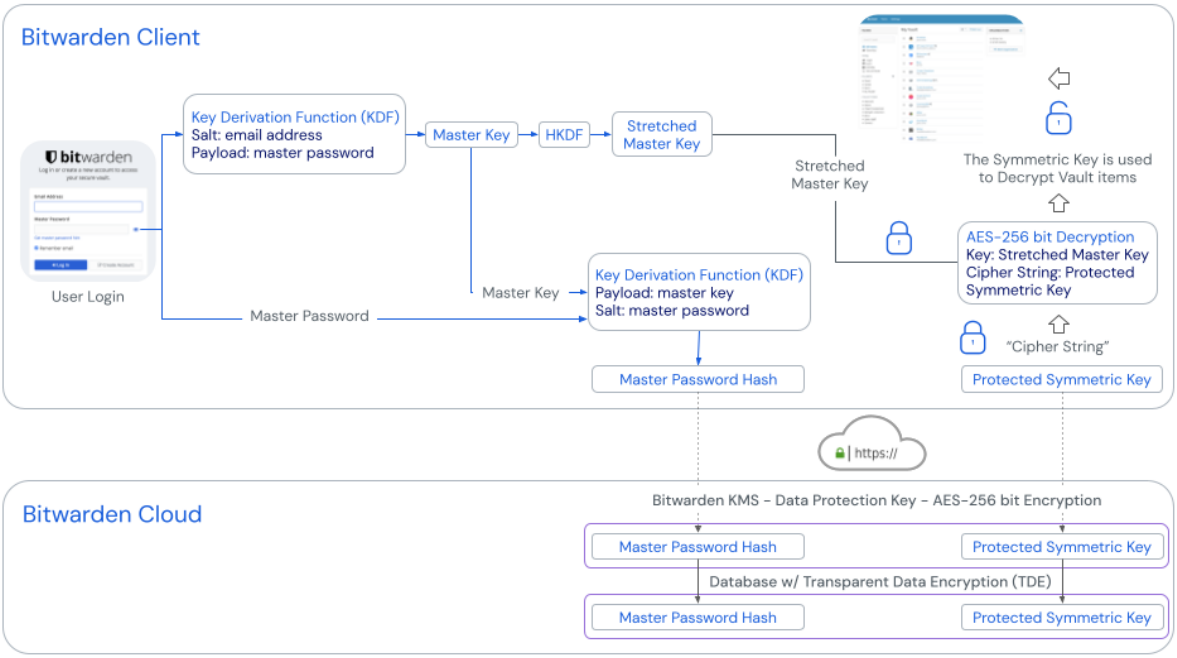
- Anhand der Code Demo, `{enc}`-Prefix
- Challenges: Da das Passwort nicht bekannt ist, muss sich der Nutzer erneut einloggen um das Passwort recodieren zu können

Bonus 2: Password Managers

Falls die Zeit es zulässt

- Mittels PBKDF2 (oder ähnlich): Master-Passwort wird zu einem Schlüssel um die restlichen Passwörter verschlüsseln zu können
- Beispiel Bitwarden (weil doku gut erklärt & eigeninteresse): https://bitwarden.com/help/bitwarden-security-white-paper/#authentication-and-decryption

Für Folien


Bitwarden Client
Key Derivation Function (KDF)
Salt: email address
Payload: master password
Master Key
HKDF
Stretched Master Key
Stretched Master Key
The Symmetric Key is used to Decrypt Vault items
AES-256 bit Decryption
Key: Stretched Master Key
Cipher String: Protected Symmetric Key
"Cipher String"
User Login
Master Password
Master Key
Key Derivation Function (KDF)
Payload: master key
Salt: master password
Master Password Hash
Protected Symmetric Key
https://
Bitwarden Cloud
Bitwarden KMS - Data Protection Key - AES-256 bit Encryption
Master Password Hash
Protected Symmetric Key
Database w/ Transparent Data Encryption (TDE)
Master Password Hash
Protected Symmetric Key