Authenticating With Passwords

Overview

Cryptography is essential for protecting passwords both in transit (via SSL/TLS) and at rest (in databases), where the latter requires secure storage methods to prevent exposure during a breach. Simply hashing passwords is insufficient due to rainbow tables, necessitating the use of a unique salt for each password before hashing to significantly improve security.

Key Information

- Data at Rest vs. In Transit: Passwords must be protected in transit (e.g., HTTPS/SSL/TLS) and when stored as data at rest in a database.
- Simple Hashing Flaw: Storing passwords as simple hashes is vulnerable because attackers can use pre-computed lists called rainbow tables to reverse the hash back to the original password.
- Salting: To counter rainbow tables, a unique, random value called a salt must be appended (or mixed) with the password before the hashing process.
- Improved Hashing: The combination of the password and its unique salt is then hashed, making the pre-computation of a rainbow table infeasible for all possible salt combinations.
- Key Derivation Functions (KDFs): For maximum security, functions like PBKDF2 are recommended; they use hashing but incorporate a high number of iterations to intentionally slow down the hashing process, which frustrates brute-force attacks.

Notes

Ways to store a Password

Least Secure (Plain Password)

UserName	Password
alice	qwerty

Better (Hash)

UserName	Hash (Password)
alice	d8578edf8458ce06fbc5bb76a58c5ca4

Best (Hash + Salt)

User	Hash (Password + salt)	Salt
alice	8a43db01d06107fcad32f0bcfa651f2f	12742

PBKDF2

PBKDF2 (Password-Based Key Derivation Function 2) takes the password and the salt and submits it through a certain number of iterations, usually hundreds of thousands

---

Task

1. You were auditing a system when you discovered that the MD5 hash of the admin password is `3fc0a7acf087f549ac2b266baf94b8b1`. What is the original password?
    1. Used Crack Station to get the value of the plain md5 hash.
    2. **qwerty123**

Conclusion

Protecting stored passwords requires more than simple hashing, which is easily defeated by rainbow tables, but mandates the use of a unique salt for every password to ensure a data breach only yields unique, un-invertible hash-salt pairs. For future-proofing against increased computing

power, best practice involves utilizing Key Derivation Functions like PBKDF2, which introduce high computational cost through iterative hashing.

Resources

- **TryHackMe**: Intro to Cryptography
- **Password Storage**: Cheat Sheet
- **MD5 Checker**: Crack Station

Revision #2
Created 2025-11-29 00:55:44 UTC by David Rizzo
Updated 2025-11-29 01:12:45 UTC by David Rizzo