INNOQ Technology Day / Online / 2023-11-13

iSAQB Advanced Level

WEBSEC

Sneak Peek

INNOQ

Christoph Iserlohn
Senior Consultant

28.08.2023

Passkeys

Nutzbar und sicher?

→

CHRISTOPH ISERLOHN
SONJA SCHEUNGRAB

26.06.2023

Browser Security

Was hat der Browser je für uns getan?

→

LISA MARIA MORITZ
CHRISTOPH ISERL

11.04.2023

KI und Security

Eine Risikobewertung

→

CHRISTOPH ISERLOHN
FELIX SCHUMACHER

13.03.2023

Firewall

Die Wiege der falschen Sicherheit

→

LISA MARIA MORITZ

CHRISTOPH ISERLOHN

26.01.2023

Das LastPass Drama

Passwort-Manager? Na klar. Aber doch nicht so!

→

LISA MARIA MORITZ
CHRISTOPH ISERLOHN

About the iSAQB e.V.

International Software Architecture Qualification Board

- non-profit organization founded in 2008
- approx. 20 founding members from industry, consulting and education
- iSAQB manages curricula

International Software Architecture
Qualification Board

About the advanced level

This course's content and structure match the standardized curriculum for software architects of the iSAQB e.V.

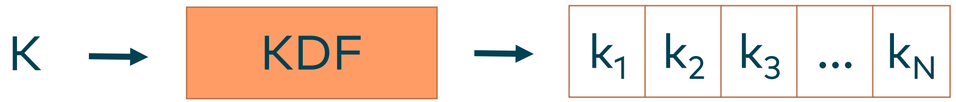
More information can be found under
http://www.isaqb.org/certifications/advanced-level/

International Software Architecture
Qualification Board

Agenda

- IT-Security & Protection Goals
- Risk Management
- Security Controls
- Secure Development Process, Design Principles & Patterns
- A "Birds View" on Cryptography
- Applied Cryptography aka Web Basics
- Attack vectors & Classifications
- Infrastructure & Operations

Deriving many keys from one



Stretch/Strengthen user supplied key

HKDF – a KDF from HMAC

RFC 5869

Step 1 (extract): k = HKDF(salt, K)

Step 2 (expand): k* = HKDF(k, CTX || N)

CTX – a string uniquely identifying an application
N – amount of derived keys

Password based KDF (PBKDF)

Deriving keys from passwords

Important:

- Passwords have a very low entropy
- HKDF is not applicable as derived passwords will be vulnerable to
  - Dictionary attacks
  - Lookup tables
  - Rainbow tables
  - Brute force attacks

Working with Passwords

… at Rest

- Plaintext – just don't do that!
- Encrypted & Integrity Protected – introduces a backdoor and requires a very high security measures for the encryption key(s)
- Hashed – Vulnerable to
  - Dictionary attacks, Lookup tables, Rainbow tables
  - Brute force attacks
  - Identification of users, which use same passwords
- …

Algorithm Requirements

So, we need something which is resistant to:

- Dictionary attacks
- Lookup tables
- Rainbow tables
- Brute force attacks
- Collision attacks
- Side-Channel attacks
- Identification of users with same passwords

And does not require additional security measures and infrastructure

Peppered hashes

How it works:

- Add a **secret** random string ("pepper") to the password before hashing it
- **Same** pepper for all passwords
- So *hashed password = H(password || pepper)*

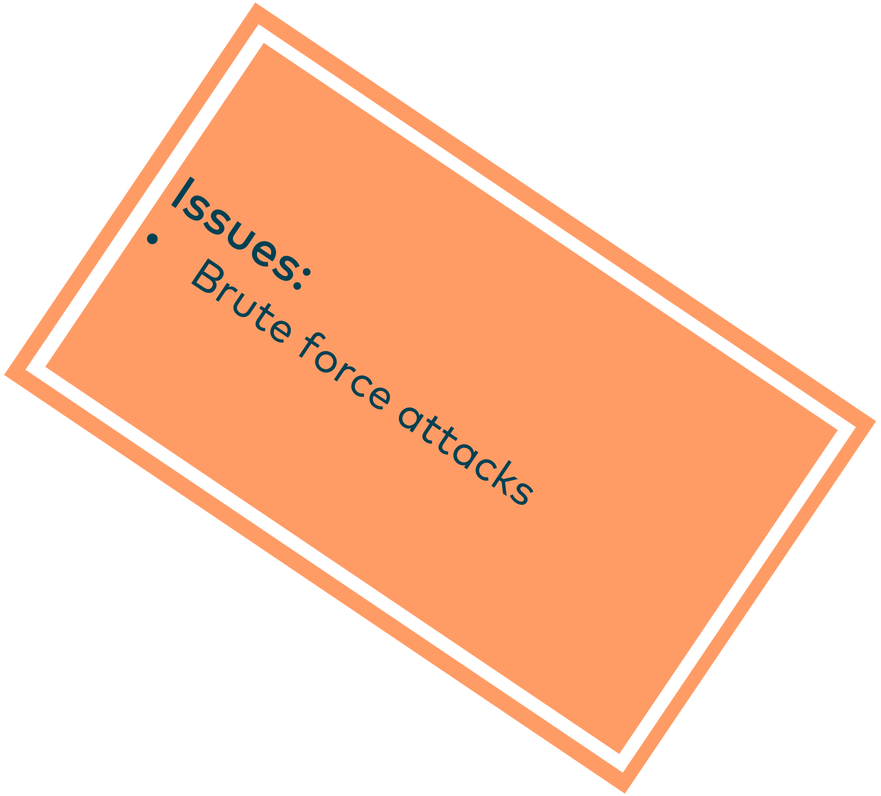
Issues:

- Brute force attacks
- Identification of users with same passwords
- Pepper must be protected

Salted hashes

How it works:

- Add a random string ("salt") to the password before hashing it
- Salt is unique per password
- Salt is not a secret and can be stored along the password
- So *hashed password = H(password || salt)*



And now?

Challenges with cryptographic hash functions

- Cryptographic hash functions are designed to be fast
- Can easily be implemented on GPUs or custom hardware (ASIC or FPGA) to parallelize computations
- Don't protect individual hashes against brute-force attacks (Blockchains)

So, we need something like:

"hash of hash of hash of hash ..." *(slow - CPU intensive)*

+

a lot of memory *(not parallelizable - memory hard)*

PBKDF2 - Password Based Key Derivation Function 2

- Family of key derivation functions defined in PKCS#5 as

derived key = PBKDF2(random function,
password,
salt,
number of iterations,
desired key length)

- Recommended by NIST and is widely used, e.g in Boxcryptor, Django, GRUB2, MediaWiki, WPA2, and others.

BCrypt

- Implements expensive key schedule Blowfish cipher (from 1999) as

$$hash = BCrypt(log_2(iteration\ count)=cost,\ salt\ of\ 16\ bytes,\ password\ [max\ 56\ bytes])$$

- Built for OpenBSD and used in many Linux distribution

SCrypt

- Family of KDF developed by Colin Percival for Tarsnap (from 2009). Defined as

$$
\begin{aligned}
\textit{derived key} = \textit{scrypt}(&\textit{password},\\
&\textit{salt},\\
&\textit{work factor } [(\textit{CPU} / \textit{memory})^2] \ (N),\\
&\textit{block size } (r),\\
&\textit{parallelization } [1..2^{32-1} * r/4] \ (p),\\
&\textit{desired key length})
\end{aligned}
$$

- Paper exists with formal proofs and cost estimations

Argon2

Multiple flavors exist:

- argon2i - focuses on side-channel attacks prevention,
- argon2d - focuses on brute-force-attacks prevention and
- argon2id – a combination of both. IETF defines it as the primary flavor of the algorithm

hash = argon2id(password, salt, number of iterations,
memory cost, parallelization,
length of resulting hash, version)

	Pros	Cons
PBKDF2	• CPU intensive (tunable) • Very mature - the algorithm itself has not been broken.	• not memory hard • can be efficiently implemented on GPUs and custom hardware like FPGAs
BCrypt	• CPU intensive (tunable) • Mature - the algorithm itself has not been broken.	• Password length is limited • Issues with non-ASCII chars and \0 termination • incompatible flavors exist • 4Kb not memory hard anymore
scrypt	• CPU intensive (tunable) • Memory hard (tunable) • Measures against parallelism	• there are few cryptoanalysis work results
Argon2	• CPU intensive (tunable) • Memory hard (tunable) • Measures against parallelism • Resistance against side-channel attacks	• very new - there are very few cryptoanalysis work results

Recommended Parameters

- Running time for interactive logins ≈ 100ms
- Running time for non interactive uses ≈ 3000ms
- Salt: At least 16, better 32 random bytes
- Memory ≈ 1-16 MB

Recommended Parameters

- PBDKF2: random function=SHA-2 or SHA-3 family, iterations => 600000, salt length >= 64-bit random string
- BCrypt: cost = 12 or more if possible (= 2^{12} = 4096 iterations)
- SCrypt: N = 16384, r = 16, p = 1; higher N is better than higher p
- Argon2: hash & salt length=128-bit, parallelization = 1, iterations=2, memory = >19MB

Estimated hardware cost for cracking

... a password in a year

KDF/password length	6 letters	8 letters	8 chars	10 chars	40 chars	80 chars
DES CRYPT	< 1$	< 1$	< 1$	< 1$	< 1$	< 1$
MD5	< 1$	< 1$	< 1$	1.1k$	1$	1.5T$
MD5 CRYPT	< 1$	< 1$	130$	1.1M$	1.4k$	$1.5 * 10^{15}$$
PBKDF2 (100ms)	< 1$	< 1$	18k$	160M$	200k$	$2.2 * 10^{17}$$
BCrypt (95ms)	< 1$	4$	130k$	1.2B$	1.5M$	48B$
SCrypt (64ms)	< 1$	150$	4.8M$	43B$	52M$	$6* 10^{19}$$
PBKDF2 (5.0s)	< 1$	29$	920k$	8.3B$	10M$	$11 * 10^{18}$$
BCrypt (3.0s)	< 1$	130$	4.3M$	39B$	47M$	1.5T$
SCrypt (3.8s)	$900	610k$	19B$	175T$	210B$	$2.3 * 10^{23}$$

Colin Percival, Stronger Key Derivation via Sequential Memory-Hard Functions, https://www.tarsnap.com/scrypt/scrypt.pdf

Password Hash Encoding

- The result of a password hash is just a hash value (raw bytes)
- So how to encode?
  - Remember Kerckhoff's principle
  - Self descriptive
- Before PHC
  - Modular Crypt Format
  - Other legacy schemes
- After PHC
  - PHC String Format

Modular Crypt Format (MCF)

$<id>$<content>

where

- id – an identifier representing the hashing algorithm/scheme
- content – content of the scheme [a-zA-Z0-9./]. Can be structured as well

Example

5QEDek12fCb8Hw.6U (5 stands for sha-256)

Modular Crypt Format (MCF)

- Ambigous, thus not portable
- Ad-Hoc
- Shouldn't be used

PHC Encoding

A well-defined and self-describing subset of the Modular Crypt Format

$<id>[$<param>=<value>(,<param>=<value>)*][$<salt>[$<hash>]]

where

- id - an identifier representing the hashing algorithm
- param – parameter name and its value, like rounds/iterations count, etc
- salt: Base64-like encoded salt
- hash: Base64-like encoded result of hashing the password and salt

PHC Encoding

$pbkdf2-sha256$c=2000,dklen=256$uOwP2QsXQ/YWncz9sHtKXAjhUOMSGR2PffFP2wq0RN4$KY2XI5CY0VOPi49P0I5x/8TeabYS/WOHNX0XojJVd38

Function	Parameters	Salt	Hash output
PKDF2 with SHA-256	Cost: 2000 Key Length: 256	Base64-encoded	Base64-encoded

- https://haveibeenpwned.com/

Questions? Answers!

INNOQ
www.innoq.com

Christoph Iserlohn
christoph.iserlohn@innoq.com
@ci@innoq.social

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180E
63067 Offenbach

Kreuzstr. 16
80331 München

Hermannstrasse 13
20095 Hamburg

Erftstr. 15-17
50672 Köln

Königstorgraben 11
90402 Nürnberg