bcrypt

un'introduzione nel mondo della cryptazione

Di Alberto Volpato

https://github.com/AlbertoVolpato

cos'è bcrypt

bcrypt è una funzione di hash crittografico utilizzata principalmente per la crittografia delle password. Invece di archiviare le password in chiaro nei database, bcrypt genera un hash criptografico della password e archivia l'hash. Quando un utente tenta di accedere al sistema, la password fornita viene nuovamente crittografata con bcrypt e il risultato viene confrontato con l'hash archiviato nel database. Se i due hash corrispondono, l'utente ha fornito la password corretta.

bcrypt è considerato uno dei metodi più sicuri per crittografare le password perché utilizza una combinazione di tecniche di hash e di "salting" (aggiunta di dati casuali al plaintext prima dell'hashing) per rendere più difficile l'individuazione della password originale da parte di un attaccante che ha accesso all'hash. Inoltre, bcrypt ha una "costante di lavoro" regolabile che consente di aumentare la complessità del processo di crittografia e quindi rendere più difficile la decrittazione delle password attraverso tecniche di forza bruta o di dizionario.

Installazione:

```
$ npm install bcrypt
```

or

```
$ yarn add bcrypt
```

successivamente nel nostro file .js

importiamo il modulo bcrypt

```
const bcrypt = require("bcrypt");
```

primo esempio di criptazione

```
const bcrypt = require("bcrypt");

bcrypt.hash("generic", 5, function (err, hash) {
  console.log(hash);
});
```

o codice sincrono:

```
const bcrypt = require("bcrypt");

const myPlaintextPassword = "generic";
const hash = bcrypt.hashSync(myPlaintextPassword, 5);
console.log(hash);
```

Una funzione di hashing richiede di aggiungere sale nel processo. Un salt è semplicemente un dato casuale che viene utilizzato come input aggiuntivo per la funzione di hashing per salvaguardare la tua password. La stringa casuale del sale rende l'hash imprevedibile.

Negli esempi di codice sopra, il salt viene generato automaticamente dal bcryptmodulo, ma puoi effettivamente generare il salt prima di eseguire l'hashing della password.

Per generare un sale, puoi utilizzare il metodo genSalt()

in codice:

```
bcrypt.genSalt(10, function (err, salt) {
  console.log(salt); // the random salt string
});
```

Una volta ottenuto il sale, potete passarlo al hash() così:

```
bcrypt.genSalt(10, function (err, salt) {
  bcrypt.hash("generic", salt, function (err, hash) {
    console.log(hash);
  });
});
```

in codice sincrono :

```
const salt = bcrypt.genSaltSync(10);
const hash = bcrypt.hashSync("generic", salt);
```

La generazione del sale per la tua funzione hash può variare da pochi secondi a molti giorni, a seconda di quanti round hai passato. bcrypt passerà attraverso piu round per generare il sale per darti un hash sicuro. Secondo la documentazione , ecco la quantità di tempo per elaborare la generazione di sale su un computer core a 2 GHz:

```
rounds=8 : ~40 hashes/sec
rounds=9 : ~20 hashes/sec
rounds=10: ~10 hashes/sec
rounds=11: ~5  hashes/sec
rounds=12: 2-3 hashes/sec
rounds=13: ~1 sec/hash
rounds=14: ~1.5 sec/hash
rounds=15: ~3 sec/hash
rounds=25: ~1 hour/hash
rounds=31: 2-3 days/hash
```

Dopo aver salvato l'hash nel database, puoi confrontare l'input di testo semplice dell'utente con l'hash memorizzato utilizzando il metodo compare().

Il compare() accetta tre parametri:

La semplice stringpassword per il confronto L'hash string creato in precedenza e la funzione callback, una volta terminato il processo di confronto Il metodo passerà l'errore e il valore booleano result, indicando se il confronto corrisponde o meno.

Ecco un esempio del metodo compare() in azione:

```
const bcrypt = require("bcrypt");

bcrypt.hash("generic", 5, function (err, hash) {

  bcrypt.compare("generic", hash, function (err, result) {
    console.log("generic:", result); // generic: true
  });

  bcrypt.compare("falsy", hash, function (err, result) {
    console.log("falsy:", result); // falsy: false
  });
});
```

Ecco un esempio del metodo compareSync() in azione:

```
const bcrypt = require("bcrypt");

const myPlaintextPassword = "generic";
const hash = bcrypt.hashSync(myPlaintextPassword, 5);
const result = bcrypt.compareSync(myPlaintextPassword, hash);
console.log(result); // true
```

Ecco un esempio di codice express per la cryptazione di password nel database mondogb creato da me:

```
const router = require("express").Router();
const User = require("../models/User");
const bcrypt = require("bcrypt");

//REGISTER
router.post("/register", async (req, res) => {
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPass = await bcrypt.hash(req.body.password, salt);
    const newUser = new User({
      username: req.body.username,
      email: req.body.email,
      password: hashedPass,
    });

    const user = await newUser.save();
    res.status(200).json(user);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

```
const router = require("express").Router();
const User = require("../models/User");
const bcrypt = require("bcrypt");

//LOGIN
router.post("/login", async (req, res) => {
  try {
    const user = await User.findOne({ username: req.body.username });
    !user && res.status(400).json("Wrong credentials!");

    const validated = await bcrypt.compare(req.body.password, user.password);
    !validated && res.status(400).json("Wrong credentials!");

    const { password, ...others } = user._doc;
    res.status(200).json(others);
  } catch (err) {
    res.status(500).json(err);
  }
});
```