
secure-password

build unknown  build failing

Making Password storage safer for all

Features

- State of the art password hashing algorithm (Argon2id)
- Safe defaults for most applications
- Future-proof so work factors and hashing algorithms can be easily upgraded
- [Buffers](#) everywhere for safer memory management

Usage

```
1 var securePassword = require('secure-password')
2
3 // Initialise our password policy
4 var pwd = securePassword()
5
6 var userPassword = Buffer.from('my secret password')
7
8 // Register user
9 pwd.hash(userPassword, function (err, hash) {
10   if (err) throw err
11
12   // Save hash somewhere
13   pwd.verify(userPassword, hash, function (err, result) {
14     if (err) throw err
15
16     switch (result) {
17       case securePassword.INVALID_UNRECOGNIZED_HASH:
18         return console.error('This hash was not made with secure-
19           password. Attempt legacy algorithm')
20       case securePassword.INVALID:
21         return console.log('Invalid password')
22       case securePassword.VALID:
23         return console.log('Authenticated')
24       case securePassword.VALID_NEEDS_REHASH:
25         console.log('Yay you made it, wait for us to improve your
26           safety')
27
28         pwd.hash(userPassword, function (err, improvedHash) {
29           if (err) console.error('You are authenticated, but we could
30             not improve your safety this time around')
```

```
28
29         // Save improvedHash somewhere
30     })
31     break
32 }
33 })
34 })
```

or with async await:

```
1  const securePassword = require('secure-password')
2
3  // Initialise our password policy
4  const pwd = securePassword()
5
6  const userPassword = Buffer.from('my secret password')
7
8  async function run () {
9      // Register user
10     const hash = await pwd.hash(userPassword)
11
12     // Save hash somewhere
13     const result = await pwd.verify(userPassword, hash)
14
15     switch (result) {
16         case securePassword.INVALID_UNRECOGNIZED_HASH:
17             return console.error('This hash was not made with secure-password
18             . Attempt legacy algorithm')
19         case securePassword.INVALID:
20             return console.log('Invalid password')
21         case securePassword.VALID:
22             return console.log('Authenticated')
23         case securePassword.VALID_NEEDS_REHASH:
24             console.log('Yay you made it, wait for us to improve your safety'
25             )
26
27             try {
28                 const improvedHash = await pwd.hash(userPassword)
29                 // Save improvedHash somewhere
30             } catch (err) {
31                 console.error('You are authenticated, but we could not improve
32                 your safety this time around')
33             }
34             break
35     }
36 }
37
38 run()
```

API

var pwd = new SecurePassword(opts)

Make a new instance of `SecurePassword` which will contain your settings. You can view this as a password policy for your application. `opts` takes the following keys:

```
1 // Initialise our password policy (these are the defaults)
2 var pwd = securePassword({
3   memlimit: securePassword.MEMLIMIT_DEFAULT,
4   opslimit: securePassword.OPSLIMIT_DEFAULT
5 })
```

They're both constrained by the constants `SecurePassword.MEMLIMIT_MIN`-`SecurePassword.MEMLIMIT_MAX` and `SecurePassword.OPSLIMIT_MIN`-`SecurePassword.OPSLIMIT_MAX`. If not provided they will be given the default values `SecurePassword.MEMLIMIT_DEFAULT` and `SecurePassword.OPSLIMIT_DEFAULT` which should be fast enough for a general purpose web server without your users noticing too much of a load time. However you should set these as high as possible to make any kind of cracking as costly as possible. A load time of 1s seems reasonable for login, so test various settings in your production environment.

The settings can be easily increased at a later time as hardware most likely improves (Moore's law) and adversaries therefore get more powerful. If a hash is attempted verified with weaker parameters than your current settings, you get a special return code signalling that you need to rehash the plaintext password according to the updated policy. In contrast to other modules, this module will not increase these settings automatically as this can have ill effects on services that are not carefully monitored.

pwd.hash(password, [function (err, hash) {}])

Takes Buffer `password` and hashes it. You can call `cancel` to abort the hashing.

The hashing is done by a separate worker as to not block the event loop, so normal execution and I/O can continue. The callback is invoked with a potential error, or the Buffer `hash`.

- `password` must be a Buffer of length `SecurePassword.PASSWORD_BYTES_MIN` - `SecurePassword.PASSWORD_BYTES_MAX`.
- `hash` will be a Buffer of length `SecurePassword.HASH_BYTES`.

If a callback is not specified, a `Promise` is returned.

```
var hash = pwd.hashSync(password)
```

Takes Buffer `password` and hashes it. The hashing is done on the same thread as the event loop, therefore normal execution and I/O will be blocked. The function may **throw** a potential error, but most likely return the Buffer `hash`.

`password` must be a Buffer of length `SecurePassword.PASSWORD_BYTES_MIN - SecurePassword.PASSWORD_BYTES_MAX`.

`hash` will be a Buffer of length `SecurePassword.HASH_BYTES`.

```
pwd.verify(password, hash, [function (err, enum) {}])
```

Takes Buffer `password` and hashes it and then safely compares it to the Buffer `hash`. The hashing is done by a separate worker as to not block the event loop, so normal execution and I/O can continue. The callback is invoked with a potential error, or one of the symbols `SecurePassword.INVALID`, `SecurePassword.VALID`, `SecurePassword.NEEDS_REHASH` or `SecurePassword.INVALID_UNRECOGNIZED_HASH`. Check with strict equality for one the cases as in the example above.

If `enum === SecurePassword.NEEDS_REHASH` you should call `pwd.hash` with `password` and save the new `hash` for next time. Be careful not to introduce a bug where a user trying to login multiple times, successfully, in quick succession makes your server do unnecessary work.

`password` must be a Buffer of length `SecurePassword.PASSWORD_BYTES_MIN - SecurePassword.PASSWORD_BYTES_MAX`.

`hash` will be a Buffer of length `SecurePassword.HASH_BYTES`.

If a callback is not specified, a `Promise` is returned.

```
var enum = pwd.verifySync(password, hash)
```

Takes Buffer `password` and hashes it and then safely compares it to the Buffer `hash`. The hashing is done on the same thread as the event loop, therefore normal execution and I/O will be blocked. The function may **throw** a potential error, or return one of the symbols `SecurePassword.VALID`, `SecurePassword.INVALID`, `SecurePassword.NEEDS_REHASH` or `SecurePassword.INVALID_UNRECOGNIZED_HASH`. Check with strict equality for one the cases as in the example above.

SecurePassword.VALID

The password was verified and is valid

SecurePassword.INVALID

The password was invalid

SecurePassword.VALID_NEEDS_REHASH

The password was verified and is valid, but needs to be rehashed with new parameters

SecurePassword.INVALID_UNRECOGNIZED_HASH

The hash was unrecognized and therefore could not be verified. As an implementation detail it is currently very cheap to attempt verifying unrecognized hashes, since this only requires some lightweight pattern matching.

SecurePassword.HASH_BYTES

Size of the `hash` Buffer returned by `hash` and `hashSync` and used by `verify` and `verifySync`.

SecurePassword.PASSWORD_BYTES_MIN

Minimum length of the `password` Buffer.

SecurePassword.PASSWORD_BYTES_MAX

Maximum length of the `password` Buffer.

SecurePassword.MEMLIMIT_MIN

Minimum value for the `opts.memlimit` option.

SecurePassword.MEMLIMIT_MAX

Maximum value for the `opts.memlimit` option.

SecurePassword.OPSLIMIT_MIN

Minimum value for the `opts.opslimit` option.

SecurePassword.OPSLIMIT_MAX

Maximum value for the `opts.opslimit` option.

SecurePassword.MEMLIMIT_DEFAULT

Default value for the `opts.memlimit` option.

SecurePassword.OPSLIMIT_DEFAULT

Minimum value for the `opts.opslimit` option.

Install

```
1 npm install secure-password
```

Credits

I want to thank Tom Streller for donating the package name on npm. The <1.0.0 versions that he had written and published to npm can still be downloaded and the source is available in his [scan/secure-password](#) repository

License

ISC