Python bcrypt

last modified November 29, 2021

Python bcrypt tutorial shows how to hash passwords in Python with the bcrypt library. It defines basic terms including encryption, hashing, and salt.

Python `bcrypt` module is a library for generating strong hashing values in Python. It is installed with `pip install bcrypt` command.

Encryption

Encryption is the process of encoding a message or information in such a way that only authorized people can read it with a corresponding key and those who are not authorized cannot. The intended information or message, referred to as *plaintext*, is encrypted using an encryption algorithm – a *cipher* – generating ciphertext that can be read only if decrypted. Encryption is a two-way function. When we encrypt something, we're doing so with the intention of decrypting it later. Encryption is used to protect data when transmitted; e.g. in a mail communication.

Hashing

Hashing is the process of using an algorithm to map data of any size to a fixed length. This is called a hash value. Whereas encryption is a two-way function, hashing is a one-way function. While it's technically possible to reverse-hash a value, the computing power required makes it unfeasible. While encryption is meant to protect data in transmit, hashing is meant to verify that data hasn't been altered and it is authentic.

Note: Hashing is not limited to security. It is also used to compare large amounts of data or fast key lookups.

Passwords are not stored as plaintext in databases but in hashed values.

Salt

Salt is a fixed-length cryptographically-strong random value that is added to the input of hash functions to create unique hashes for every input. A salt is added to make a password hash output unique even for users adopting common passwords.

The bcrypt hashing function

The *bcrypt* is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher. The bcrypt function is the default password hash algorithm for OpenBSD. There are implementations of bcrypt for C, C++, C#, Java, JavaScript, PHP, Python and other languages.

The bcrypt algorithm creates hash and salt the password for us using strong cryptography. The computation cost of the algorithm is parameterized, so it can be increased as computers get faster. The computation cost is called *work factor* or *cost factor*. It slows down the hashing, making brute force attempts harder and slower. The optimal cost factor changes over time as computers get faster. The downside of a high cost factor is increased load on system resources and affecting user experience.

Python bcrypt create hashed password

In the next example, we create a hashed password.

create_hashed_password.py

```
#!/usr/bin/env python

import bcrypt

passwd = b's$cret12'

salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)
```

```
print(salt)
print(hashed)
```

The example creates a salt and a hashed password with bcrypt.

```
import bcrypt
```

We import the `bcrypt` module.

```
salt = bcrypt.gensalt()
```

A salt is generated with the `gensalt` function.

```
hashed = bcrypt.hashpw(passwd, salt)
```

A hashed value is created with `hashpw` function, which takes the cleartext value and a salt as parameters.

```
$ python first.py
b'$2b$12$mwSIOyxLJid1jFLgnU0s0.'
b'$2b$12$mwSIOyxLJid1jFLgnU0s0.7pmzp8Mtx.GEO30x0AbI2v8r2sb98Cy'
$ python first.py
b'$2b$12$MgGs11HIXGkg1Bm1Epw0Du'
b'$2b$12$MgGs11HIXGkg1Bm1Epw0Du20TV8ppi2Latgq7kKng8UjM5ZFWKKeS'
```

Note that the salt is the first part of the generated hash value. Also note that each time a unique salt and hashed values are generated.

Python bcrypt check password

The following example checks a password against a hashed value.

check_passwd.py

```
#!/usr/bin/env python

import bcrypt

passwd = b's$cret12'

salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)

if bcrypt.checkpw(passwd, hashed):
    print("match")
```

```
else:
    print("does not match")
```

A password is checked with the `checkpw` function.

```
$ python check_passwd.py
match
```

Python bcrypt cost factor

The cost factor increases security by slowing down the hashing.

cost_factor.py

```
#!/usr/bin/env python

import bcrypt
import time

passwd = b's$cret12'

start = time.time()
salt = bcrypt.gensalt(rounds=16)
hashed = bcrypt.hashpw(passwd, salt)
end = time.time()

print(end - start)

print(hashed)
```

We set the cost factor with the `rounds` parameter to sixteen. We measure the time to generate the password hash.

```
$ cost_factor.py
4.268407821655273
b'$2b$16$.1FczuSNl2iXHmLojhwBZO9vCfA5HIqrONkefhvn2qLQpth3r7Jwe'
```

It took more that four seconds to generate the hash value with the specified cost factor.

In this tutorial, we have used the Python bcrypt module to generate password hashes.

List all Python tutorials.

s

Home Facebook Twitter Github Subscribe Privacy

© 2007 - 2021 Jan Bodnar admin(at)zetcode.com