Storing Passwords Securely With PostgreSQL and Pgcrypto

x-team.com/blog/storing-secure-passwords-with-postgresql/

July 4, 2017

code

There are 3 basic rules for keeping user credentials secure:

1. NEVER store passwords as plain text.
2. ALWAYS use a random salt when encrypting passwords.
3. DO NOT roll your own crypto.

Lucky for us, the pgcrypto module in PostgreSQL makes it very easy to follow these rules. Let us take a look at an example.

First, we need to enable pgcrypto:

```
CREATE EXTENSION pgcrypto;
```

Then, we can create a table for storing user credentials:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email TEXT NOT NULL UNIQUE,
  password TEXT NOT NULL
);
```

When creating a new user, we can use the `crypt` function to encrypt the password.

```
INSERT INTO users (email, password) VALUES (
  'johndoe@mail.com',
  crypt('johnspassword', gen_salt('bf'))
);
```

The `crypt` function accepts two arguments:

1. The password to encrypt
2. The salt to use when encrypting

We should always use the `gen_salt` function, to let PostgreSQL generate a random salt for us. I prefer using the blowfish algorithm (`bf`) with `gen_salt` , but here is a list of the algorithms you can use:

Table F-17. Supported Algorithms for `crypt()`

Algorithm	Max Password Length	Adaptive?	Salt Bits	Output Length	Description
bf	72	yes	128	60	Blowfish-based, variant 2a
md5	unlimited	no	48	34	MD5-based crypt
xdes	8	yes	24	20	Extended DES
des	8	no	12	13	Original UNIX crypt

To authenticate a user, we use `crypt` again, but this time we pass these arguments:

1. The submitted password
2. The encrypted password we already have in the database

If the password matches, `crypt` will return the same value as the one we already have in the database.

```
SELECT id
  FROM users
 WHERE email = 'johndoe@mail.com'
   AND password = crypt('johnspassword', password);

 id
----
  1
(1 row)


SELECT id
  FROM users
 WHERE email = 'johndoe@mail.com'
   AND password = crypt('wrongpassword', password);

 id
----
(0 rows)
```