How can I hash passwords in postgresql?

Asked 9 years, 6 months ago Active 3 years ago Viewed 67k times

47

29

I need to hash some passwords with salt on postgresql, and I haven't been able to find any relevant documentation on how to get that done.

So how can I hash passwords (with some salts) in postgresql?

security postgresql hash cryptography salt

edited Oct 29 '10 at 4:14
rook
48.8k ● 32 ● 142 ● 225

asked Apr 15 '10 at 16:30
Kzqai
15.9k ● 20 ● 91 ● 124

3 Answers

75

✓

It's been a while since I asked this question, and I'm much more familiar with the cryptographic theory now, so here is the more modern approach:

Reasoning

- Don't use md5. Don't use a single cycle of sha-family quick hashes. Quick hashes help attackers, so you don't want that.
- Use a resource-intensive hash, like bcrypt, instead. Bcrypt is time tested and scales up to be future-proof-able.
- Don't bother rolling your own salt, you might screw up your own security or portability, rely on gen_salt() to generate it's awesome unique-to-each-use salts on it's own.
- In general, don't be an idiot, don't try to write your own homegrown crypto, just use what smart people have provided.

Debian/Ubuntu install packages

```
sudo apt-get install postgresql   // (of course)
sudo apt-get install postgresql-contrib libpq-dev   // (gets bcrypt, crypt() and
gen_salt())
sudo apt-get install php5-pgsql   // (optional if you're using postgresql with php)
```

Activate crypt() and bcrypt in postgresql in your database

```
// Create your database first, then:
cd `pg_config --sharedir` // Move to the postgres directory that holds these scripts.
```

```
echo "create extension pgcrypto" | psql -d yOuRdATaBaSeNaMe // enable the pgcrypo
extension
```

Use crypt() and gen_salt() in queries

Compare :pass to existing hash with:

```
select * from accounts where password_hash = crypt(:pass, password_hash);
//(note how the existing hash is used as its own individualized salt)
```

Create a hash of :password with a great random salt:

```
insert into accounts (password) values crypt(:password, gen_salt('bf', 8));
//(the 8 is the work factor)
```

From-in-Php bcrypt hashing is slightly preferrable

There are `password_*` functions in php 5.5 and above that allow trivially simple password hashing with bcrypt (about time!), and there is a backward compatibility library for versions below that. *Generally* that hashing falls back to wrapping a linux system call for lower CPU usage anyway, though you may want to ensure it's installed on your server. See: https://github.com/ircmaxell/password_compat (requires php 5.3.7+)

Be careful of logging

Note that with pg_crypto, the passwords are in plaintext all during the transmission from the browser, to php, to the database. This means they can be logged *in plaintext* from queries if you're not careful with your database logs. e.g. having a postgresql slow query log could catch and log the password from a login query in progress.

In Summary

Use php bcrypt if you can, it'll lessen the time that the password remains unhashed. Try to ensure your linux system has bcrypt installed in it's `crypt()` so that is performant. Upgrade to at least php 5.3.7+ is highly recommended as php's implementation is slightly buggy from php 5.3.0 to 5.3.6.9, and inappropriately falls back to the broken `DES` without warning in php 5.2.9 and lower.

If you want/need in-postgres hashing, installing bcrypt is the way to go, as the default installed hashes are old and broken (md5, etc).

Here are references for more reading on the topic:

- http://codahale.com/how-to-safely-store-a-password/
- http://www.postgresql.org/docs/9.2/static/pgcrypto.html
- https://github.com/ircmaxell/password_compat

edited Oct 13 '16 at 19:25

answered Sep 8 '13 at 18:58
Kzqai
15.9k ● 20 ● 91 ● 124

2 So, it's better to do hashing with pgcrypto vs on the app-side? In general, should routine work hashing, guid generation, etc. be done by pg instead of the app? Thanks! – paulkon Oct 18 '14 at 23:52

1 I've edited my answer above with more detail. Since pg_crypto requires that the plaintext of the password hit the database queries, with the potential problems when incidental query loggings occurs, I recommend trying for in-php `password_hash()` first these days if you can make it happen. Bcrypt is the current state-of-the-art in password hashing, so it blows away the other options, whether in postgresql or php. Bcrypt hashing is resource-intensive by design, so if you use it in php, try to get bcrypt available from crypt() to decrease your server's resource use. – Kzqai Oct 19 '14 at 4:48

don't use md5, it's broken, postgres hear's "use md5" because that's what it uses to hash it's passwords... – xenoterracide Feb 25 '17 at 7:57

I had a GIANT freakout and rewrote a bunch of code after reading about the logging issue here. Then I realized that I was using parameterized queries, which meant that my db logs didn't have plaintext content in them. Notably, when using pgp encryption, I needed to pass my public/private key string as a parameter as well to avoid logging – deltree Apr 28 '17 at 19:24

16

An application should hash its passwords using key derivation function like bcrypt or pbkdf2. Here is more information on secure password storage.

... but sometimes you still need cryptogrpahic functions in a database.

You can use pgcrypto to get access to sha256 which is a member of the sha2 family. Keep in mind sha0,sha1 md4, and md5 are very broken and should **never** be used for password hashes.

The following is an alright method of hashing passwords:

```
digest("salt"||"password"||primary_key, "sha256")
```

The salt should be a large randomly generated value. This salt should be protected, because the hashes cannot be broken until the salt is recovered. If you are storing the salt in the database then it can be obtained along with the password hash using sql injection. Concatenating the primary key is used to prevent 2 people from having the same password hash even if they have the same password. Of course this system could be improved, but this is much better than most systems I have seen.

Generally it is best to do hashing in your application before it hits the database. This is because querys can show up in logs, and if the database server was owned then they could enable logging to get clear text passwords.

edited May 23 '17 at 12:18
Community ♦
1 ● 1

answered Apr 15 '10 at 16:33
rook
48.8k ● 32 ● 142 ● 225

1 Yep, I was a bit late. Deleted my answer, as you were first and more detailed ;). – Tagore Smith Apr 15 '10 at 16:36

@T Duncan Smith thanks man, i gave you some points for being a good SO member. – rook Apr 15 '10 at 16:46

Hmmm, the logging issue is a good point, I suppose, but for practical reasons I want to be able to run a sql statement to un-personalize the passwords (along with other personal information) in order to publish a cleaned database. – Kzqai Apr 15 '10 at 18:23

Err, could you clarify the statement "The salt should be a large randomly generated value"? Would I add that random value salt via the traditional means of string concatenation? Does: `update account set pswhash = crypt('global salt' || 'new password' || 'user created date', gen_salt('sha256')) where account_id = 5` or something like that actually make sense for creating the initial hash? Or am I missing something about the crypt() function? – Kzqai Apr 15 '10 at 18:27

5 The documentation on the pgcrypto page is pretty good, and clarifies why this is a really silly way to do a hash. In fact, a recipe for disaster. Use the crypt function, with the 'bf' hash instead. See more, including how to do a custom crack with a few billion salted hashes a second at codahale.com/how-to-safely-store-a-password – nealmcb Mar 28 '11 at 0:04

|

8

Examples and documentation on: http://www.postgresql.org/docs/8.3/static/pgcrypto.html

```
UPDATE ... SET pswhash = crypt('new password', gen_salt('md5'));

SELECT pswhash = crypt('entered password', pswhash) FROM ... ;
```

edited Apr 27 '16 at 16:19
Igor
26.3k ● 14 ● 64 ● 100

answered Apr 15 '10 at 16:35
Matej Puntar
562 ● 5 ● 6

1 Yeah, pgcrypto looks like what I'm looking for, but I'm having a hard time figuring out the usage, is the example usage saying that I don't have to hardcode my own salt into the hash? I.e. I do no longer have to provide my own salt data like: `update account set pswhash = crypt('global salt' || 'new password' || 'user created date', gen_salt('sha256')) where account_id = 5` ? or is salting still a manual process? – Kzqai Apr 15 '10 at 18:13

2 Using the md5 algorithm, without any iteration count (adaptation for increases in hashing speed over time) is a recipe for disaster. Instead, use 'bf': gen_salt('bf'). See more, including how to do a custom crack with a few billion salted hashes a second at codahale.com/how-to-safely-store-a-password – nealmcb Mar 28 '11 at 0:01

3 @Tchalvak correct - you no longer have to provide your own salt data. In fact the gen_salt also encodes the algorithm, which should really be 'bf' as I note above - see the reference for more. Given the use of 'bf', this answer is far superior to the one by rook. – nealmcb Mar 28 '11 at 0:07

1 This is the right answer. You should not use the `digest` function to encrypt passwords, it's not secure enough. Just make sure you use the Blowfish algoritm and not MD5. – GetFree Jun 4 '13 at 23:38