Secure passwords in PostgreSQL

Postgres Open 2013
Chicago, IL, USA

Magnus Hagander
magnus@hagander.net

Magnus Hagander

- PostgreSQL
  - Core Team member
  - Committer
  - PostgreSQL Europe
- Redpill Linpro
  - Infrastructure services
  - Principal database consultant

So what's this about

- We collect more and more data
- Let's focus on what *everybody* collects
- Which is valuable enough

Typical webapp

- Collects mandatory information:
  - Username
  - Password
  - Email

And then what happens?

- What typically happens?

And then what happens?

- You get hacked hacked
  - Seems to only be a matter of time
  - So plan for that!

So what do we do?

- Didn't we already solve this?
- Passwords are *hashed*!
  - We've even got extra advanced methods!

People still get hacked

- Hashed passwords prevent some hacks
- But "dumping" those still allow offline attacks
- Leaked email addresses are *valuable*
  - Valuable makes it a target

So what can we do?

- We can easily improve on this
- There is no reason for bulk downloads
- Your database can help
- So let's look at a typical webapp

The valuable users table

```
CREATE TABLE users (
  userid text,
  pwdhash text,
  email text
)
```

The SQL injection attack

- Lets the attacker do:

```
SELECT * FROM users
```

- And they get all data...
  - Hashed passwords for offline attacks
  - Email addresses for sale

Remind you of anything?

- Haven't we seen this before?

Remind you anything?

- Haven't we seen this before?
  - Like pre-1990?

Remind you anything?

- Haven't we seen this before?
  - Pre-1990
  - /etc/passwd

Remind you anything?

- Shadow passwords!!
  - Invented a long time ago (1988, SysV 3.2 - Linux 1992)
  - Why are we repeating the mistakes?

Shadow passwords in PG

- Shadow passwords are based on "views"
  - We have this in PostgreSQL
- Shadow passwords reqiures "suid"
  - We have this in PostgreSQL

Shadow passwords in PG

- The problem:

```
webapp=# SELECT * FROM users;
 userid |                  pwdhash                   |        email
--------+--------------------------------------------+---------------------
 mha    | $2a$06$1dtSqWdv0hfsbpDRsfZ9eOHlGoLUj...   | magnus@hagander.net
```

Shadow passwords in PG

```
webapp=# ALTER TABLE users RENAME TO shadow;
ALTER TABLE
webapp=# REVOKE ALL ON shadow FROM webuser;
REVOKE
```

Shadow passwords in PG

```
webapp=# CREATE VIEW users AS
webapp-# SELECT userid, NULL::text AS pwdhash, NULL::text as email
webapp-# FROM shadow;
CREATE VIEW
webapp=# GRANT SELECT ON users TO webuser;
GRANT
```

Shadow passwords in PG

```
webapp=> SELECT * FROM shadow;
ERROR:  permission denied for relation shadow
webapp=> SELECT * FROM users;
 userid | pwdhash | email
--------+---------+-------
 mha    |         |
```

Shadow passwords in PG

- But now it's useless...
- No way to log in

Shadow passwords in PG

```
webapp=# CREATE EXTENSION pgcrypto;
CREATE EXTENSION
```

pgcypto password hashing

- pgcrypto provides *crypt()*
- Dual-use function
- Create password hashes (salted, of course!)
- Validate password hashes

Shadow passwords in PG

```
CREATE OR REPLACE FUNCTION login(_userid text,
        _pwd text, OUT _email text)
 RETURNS text
 LANGUAGE plpgsql
 SECURITY DEFINER
AS $$
BEGIN
  SELECT email INTO _email FROM shadow
    WHERE shadow.userid=lower(_userid)
    AND pwdhash = crypt(_pwd, shadow.pwdhash);
END;$$
```

Shadow passwords in PG

```
webapp=> SELECT * FROM login('mha', 'foobar');
 _email
--------


(1 row)
webapp=> SELECT * FROM login('mha', 'topsecret');
         _email
----------------------
 magnus@hagander.net
```

Shadow passwords in PG

```
CREATE OR REPLACE FUNCTION set_password(_userid text, _pwd text)
RETURNS void LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
  UPDATE shadow SET pwdhash = crypt(_pwd, gen_salt('bf'))
   WHERE shadow.userid=lower(_userid);
END;
$$
```

Problems solved

- No bulk informaiton leak
- Can only get information *after* you have the passwod
  - But then you presumably have it already
- Protect selected attributes
- While maintaining database modelling properties

Problems created

- *SECURITY DEFINER* functions are a point of attack
- Be careful writing them
  - SQL injection inside SQL...

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander