SCRAM authentication

Heikki Linnakangas / Pivotal

pg_hba.conf

```
# TYPE  DATABASE   USER       ADDRESS           METHOD
# "local" is for Unix domain socket connections only
local   all        all                          trust

# Use plaintext authentication from localhost
host    all        all        127.0.0.1         plain

# Allow md5 authentication from example.com, with SSL
hostssl all        all        .example.com      md5

# Require SCRAM for everyone else
host    all        all        0.0.0.0/0         scram
```

PostgreSQL authentication methods

- Password-based:
  - `password` (plaintext)
  - ~~`crypt`~~
  - md5
  - **scram**
  - RADIUS / LDAP / PAM
- Others:
  - SSL certificate
  - kerberos

(Plain) Password authentication

Server: *Hey, what’s your password?*

Client: *“Swordfish”*

Server: *ok, cool*

Plain password authentication

- Obviously weak
  - Password sniffing
- Ok over SSL
  - With `sslmode=verify-full`
- Used by RADIUS, LDAP, PAM, BSD authentication methods!

MD5 authentication

Server: Here are 4 random bytes (salt). Please compute:

md5(md5(password || username), salt)

Client: 23dff85f7c38ee928f0c21ae710bba5d

Server: Ok, cool

MD5 weaknesses

*md5(**md5(password || username)**, salt)*

- Password guessing
  - My laptop can compute about 7 million MD5 hashes per second
- Replay
  - Only 4 billion unique 4-byte salts (birthday attack)
- Stolen hashes
  - You don’t need the original password to log in. The hash stored in `pg_auth.rolpassword` is enough.

Other MD5 issues

- Renaming a user invalidates the password
  - Because the hash includes the username
- `db_user_namespace` cannot be used
  - For same reason
- MD5 has a bad reputation

SCRAM to the rescue!

- **S**alted Challenge **R**esponse **A**uthentication **M**echanism
- To be precise, PostgreSQL implements SCRAM-SHA-256
- Defined by RFC 5802 and RFC 7677
- Challenge-response like MD5 authentication

SCRAM

Client: Hi! Here’s a random nonce:
r=fyko+d2lbbFgONRv9qkxdawL

Server: Hi! Here’s my random nonce, salt and iteration count:
r=fyko+d2lbbFgONRv9qkxdawL3rfcNHYJY1ZVvWVs7j,
s=QSXCR+Q6sek8bf92,
i=4096

Client: Here’s my proof that I know the password:
<ClientProof>

Server: Ok, cool. And here’s my proof that I knew it too:
<ServerProof>

SCRAM

- More resistant to dictionary attacks
  - The computation to guess password is much more resource intensive
  - Configurable iteration count
- Longer nonces defeat replay attacks
- The verifiers stored in `pg_authid.rolpassword` don’t allow impersonating the user

SCRAM-SHA-256

- Relatively simple implementation
  - < 1000 lines of code in libpq
- Relies only on SHA-256 hash function

Simple Authentication and Security Layer (SASL)

- *“The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms.”*
- Decouples authentication from application protocol (like PostgreSQL’s FE/BE protocol)
- SCRAM is one SASL authentication mechanism

SASL

- Currently, PostgreSQL has a built-in SCRAM-SHA-256 implementation
- Would be straightforward to add more SASL authentication mechanisms
- Could use an external library to add support for more (e.g. Cyrus libsasl)
- Client can use a library that implements SASL and SCRAM-SHA-256
  - Java has a very generic SASL implementation, but no built-in SCRAM-SHA-256 provider

Password verifiers

```
set password_encryption='plain';
create user plain_user password 'foo';

set password_encryption='md5';
create user md5_user password 'foo';

set password_encryption='scram';
create user scram_user password 'foo';
```

Password verifiers

```
scram-sha-256:<salt>:<iteration count>:<hashes>
```

```
postgres=# select rolname, rolpassword from pg_authid

  rolname   |        rolpassword
------------+-------------------------------------------------------------
 plain_user | foo
 md5_user   | md591334fcda28129398a9cdb3f551e3cc8
 scram_user | scram-sha-
256:pXLPIrUTzmEzow==:4096:6eee6029927f2bafd38063a6f7dcc110c13065863e641b65
a9b84157651a462d:3cad59b7017388e37ee7eb5db0e11c811f52d7008735d609204155540
d3e3b09

(3 rows)
```

Compatibility matrix

	Kind of verifier		
Authentication method in pg_hba.conf	`plain`	`md5` hash	`scram` verifier
password	✔	✔	✔
md5	✔	✔	✔[1]
scram	✔		✔

[1] Will use SCRAM, requires client support

SCRAM is not encryption!

- SSL is still recommended
  - SCRAM is only authentication, not encryption!

PostgreSQL implementation

- SCRAM-SHA-256
- Channel binding not supported
- Unicode normalization not implemented yet
- Username is always passed as empty

Migrating

1. Upgrade all clients
2. Set `password_encryption=’scram’`
3. Change all user passwords

Future, short-term

- Implement SCRAM-SHA-256 in all the drivers
  - JDBC, ODBC (uses libpq), Python, .Net, Ruby, …
- Add option to libpq to require SCRAM
- Unicode normalization

Future, long-term

- Allow storing SCRAM verifier in LDAP
- Zero-knowledge proof
  - SRP

Questions?