Encrypted PostgreSQL

PGCon 2009
Ottawa, Canada

Magnus Hagander
Redpill Linpro AB
Decide what your threat is is

- Everything comes at a cost
  - Performance or maintainability
- Encryption for the sake of encryption?
- Compliance/regulations?
Encryption at different layers

Application
- Application data encryption

Database
- Pgcrypto encryption functions

Storage
- Full harddrive/filesystem encryption
Encryption at different layers

Application
- Application data encryption

Database
- Pgcrypto encryption functions

Storage
- Full harddrive/filesystem encryption

SSL or VPN
Application data encryption

- Independent of the database
- Implemented in the application layer
  - No, we won't talk about the myriad of options here
Harddrive/filesystem encryption

- Independent of the database
- Filesystem och block device level
- Needs to keep fsync behaviour!
- Keeps all database functionality
- Where to store the key?
Pgcrypto

- Encryption as database functions
- Client independent
- Don't forget to encrypt the connection!
Pgcrypto - challenges

• Encryption is easy
  – Relatively speaking
  – As long as you don't invent your own!

• Key management is not
 Pgcrypto – overview

• Raw encryption
• PGP compatible encryption
• Hashing
pgcrypto: raw encryption

SELECT encrypt(data, key, type)
SELECT decrypt(data, key, type)
SELECT encrypt_iv(data, key, iv, type)

- Type: bf-cbc, aes-cbc, ... (ecb supported, but..)
- Operates on bytea, returns bytea
- gen_random_bytes() can be used to create key
pgcrypto: PGP encryption

pgp_sym_encrypt(data, password[, opt])
pgp_sym_decrypt(data, password[, opt])

- Operates on text in plaintext, bytea in ciphertext
  - armor(), dearmor()
- Takes gpg style options like cipher-algo=aes256
pgcrypto: PGP encryption

pgp_sym_encrypt(data, password[, opt])
pgp_sym_decrypt(data, password[, opt])

• Public key encryption also supported, but no key generation
• Will detect wrong key/corrupt data
pgcrypto: Hashing

- SELECT digest(txt, type)
  - Returns bytea, use encode() to get hex
  - Md5, sha1, sha<more>

- SELECT encode(
  digest('lolcats!', 'sha256'),
  'base64')
pgcrypto: Hashing

- `SELECT crypt('secret', gen_salt('bf'))`
  - Stores salt as part of hash
  - Autodetects algorithm
  - md5, bf, etc

- `SELECT hash=crypt('secret', hash)`
Key management

• Where to store the key
• How to protect the key
• How to access the key
• How to do key recovery
Searching encrypted data

- Sorry, can't really be done by index

- Match encrypted data for raw encrypted without padding
  - But this decreases security
  - And does «is equal» matching only

- Index on expression
  - But why did you encrypt in the first place?
SSL
SSL secured connections

- Encryption
- Man-in-the-middle protection
- Authentication
SSL secured connections

• Enabled on the server (ssl=yes)
• Optionally required through pg_hba
• Optionally required in libpq
SSL secured connections

• Need to protect data in both directions

• For example username/password

• Must *know* before connection is started

  – Unknown equals unprotected
SSL encryption

• SSL *always* requires a server certificate
• Can be self-signed
• Does not need to be known by client
Certificate chains

Issuer → Root certificate

Issuer → Intermediate certificate

Issuer → Server certificate
Certificate chains

- Issuer
- Root certificate
- Intermediate certificate
- Server certificate
- Self-signed certificate
SSL secured connections
Threats handled by SSL: Eavesdropping

SELECT * FROM secret_stuff

Client → SSL → Server
Eavesdropping

- Prevented by encrypting all data
- Key negotiation is automatic
- Server certificate used but not verified
Threats handled by SSL: Man in the middle

Client → Fake server → Server

Valid SSL session

Man in the middle
SSL server verification

- On top of encryption
- Validate that the server is who it claims to be
- CA issues certificate, can be self-signed
- CA certificate known by client
Threats handled by SSL: Man in the middle

Valid SSL session

Client

Fake server

Server
SSL client authentication

- On top of encryption
- Normally on top of server verification, but not necessary
- CA issued certificate on client
- Match CN on certificate to user id
- Protect client certificate!
SSL in libpq

- Controlled by `sslmode` parameter
- Or environment `PGSSLMODE`
- For security, must be set on client
  - Remember, `unknown = unsecure`
# Summary of libpq SSL modes

<table>
<thead>
<tr>
<th>Client Mode</th>
<th>Protect against Eavesdrop</th>
<th>Protect against MITM</th>
<th>Compatible with server set to...</th>
<th>Performance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSL required</td>
<td>SSL disabled</td>
</tr>
<tr>
<td>disable</td>
<td>no</td>
<td>no</td>
<td>FAIL</td>
<td>works</td>
</tr>
<tr>
<td>allow</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>prefer</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>require</td>
<td>yes</td>
<td>no</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-ca</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-full</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
</tbody>
</table>
### Summary of libpq SSL modes

<table>
<thead>
<tr>
<th>Client Mode</th>
<th>Protect against</th>
<th>Compatible with server set to...</th>
<th>Performance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eavesdrop</td>
<td>SSL required</td>
<td>SSL disabled</td>
</tr>
<tr>
<td>disable</td>
<td>no</td>
<td>FAIL</td>
<td>works</td>
</tr>
<tr>
<td>allow</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>prefer</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>require</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-ca</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-full</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
</tbody>
</table>
## Summary of libpq SSL modes

<table>
<thead>
<tr>
<th>Client Mode</th>
<th>Protect against Eavesdrop</th>
<th>Protect against MITM</th>
<th>Compatible with server set to...</th>
<th>Performance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSL required</td>
<td>SSL disabled</td>
</tr>
<tr>
<td>disable</td>
<td>no</td>
<td>no</td>
<td>FAIL</td>
<td>works</td>
</tr>
<tr>
<td>allow</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>prefer</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>require</td>
<td>yes</td>
<td>no</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-ca</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-full</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

- **Disable**: No overhead required.
- **Allow**: If necessary.
- **Prefer**: If possible.
- **Require**: Yes overhead required.
- **Verify-ca** and **Verify-full**: Yes overhead required.
## Summary of libpq SSL modes

<table>
<thead>
<tr>
<th>Client Mode</th>
<th>Protect against Eavesdrop</th>
<th>Protect against MITM</th>
<th>Compatible with server set to...</th>
<th>Performance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSL required</td>
<td>SSL disabled</td>
</tr>
<tr>
<td>disable</td>
<td>no</td>
<td>no</td>
<td>FAIL</td>
<td>works</td>
</tr>
<tr>
<td>allow</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>prefer</td>
<td>no</td>
<td>no</td>
<td>works</td>
<td>works</td>
</tr>
<tr>
<td>require</td>
<td>yes</td>
<td>no</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-ca</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
<tr>
<td>verify-full</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
<td>FAIL</td>
</tr>
</tbody>
</table>
Summary

- Only encrypt what you really need
- Only encrypted *where* you really need
- Key management is *hard*
- Many use-cases are very narrow
Encrypted PostgreSQL

Questions?

magnus@hagander.net

http://blog.hagander.net