Replication Security

PGConf NYC, 2014
New York City, NY

Magnus Hagander
magnus@hagander.net

Magnus Hagander

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

Replication security

- Increase of distributed systems
- Sometimes just local DC failover
- Cross-DC availability solution
- Geo/net-local performance
- (OK, I'll say it, cloud)

Two separate use cases

- Replication for failover
  - Server or data center HA
- Replication for performance
  - "Reporting nodes"
  - "Local read copy"
- Different security concerns!

Evolution of PostgreSQL replication

- erserver
- Slony
- Londiste
- etc

Evolution of PostgreSQL replication

- 8.3 added pg_standby
- 9.0 added Streaming Replication
- 9.0 added Hot Standby
- 9.1 added Synchronous Replication
- 9.4 will add logical replication

PostgreSQL replication recap

- Starts from base backup
- Streams all transaction log
  - pg_xlog contents
- Fallback to file-based through archive
  - Same pg_xlog contents

PostgreSQL replication recap

- Always cluster wide!
- Everything passes the replication channel
  - Eventually
- Database objects are identical
  - Users, roles, passwords
  - Permissions on all objects

PostgreSQL replication recap

- Configuration files not identical
- By default included in base backup
  - Depends on platform (hi, Debian!)
- postgresql.conf
  - Different memory settings on report nodes?
- pg_hba.conf / pg_ident.conf
  - Login and security settings

pg_hba.conf considerations

- Contents are not in sync
- Useful on read nodes
  - Different users/nets
  - Different replication permissions
  - Different security requirements

pg_hba.conf considerations

- Contents are not in sync
- Trouble at failover?
  - Different users can log in
  - Wrong database access
  - Can new replication nodes log in?
- Enforce and/or audit!

Database grants

- Same on master and slave
- Create reporting users on master
- Even if only used on slave
- Control access using pg_hba.conf

Streaming replication

- Uses standard Postgres connection
- To "virtual" database 'replication'
- Requires REPLICATION privilege
  - (or superuser)
- Can use standard PostgreSQL security features

REPLICATION privilege

- Attribute on user role
  - Not a grantable permission
  - Like superuser, createdb, createuser

```
CREATE USER replica WITH REPLICATION
```

- Can be combined with other attributes

```
CREATE USER replica
 WITH REPLICATION CONNECTION LIMIT 2
```

REPLICATION privilege

- Still a very high level privilege
  - Will see the whole database
  - "Read-only superuser"
  - Offline attacking etc
  - (9.4: block pg_xlog recycling)

REPLICATION privilege

- Always use instead of superuser
  - Especially for read nodes
  - Separation of privileges!
  - No code execution
  - No writing of files

Replication connection

- Uses standard PostgreSQL connection
- Standard authentication
- Standard security
- Treat as sensitive!

Replication vs pg_hba.conf

- "Virtual" database replication
- Not matched by all
- Always requires specific line

A bad example from reality

- When does this make sense

```
..
host all             all       10.0.0.0/24 md5
..
host replication replica  10.0.0.0/24 trust
..
```

A bad example from reality

- Or this

```
..
hostssl all             all         10.0.0.0/24 md5
..
host    replication replica  10.0.0.0/24 md5
..
```

Authentication

- All authentication methods are supported
- Never use trust
- md5 often a good choice
  - Easy to automate

Centralized authentication

- Centralized authentication typically a bad idea
- ldap
  - Usually not a good choice
  - External dependency for HA
- gss
  - Need to manage initial ticket
  - Need to manage ticket expiry

Network access

- Always limit network as much as possible
- By node or subnet
  - Not including application servers etc
  - Consider NAT issues
- Always restrict to replication user only

SSL

- A good choice for cross-DC replication
- Long-lived connections have lower overhead
- Consider certificate authentication!

```
..
host replication replica  10.0.0.5/32 cert clientcert=1
..
```

SSL

- Always verify server certificate
- Consider compression overhead
  - Disable on fast networks

```
primary_conninfo = 'user=replica host=10.0.0.1
                    sslmode=verify-full sslcompression=0'
```

Base backups

- Everything starts from base backup
- Secure as well (obviously)
- Making base backups
  - pg_basebackup
  - Manual

pg_basebackup

- Same security as replication will have
- So set up with same account/user
- Can use -R to create recovery.conf
- Use -d for connection string

```
pg_basebackup -D data.slave -P -R -d \
     "host=10.0.0.1 user=replica \
      sslmode=verify-full sslcompression=0"
```

Manual base backups

- Manual call to pg_start_backup/pg_stop_backup
  - Replication user needs access to regular database
- Backup files with "whatever technology"
  - Make sure this technology is secure
  - Typical uses are rsync or tar
  - Probably over ssh!

Pull or push base backups

- Push from master
  - Initiate rsync job on master
  - Master needs write permissions on slave
- Or pull on slave
  - Initiate rsync job on slave
  - Slave needs read permissions on master
- Or pull from backups
  - When full log archive exists

Remote SSH access

- Typical authorized_keys:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAIEAtk/kI... postgres@domain
```

- Share keys between multiple slaves?
- Manage key distribution!

Remote SSH access

- Typical authorized_keys:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAIEAtk/kI... postgres@domain
```

- Allows complete read access to system!
  - (or write if this was push)
- And arbitrary command execution!
- Restrict commands:

```
command="rsync --server ..." ssh-rsa AAA... postgres@domain
```

- (run rsync in verbose mode to find exact command)

Log archive

- Similar concerns as manual base backups
  - Should always be push
  - Push controlled by PostgreSQL
- Centralized log archive reduces key mesh
  - Consider for base backup distribution
- Control access per server

Encrypting logs and backups

- Depends on "trust domains"
- Do you trust backup/log location less than slaves?
- If not, then no point
- "Yes" is fairly common
  - E.g. S3 or other cloud stores

Encrypting logs and backups

- Supported by some tools
- Just encrypt all files before sending
  - Consider sending locally first!
  - Same scenario as compression
- Encryption is slow, decryption is fast
- Symmetric shared-key easiest
  - Or just use GPG

Synchronous replication

- Adds extra DOS possibility
- Forcing sync slaves to disconnect
- Use up all connections

Synchronous replication

- Log in and "pretend" to be sync
  - Sync rep trusts client!
  - (Nope, "secret" application names don't work)
  - Restrict your replication clients!
- Never use "*" for client names
  - For non-security reasons

Conclusions

- Replication sees all your data
- "Keys to the kingdom"
- It's not hard to secure it
- But needs to be part of security design

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander