A PostgreSQL Security Primer

PGDay'15 Russia
St Petersburg, Russia

Magnus Hagander
magnus@hagander.net

Magnus Hagander

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

Security

Security

- It's hard

Security

- It's hard
  - No, really!

Security

- There is no one solution

Security

- There is no one requirement

Security

- PostgreSQL provides a toolbox
- You don't need everything
- Maybe you don't need anything...

Agenda today

- Environment
- Communication
- Authentication
- Application

Agenda today

- Environment
- Communication
- Authentication
- Application

Secure PostgreSQL Environment

- Only as secure as the environment
- If someone owns the OS, they own the db
  - Owns the server -> owns the OS
  - Owns the datacenter -> owns the server
- Defined trust levels!
  - e.g. outsourcing/cloud vendors

Operating system

- Pick your operating system
  - Something you know
  - Regardless of PostgreSQL
- Secure "reasonably"
- No other local users!

Operating system

- Use standard installers
  - Don't roll your own
- Usually adapted for OS
  - E.g. SELinux
- Consistent security!

Operating system

- Keep updated
- Both operating system and PostgreSQL
- yum/apt makes it easier
  - But you have to use it!
- Monitor!
- Mind restarts!

Operating system

- Encrypted disks?
  - Performance/reliability implications
  - Attack vectors?
- Key management?
  - What happens on restart?

Multi instance

- Different security domains?
- Different OS user
  - Sometimes not well packaged
- Virtualization/containers?

Agenda today

- Environment
- Communication
- Authentication
- Application

Securing communications

- Do you need it?
  - Attack vectors?
- Overhead!

Securing communications

- (physical)
- Firewalls
- VPN
- ipsec
- SSL

Firewalls

- PostgreSQL traffic is simple
  - Single port TCP
- Block at perimeter
- Block at host
  - (Does not replace pg_hba!)

VPN

- Many scenarios
  - Site -> Site
  - Host -> Site
  - Host -> Host
- Typically ipsec or pptp
- Combine with firewall!

IPSEC

- Transport security
- Individual connections
- Allows for detailed policies
- Kernel/system implementation

SSL

- Connection encryption
- Individual connections
- Protocol adapted

SSL in PostgreSQL

- OpenSSL only (sorry)
  - Abstraction in 9.5
  - No other implementations yet
- Certificate/key
  - Like any other service
- Disabled by default on server
  - Enabled on client!!

SSL in PostgreSQL

- Negotiated upon connection
- Same port!
- First packets of exchange
- Before authentication etc

Certificates

- Server certificate mandatory
- Does not need public ca
  - Probably should not use public ca
- "Snakeoil" self-signed works
  - But no MITM protection!
- Use custom (dedicated?) CA!

OpenSSL CA

- OpenSSL comes with built in CA
- Or use other CA software
- Always distribute CA certificate
  - But not the key

Setting up certificate

- Generate secret and public key
- Generate certificate request
- Sign :g:`certificate request
- Deploy certificate

Generating OpenSSL cert

```
$ openssl req -new -newkey rsa:4096 -text -out server.req
```

- General SSL parameters apply
  - Use large enough keys!
  - Always set CN to server name
  - Other attributes ignored

Generating OpenSSL cert

- OpenSSL always secures key with passphrase
- Makes auto-start impossible
- Remove key:

```
$ openssl rsa -in privkey.pem -out server.key
$ rm privkey.pem
```

Generating OpenSSL cert

- Securely store server.key
- Transfer server.req to CA
  - Does not have to be secured
  - If you verify fingerprint!

Sign certificate request

- Use your CA
  - For example, OpenSSL built-in one
- Or generate self-signed:

```
$ openssl req -x509 -in server.req -text -key server.key -out server.crt
```

- Securely transfer server.crt

Distribute CA certificate

- Each client needs cert to verify CA
- Not required, but strongly recommended
  - `~/.postgresql/root.crt`
- Also distribute CRL if used
  - `~/.postgresql/root.crl`
- Connection string can override file names

Enable server SSL

- Set ssl=on
- server.key/server.crt in data directory
  - Check permissions!
  - Should be 0600, must be 0x00.
- Restart, done.

CA Certificate on server

- Required for client certificate auth
  - `root.crt`
- CRL not required but recommended
  - `root.crl`
- File names controllable in postgresql.conf

SSL negotiation

- SSL negotiated between client and server
- Server provides
- Client decides
- Controlled by sslmode parameter

SSL negotiation

- sslmode default is prefer
  - This is stupid....
- No guarantees
- Don't use!

SSL negotiation

	Protect against		Compatible with server set to...		Performance
Client Mode	**Eavesdrop**	**MITM**	**SSL required**	**SSL disabled**	**overhead**
disable	no	no	FAIL	works	no
allow	no	no	works	works	If necessary
prefer	no	no	works	works	If possible
require	yes	no	works	FAIL	yes
verify-ca	yes	yes	works	FAIL	yes
verify-full	yes	yes	works	FAIL	yes

SSL enforcement

- Client decides??!!?!?!
  - Huh??
- Client decides, but server can reject
- Using hostssl in pg_hba.conf

SSL enforcement

```
..
hostssl xxx yyy ...
..
```

- Always use!

Client certificates

- Not required by default
- Can be requested by server
  - clientcert=1 in pg_hba.conf

```
..
hostssl xxx yyy zzz abc clientcert=1
..
```

Client certificates

- Provide in PEM format file
  - Or through OpenSSL compatible engine
- Validated against root CA on server
  - PostgreSQL specific root
- By default just needs to exist

Client certificate authentication

- Use for full login
- Username extracted from CN attribute
- Must chain to known trusted CA
- Can map using pg_ident.conf

Agenda today

- Environment
- Communication
- Authentication
- Application

Authentication

- Make sure it's the correct user
- And that they can prove it

Authentication

- PostgreSQL supports many methods
  - Host Based Authentication
- Combined in the same installation!
- Don't just "dumb down"

pg_hba.conf

- Top-bottom file
- Filter by:
  - Connection type
  - User
  - Database
  - Connection source
- "Firewall" and authentication choice

pg_hba.conf

- Order by most specific:

```
local       all     all                             peer
host        all     all       127.0.0.1/32          md5
hostnossl   webdb   webuser   10.1.1.0/30           md5
hostssl     all     +admin    192.168.0.0/24        gss
```

- Implicit reject at end

Authentication methods

- Many choices
  - Internal
  - OS integrated
  - Fully external
- And some really bad ones...

trust

- Trust everybody everywhere
  - Why would anybody claim they're someone else?
- "Turn off all security"
- Any use case? Maybe one...

trust

- Use it? Change it!

peer

- Only over Unix sockets
  - Sorry Windows, sorry Java
- Local connections only
- Asks OS kernel
  - Trustworthy!

md5

- Simplest one?
- Username/password
- Double MD5-hash
- Do not use "password"

ldap

- Looks like password to client
  - Regular prompt
  - Passed over to LDAP server
  - No special support needed
- Construct URLs different ways
  - Prefix+suffix
  - Search+bind

ldap

- Suffix and prefix

```
ldapprefix="CN=" ldapsuffix=", DC=domain, DC=com"
```

- Binds to

```
CN=mha, DC=domain, DC=com
```

ldap

- Double binding

```
ldapbasedn="DC=domain, DC=com"
 ldappbinddn="CN=postgres, DC=domain, DC=com"
 ldapbindpasswd="supersecret"
 ldapsearchattribute="uid"
```

ldap

- Double binding URL syntax

```
ldapurl="ldap://1.2.3.4/dc=domain, dc=com?uid?sub"
 ldappbinddn="CN=postgres, DC=domain, DC=com"
 ldapbindpasswd="supersecret"
```

ldap

- Cleartext!
  - Use with ldaptls=1
  - Use with hostssl
- Password policies from LDAP server
- Only authentication!

gss

- Kerberos based GSSAPI
  - Including Active Directory
- Single Sign-On
  - No password prompt!
  - All Kerberos supported auth methods
- Secure tickets
- "krb5" deprecated/removed

gss

- Uses kerberos keytabs
- Uses principals and realms
  - Similar to users and domains
- Mutual authentication
- Default service principal
  - postgres/server.domain.com
  - Case sensitive!

gss

- Install keytab
  - Readable by PostgreSQL
  - Can be specific for PostgreSQL or shared
  - Any principal will be accepted
  - But must match client!

gss

- Client principals
  - `user@domain.com`
- Matched with or without realms
  - Recommendation is to always include
  - Strip with pg_ident.conf

gss

```
gss include_realm=1 map=gss
```

- can also restrict realms

```
gss include_realm=1 krb_realm=DOMAIN.COM
```

radius

- Looks like password to client
  - Use with hostssl!
- Shared-secret encryption to Radius server
- Common for OTP solutions

radius

```
radiusserver=1.2.3.4
 radiussecret=supersecret
```

cert

- Map client certificate to login
  - Uses CN attribute
- Any certificate "engine" supported by OpenSSL
  - Normally uses PEM encoded files

cert

- Server must have CA certificate
  - And CRL if used
- Client must have CA certificate
  - And CRL if used

User name mapping

- External systems with different usernames
  - Peer
  - gss/sspi
  - cert
- Allow static or pattern mapping

User name mapping

- pg_hba.conf:

```
local      all    all                peer map=local
hostssl    all    all 10.0.0.0/24    gss  map=gss includerealm=1
hostssl    all    all 0.0.0.0/0      cert map=cert
```

User name mapping

- pg_ident.conf:

```
local    root                          postgres
..
gss      /^(.*)@DOMAIN.COM$/   \1
..
cert     /^cn=(.*)$/                   \1
```

Agenda today

- Environment
- Communication
- Authentication
- Application
- Summary

Application security

- Huge topic
- Let's stick to a few tips...
- And an example or two

Superuser

- Never use superuser
- Disables all security
  - Allows arbitrary code execution!
  - Allows replacement of configuration!

Database owner

- Avoid using database owner
- Overrides any object permissions
  - But much better than superuser

Schema boundaries

- Schemas for compartmentalization
- USAGE required to access all objects
- Object permissions required as well
- Sub-divide access

Password management

- Specifically considering webapps
- Lots of data collected today
  - Username
  - Password
  - Email
- and more

And then what happens?

- What typically happens?

And then what happens?

- You get 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 of anything?

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

Remind you of anything?

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

Remind you of 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 requires "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

SECURITY DEFINER

- Functions with SECURITY DEFINER
- Acts like setuid binary
- Powerful access

SECURITY DEFINER

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

SECURITY DEFINER

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

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

SECURITY DEFINER

- Beware!!
  - SQL-in-SQL injections
  - Unbounded data access
- Never use superuser

Agenda today

- Environment
- Communication
- Authentication
- Application
- Summary

Security

- Determine your requirements
- Determine your trust levels
- Determine your attack surface
- Determine your threat vectors

Security

- Deploy correct countermeasures
  - "Checkbox featuring" is useless
  - Or even counterproductive
- Lock all doors
  - E.g. why encrypt disks if keys are local?
  - Why require smartcards if data is cleartext?

Layered security

- A firewall alone doesn't protect you
- Doesn't mean you shouldn't have one

Too simple to mention

- Never use trust
  - (not even in testing)
- Use pg_hba.conf
  - Mix auth methods
  - Restrict IP addresses
- Go SSL if you have to

Iterative process

- Re-evaluate
- Requirements and landscape are dynamic!
- Stay secure!

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander
http://www.hagander.net/talks/