Secure PostgreSQL Deployment

PGDay'14 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...
Secure PostgreSQL Deployment

• Environment
• Communication
• Authentication
Secure PostgreSQL Applications

- Authorization/Permissions
- Roles
- Security barrier views
- Security definer functions
- RLS
- etc...
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
- Consistent security!
Operating system

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

- Encrypted disks?
  - Performance/reliability implications
- Key management?
  - What happens on restart?
Multi instance

• Different security domains?
• Different OS user
  • Sometimes not well packaged
• Virtualization/containers?
Securing communications
Securing communications

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

- (physical)
- VPN
- ipsec
- SSL
SSL in PostgreSQL

- OpenSSL only (sorry)
- Certificate/key
  - Like any other service
  - Disabled by default on server
  - Enabled on client!!
Certificates

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

• Set ssl=on
• server.key/server.crt in data directory
  • Check permissions!
• Restart, done.
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
## SSL negotiation

<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>MITM</td>
<td></td>
</tr>
<tr>
<td>disable</td>
<td>no</td>
<td>no</td>
<td>FAIL</td>
</tr>
<tr>
<td>allow</td>
<td>no</td>
<td>no</td>
<td>works</td>
</tr>
<tr>
<td>prefer</td>
<td>no</td>
<td>no</td>
<td>works</td>
</tr>
<tr>
<td>require</td>
<td>yes</td>
<td>no</td>
<td>works</td>
</tr>
<tr>
<td>verify-ca</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
</tr>
<tr>
<td>verify-full</td>
<td>yes</td>
<td>yes</td>
<td>works</td>
</tr>
</tbody>
</table>
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`

```plaintext
.. 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
Authentication
Authentication

• Make sure it's the correct user
• And that they can prove it
A step back

• Authorization and roles
• I know I said I wouldn't...
Superuser

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

<table>
<thead>
<tr>
<th>Type</th>
<th>User</th>
<th>Role</th>
<th>Address</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>all</td>
<td>all</td>
<td></td>
<td>peer</td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>all</td>
<td>127.0.0.1/32</td>
<td>md5</td>
</tr>
<tr>
<td>hostnossl</td>
<td>webdb</td>
<td>webuser</td>
<td>10.1.1.0/30</td>
<td>md5</td>
</tr>
<tr>
<td>hostssl</td>
<td>all</td>
<td>+admin</td>
<td>192.168.0.0/24</td>
<td>gss</td>
</tr>
</tbody>
</table>

- 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

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

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

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

- Map client certificate to login
  - Uses **CN** attribute
- Any certificate "engine" supported by OpenSSL
  - Normally uses PEM encoded files
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 0.0.0.0/0 cert map=cert

- **pg_ident.conf**

  local root postgres
  ..
  cert /^cn=(.*)$/ \1
Secure PostgreSQL Deployment
Secure PostgreSQL Deployment

• Determine your requirements
• Determine your trust levels
• Determine your attach surface
• Determine your threat vectors
Secure PostgreSQL Deployment

• Deploy correct countermeasures
  • "Checkbox featuring" is useless
• Lock all doors
  • E.g. why encrypt if disks are insecure
  • 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!
Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander