Greetings!

- Christophe Pettus
- CEO, PostgreSQL Experts, Inc.
- thebuild.com — personal blog.
- pgexperts.com — company website.
- Twitter @Xof
- christophe.pettus@pgexperts.com
We’re Here To Do The Impossible.

• “Security” is not a single topic or a single practice.
• Essentially everything you do has security implications.
• Perfect security is impossible.
• All life is a tradeoff, followed by certain death.
OLD MAN YELLS AT CLOUD

Old man Abraham, Surname granddaughter...
A Crazy Man Is Yelling At Me.

• Every installation makes tradeoffs on utility, convenience, and security.

• Almost no one does everything we’ll do here. That’s (probably) OK.

• Just make sure you understand what the risks are, and how to mitigate them.
The Stack.

• Host system.
• PostgreSQL itself.
  • Access to the database server.
• The data in PostgreSQL.
  • Encryption, permissions, etc.
• The application.
If the database server host is compromised, nothing else matters.

Assume that local privilege escalation will always be a thing.

Always assume a local user can get root.

… because they probably can.
DIRTY COW
Minimize Attack Surface.

• Always put your database server behind a firewall / VPC.

• Never expose port 5432 to the public internet.

• On AWS, everything is the public internet.
No Direct SSH.

• Do not allow direct public logins via SSH to the database host. Require a hop through a specific bastion host.

• Restrict access to the bastion host by VPN or IP; do not simply trust bare SSH (even on a nonstandard port).

• Everyone tries 2222 now. C’mon.
• Don’t run unnecessary services on your database host.

• No application server, IRC server, mail server, giant mysterious Java VM the last sysadmin installed…

• Run nmap against it and see what’s open.
iptables is your friend.

- Or whatever local firewall you have.
- Restrict access just to expected servers.
- Don’t rely on just pg_hba.conf.
- Especially important in a cloud hosting environment.
And Do The Basics.

- For system administration, use specific users and sudo; never, ever allow root logins.
- Use a password manager. Always always always.
- For critical passwords, use split passwords with dual custody.
Keep up to date!

- Always subscribe to the pgsql-announce list.
- Always immediately apply any security-related updates.
- Also subscribe to the appropriate security list for your platform.
- Keep up to date with patches, already!
Apply Patches Promptly.

- Make it someone’s job.
- Make sure they do it.
- Never, ever allow a critical security patch to go unheeded.
- Ever ever ever.
In a perfect world...

- Use multi-factor authentication for all logins (VPN, host, etc.).
- Use LDAP for all logins (so that credentials can be revoked globally).
- Require password rotation.
- At an absolute minimum, never reuse passwords.
Google
“codespaces”
The Glass House

- Make sure your machines are properly secured in the data center.
- This means real security (access control, video, mantrap, biometrics) on your server room.
- Make sure your cloud provider provides this for the cloud they are providing to you!
pg_hba.conf
<table>
<thead>
<tr>
<th># TYPE</th>
<th>DATABASE</th>
<th>USER</th>
<th>ADDRESS</th>
<th>METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>all</td>
<td>all</td>
<td></td>
<td>trust</td>
</tr>
<tr>
<td>TYPE</td>
<td>DATABASE</td>
<td>USER</td>
<td>ADDRESS</td>
<td>METHOD</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>local</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>trust</td>
</tr>
</tbody>
</table>
Securing the Database Instance.

- There is no such thing as “trust” mode authentication. Forget it ever existed.
- Always require specific users, even superusers.
- Do not use the postgres Unix or database user. Require specific users.
- LDAP is your “friend,” here.
But what about “postgres”?

- Create a nasty password for it, keep it in dual custody.
- Never use it except in dire emergency.
- Don’t allow non-local logins for it (or any other superuser).
- Don’t use it for routine system administration tasks.
listen_address

• Set it to the specific addresses that you know are on the right networks.

• listen_address = ‘*’ is for the brave.

• In a cloud environment, you can’t always guarantee that all interfaces are within a VPC.
• Use LDAP to manage credentials.
• Every user and role should have its own PostgreSQL role.
• Only grant the permissions that role actually needs.
• A data analyst does not need to drop tables.
Passwords.

• If not using LDAP, PostgreSQL passwords must be singletons.
• MD5 passwords might as well be cleartext at this point.
• Don’t reuse PostgreSQL user passwords anywhere else.
• Make them horrible and long.
• Most common bad habit: the singleton web user than can do anything.

• This is made worse by some frameworks’ migration system.

• Fight it. Only give app roles the minimum that they need to work.

• Lock it down to app server IPs.
Connections.

- Require SSL and CA certificates.
- Especially in cloud environments.
- Anything less runs the risk of MitM attacks.
Data Security.

- Every database has sensitive information.
- Just customer and order info is sensitive.
- Some things are really sensitive.
  - Credit cards, health records, utility bills…
- Essential to protect it against theft.
“We’ll Just Park Here.”

- “No problem! We’ve layered luks on top of lvm on top of EBS, and we’re all set!“
- No.
- Full disk encryption is useless.
- Let me say that again.
FULL DISK ENCRYPTION IS USELESS.
FDE protects against...

• … theft of the media.
• That’s it.
• That is about 0.00000002% of the actual intrusions that you have to worry about.
• Easy rule: If psql can read it in cleartext, it’s not secure.
• (It’s a great idea for laptops, of course.)
That Being Said.

- Sometimes, regulations or contracts require full-disk encryption.
- Ugh. Fine.
- Make sure your key management is safe.
- Don’t bake keys into startup scripts, etc.
Per-Column Encryption.

- Always encrypt specific columns, not entire database or disk.
- Better performance, higher security.
- Key management is a pain.
- Automatic restart in a high-security environment is essentially impossible.
- Assume a human will be in the loop.
Per-Column Techniques.

- Encrypt each column as TEXT or bytea.
- Good for small items: credit cards, etc.
- Create a JSON blob, encrypt that, store it as bytea.
- More complex things, like medical records.
Good Crypto Hygiene.

- Use a well-known secure algorithm (AES256 is considered the standard).
- **Never** roll your own crypto.
- Do not bake keys into code or store them in repositories.
Indexing.

- You often have to store a partial version, or hash, of a value for indexing purposes.
- Example: CSRs may need to look up an order by credit card number.
- There’s nothing wrong with this, BUT:
Be careful with hashes!

- It's very easy to reverse some hashes, especially if you have partial data!
- Store the minimum necessary.
- Use a strong hash, like SHA-256.
So, how about pgcrypto?

• pgcrypto is a /contrib module that contains cryptography functions.

• Why not use it to encrypt the data?

• I mean, it’s just sitting there, right?
INSERT INTO super_secret_table(card)
VALUES(
    pgp_sym_encrypt('4111111111111111',
    'mysuperpassword'));
INSERT INTO super_secret_table(card)
VALUES(pgp_sym_encrypt('4111111111111111', 'mysuperpassword'))
Not so great.

- Be careful about what you expose in text logs.
- That “diagnostic” pgbadger run with `log_min_statement_duration = 0`?
- Always do the encryption in the application, not in the database.
Log Everything!

• Connections, disconnections, DML changes.
• Make sure logs are kept secure and cannot be tampered with (rsyslog, etc.)
• Make sure that the log record can be traced back to an individual person.
• Log *all* activity by directly-connecting users (as opposed to the application).
BUT!

- Make sure you are not logging sensitive information in cleartext!
- This is another good reason to encrypt in the application, not in the database.
Restrict the Data.

- … don’t give every developer production system access.
- … identify and qualify the system administrators who need global system access.
- … scrub data that comes out of production for development testing.
Backup Security.

• Be sure your backups are as secure as your primary database.
• A recent backup is just as good as your production system for a data theft.
• If using a shared cloud store like S3, make sure contents are properly encrypted and private.
Row-Level Security.

- Restricts access to data by row, rather than just by database object.
- Conceptually, a “mandatory view” applied based on access controls.
- Allows removal of sensitive columns, multi-tenancy in a table, etc.
Application Security.

• After all that, this is not where most breaches happen.
• Most breaches are either application breaches or malware-infected clients.
• POS tills, compromised user workstations.
Application Basics.

• Always use proper parameter substitution in your library!

• Never build SQL by text substitution unless it is absolutely necessary (for example, variable table names).

• All user input is hostile and wants to kill you all the time.
API Hygiene

- Always require TLS 1.2 for all remote APIs.
- For dedicated clients (mobile apps, etc.) use proper certificate management.
- Make API keys long, unique, and random.
- Log everything.
Prepare for War.

- Detect unusual access patterns and take action.
- Blocking, rate-limiting, admin alerts, etc.
- Users will generally share passwords across systems.
- Use Captchas to reduce automated attack risks.
Application Testing.

- Make security testing a critical part of testing.
- Always write tests that deliberately try to get around security controls.
- Get new engineers to try to hack your system, and praise them highly if they do.
Basic Infosec.

• Run appropriate malware-detecting email services.
• Use all of the OS vendor’s anti-virus tools.
• Follow @SwiftOnSecurity.
Trust, but Verify.

- Hire external penetration testing firms. Encourage developers to poke at security.
- Hire security audit companies that actually understand security, not just run pen test scripts.
This actually happened.

- “We need you to disable your firewall.”
- “Um, why?”
- “Our penetration test script is failing because the firewall won’t let it through.”
- “This… sounds kind of like what a firewall is supposed to do, to me.”
By now, you are probably...
We’re doomed.

• Data security is a lot of work.
• You will never be perfectly secure.
• Even the most secure companies get intrusions.
• Life is full of pain and despair.
Have hope!

• Do as much “set it and forget it” security as possible.

• Do regular audits and destruction tests (great things for new engineers to do).

• Be sure that the company, from the top, takes security seriously.
Life is full of tough choices.

• You will always trade off some security for convenience.

• But don’t get complacent and have convenience become the most important thing.

• Make security one of the things the organization is proud of!
Questions?

Christophe Pettus
@xof

thebuild.com
pgexperts.com
Thank you!

Christophe Pettus
@xof

thebuild.com
pgexperts.com