Database systems are rich with attack vectors to exploit. This presentation explores the many potential PostgreSQL external vulnerabilities and shows how they can be secured. 

*Includes concepts from Magnus Hagander*
Attack Vectors

https://www.flickr.com/photos/twalmsley/
External Attack Vectors

- 'Trust' security
- Passwords / authentication theft
- Network snooping
- Network pass-through spoofing
- Server / backup theft
- Administrator access
Internal Attack Vectors (Not Covered)

- Database object permissions
- SQL injection attacks
- Application vulnerability
- Operating system compromise
Avoid “Trust” Security in `pg_hba.conf`

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DATABASE</th>
<th>USER</th>
<th>CIDR-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># &quot;local&quot; is for Unix domain socket connections only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>all</td>
<td>127.0.0.1/32</td>
<td>trust</td>
</tr>
<tr>
<td># IPv4 local connections:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>all</td>
<td>all</td>
<td>::1/128</td>
<td>trust</td>
</tr>
<tr>
<td># IPv6 local connections:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Solution:** Use the `initdb -A` flag, i.e., you don’t want to see this:

```
WARNING: enabling "trust" authentication for local connections
You can change this by editing `pg_hba.conf` or using the -A option the next time you run initdb.
```
Password Snooping

Using ‘username’ in the MD5 string prevents the same password used by different users from appearing the same. It also adds some randomness to the md5 checksums.
MD5 Authentication Prevents Password Snooping

Connection request
Need md5, sent random salt
md5(md5(password+username) + salt)

md5(password+username)
md5(password+username)
md5(password+username)
md5(password+username)
md5(password+username)
MD5 Authentication Prevents Password Replay

`salt` is a random four-byte integer so millions of connection attempts might allow the reuse of an old authentication reply.
scram-sha-256, available in Postgres 10, eliminates less-secure MD5, and avoids the risk of duplicate salt values being replayed. SCRAM with channel binding, available in Postgres 13, allows authentication, similar to certificate authentication. scram-sha-256 is the default in Postgres 14.
SCRAM-SHA-256 Authentication

Connection request
Username, client nonce
Client nonce, server nonce, salt
Client nonce, server nonce, client proof

Need scram−sha−256

https://en.wikipedia.org/wiki/Salted_Challenge_Response_Authentication_Mechanism
Password Attacks

• Weak passwords
• Reuse of old passwords
• Brute-force password attacks

None of these vulnerabilities is prevented by Postgres directly, but external authentication methods, like LDAP, PAM, and SSPI, can prevent them. Some authentication methods are difficult to use with connection pooling.
Passwords changes are also vulnerable to snooping.
SSL Prevents Snooping By Encrypting Queries and Data

Queries and data encrypted by SSL

Database Client

AES256(SELECT * FROM customer);

AES256(Barr Bearings | $10230 | James Akel)

Queries and data encrypted by SSL

PostgreSQL

Database Server
Preventing Spoofing

https://www.flickr.com/photos/tomhickmore/
Localhost Spoofing While the Database Server Is Down

The server controls the choice of 'password' instead of 'md5'.

Uses a fake socket or binds to port 5432 while the real server is down. (/tmp is world-readable and 5432 is not a root-only port. libpq’s "requirepeer" helps here.)
Network Spoofing

Database Client

Connection Request
Need Plain Password
Password Sent

Fake PostgreSQL
Database
Server

Records passwords for later use with the real server

Without SSL ‘root’ certificates there is no way to know if the server you are connecting to is a legitimate server.
Network Spoofing Pass-Through

Database Client
- Connection Request
- Need Plain Password
- Password Sent
- Query
- Result

Fake PostgreSQL
- Database
- Server
Records passwords for later use with the real server. It can also capture queries, data, and inject its own queries.

PostgreSQL
- Database
- Server

OK

Query
- Result

Without SSL 'root' certificates there is no way to know if the server you are connecting to is a legitimate server.
SSL ‘Prefer’ Is Not Secure

Without SSL ‘root’ certificates there is no way to know if the server you are connecting to is a legitimate server.

**Fake PostgreSQL Database Server**

Records passwords for later use with the real server. It can also capture queries, data, and inject its own queries.
SSL 'Require' Is Not Secure From Spoofing

Without SSL 'root' certificates there is no way to know if the server you are connecting to is a legitimate server.

Fake PostgreSQL

Database

Server

Records passwords for later use with the real server. It can also capture queries, data, and inject its own queries.

PostgreSQL

Database

Server

Query

Result

SSL or Non-SSL

OK

Query

Result

Database

Client

Require SSL

OK SSL

SSL

Query

Result

Ok

Client

Database

PostgreSQL

Fake PostgreSQL

Database

Server

Without SSL 'root' certificates there is no way to know if the server you are connecting to is a legitimate server.
SSL 'Verify-CA' Is Secure From Spoofing

Database Client

Fake PostgreSQL
Database
Server

PostgreSQL
Database
Server

SSL verify–ca
Invalid certificate
(no CA signature)
Channel Binding

Certificates are sent to peers as part of the TLS handshake. Later the certificate hash is hashed with the password hash to prove the TLS peer knows the password hash. This is tls-server-end-point channel binding.

Connection request, need SSL
Need scram-sha-256
Channel binding req., username, client nonce
Client nonce, server nonce, salt
Client nonce, server nonce, client proof
Server proof

Database
Client

PostgreSQL
Database
Server

server.crt

sha256(password+salt)
sha256(password+salt)
sha256(password+salt)
sha256(password+salt)
SSL Certificates for Authentication

Request for SSL certificate

SSL certificate with CN

Database Client

root.crt

PostgreSQL Database Server

server.crt
Data Encryption To Avoid Data Theft

https://www.flickr.com/photos/debarshiray/
Disk Volume Encryption

This helps prevent stolen storage devices from being read, and helps with secure media destruction.

https://www.flickr.com/photos/icebrkr/
Encryption methods are decryptable (e.g., AES), while hashes are one-way (e.g., MD5). A one-way hash is best for data like passwords that only need to be checked for a match, rather than decrypted.
Where to Store the Key? On the Server

SELECT * FROM customer;
Barr Bearings | $10230 | James Akel
_Decrypted data_

Database
Client

PostgreSQL
Database
Server

Decrypted data
Store the Key on an Intermediate Server
Store the Key on the Client and Encrypt/Decrypt on the Server

SELECT decrypt(col, key) FROM customer;

Decrypted data
Encrypt/Decrypt on the Client

This prevents server administrators from viewing sensitive data.
Store the Key on a Client Hardware Token

This prevents problems caused by client hardware theft.
Conclusion

https://momjian.us/presentations

https://www.flickr.com/photos/stevensnodgrass/