Co-funded by
the European Union

MERiT

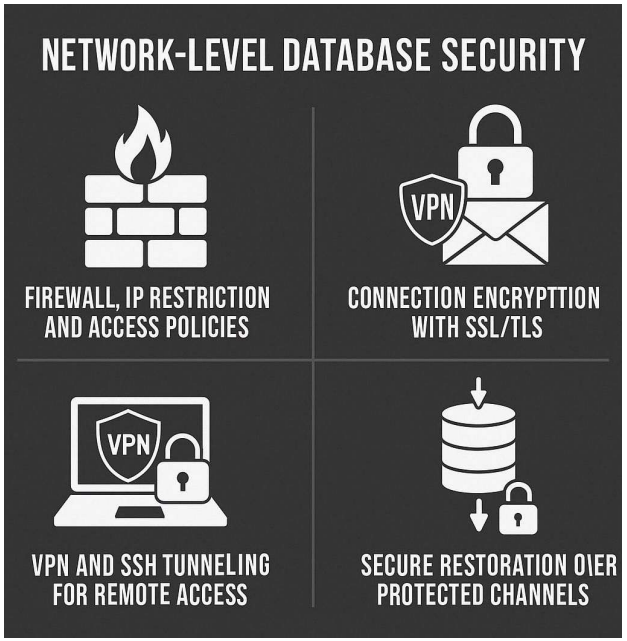
DataBase Security

Network-Level Database Security

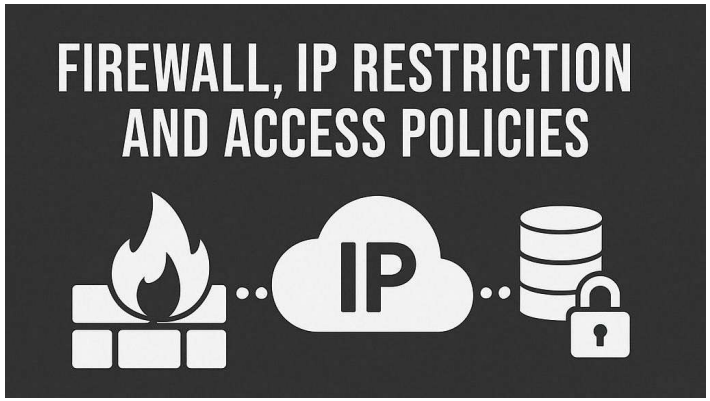
Network-Level Database Security

Network-level security mechanisms are essential to protect database systems from unauthorized access and interception during communication. The following practices strengthen the security posture of databases across network layers:

- Firewall, IP restriction and access policies
- Connection encryption with SSL/TLS
- VPN and SSH tunneling for remote access
- Secure restoration over protected channels



Firewall, IP Restriction and Access Policies



1. Firewall, IP Restriction, and Access Policies

Firewalls are the first line of defense, filtering incoming and outgoing traffic to database servers.

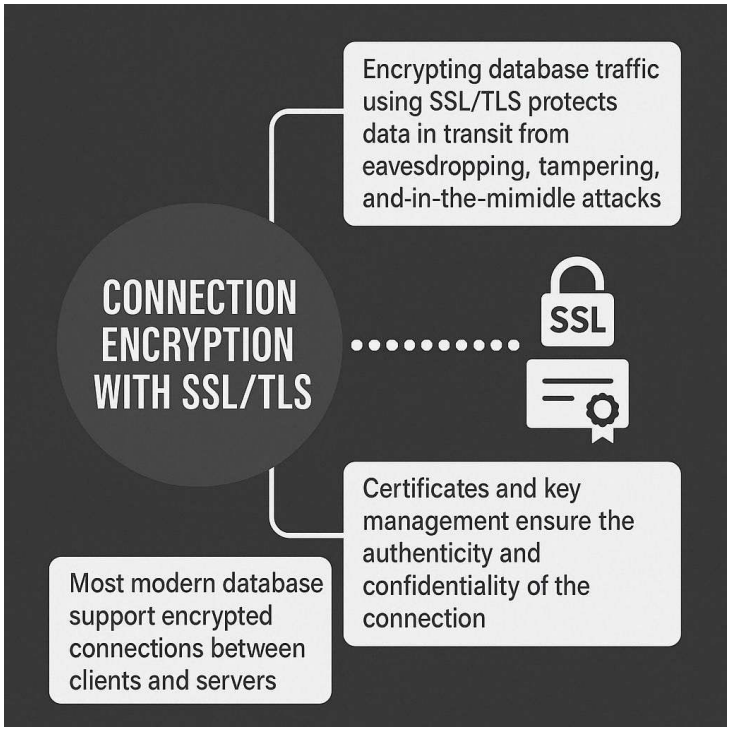
- **IP restriction** ensures that only authorized IP addresses can connect.
- **Access policies** define who can access which resources and under what conditions.

Together, they reduce exposure to external threats and unauthorized users.

Example – UFW Rules (Ubuntu)

```
sudo ufw allow from 192.168.1.0/24 to
any port 5432
sudo ufw deny from any to any port 5432
sudo ufw enable
ufw status
```

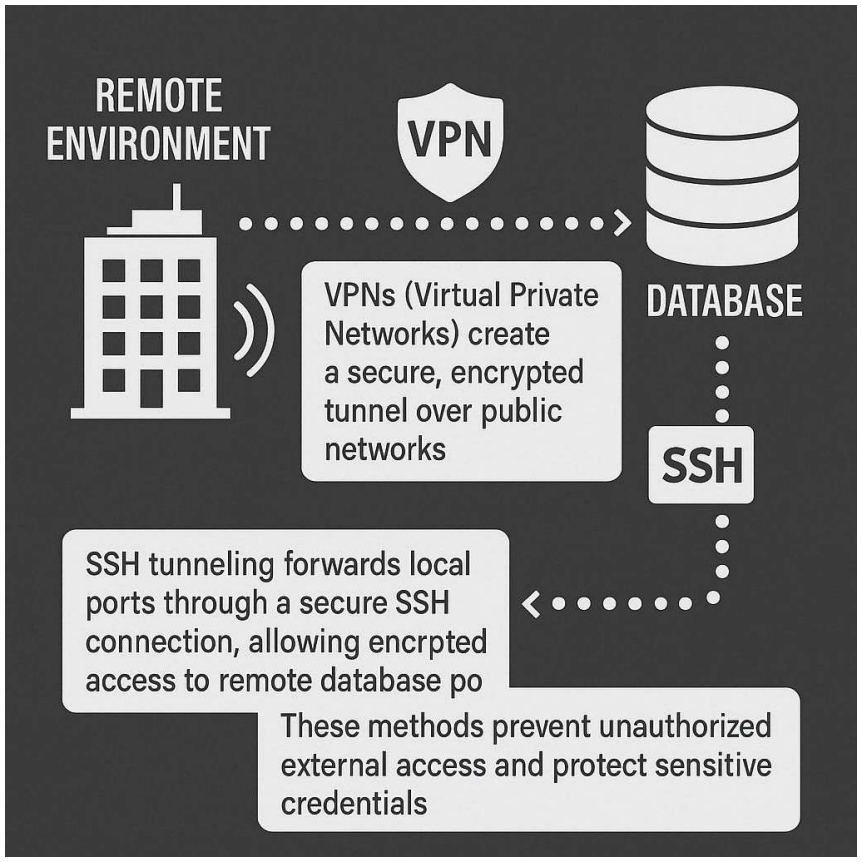
Connection Encryption with SSL/TLS



Encrypting database traffic using **SSL/TLS** protects data in transit from eavesdropping, tampering, and man-in-the-middle attacks.

- Most modern databases support encrypted connections between clients and servers.
- Certificates and key management ensure the authenticity and confidentiality of the connection.

VPN and SSH Tunneling



To securely access databases from remote environments:

- **VPNs (Virtual Private Networks)** create a secure, encrypted tunnel over public networks.
- **SSH tunneling** forwards local ports through a secure SSH connection, allowing encrypted access to remote database ports.

These methods prevent unauthorized external access and protect sensitive credentials.

Secure Restoration over Protected Channels

When backups are restored, sensitive information (such as user data, passwords, logs, and configurations) is being **moved and rewritten**—this is a vulnerable moment.
If the channel used to transfer the backup isn't protected, **attackers could intercept or manipulate** the data.

Use Encrypted Transfer Protocols

To avoid this, you should **always transmit backup files over encrypted channels**, such as:

- 🔒 **SFTP (Secure File Transfer Protocol)**: Adds encryption to FTP using SSH.
- 🔐 **HTTPS (Hypertext Transfer Protocol Secure)**: Encrypts backup file transfers through the web.
- 📦 **rsync over SSH**: Combines efficient file syncing with encryption.

These protocols **scramble the data** during transmission so that only trusted systems can read it.

What Happens Without Encryption?

Imagine restoring a backup over an unencrypted FTP connection. Anyone with access to the network could:

- **Read** confidential information in transit
- **Alter** the contents before they reach the server
- **Inject malicious files or code**

This could lead to data breaches, corrupted databases, or even system compromise.

Monitoring and Alerts

Database Access Monitoring & Protection

- **Log all connections** (successful & failed)
  Enables detection of brute-force attacks and unusual behavior.
- **Block suspicious IPs with Fail2Ban**
  Automatically bans IPs after repeated login failures using firewall rules.
- **Set up real-time alerts**
  Receive notifications for unauthorized access attempts via tools like **Alert manager or Elastalert**.
- **Monitor logs centrally**
  Use tools like **journalctl, rsyslog,** or the **ELK Stack** for centralized logging and analysis.

✅ **Summary of Tools:**

Purpose	Open Source Tools
Log Access	PostgreSQL logs, MySQL logs, journald,GoAccess
Auto Ban on Failures	Fail2Ban
Alerting	Prometheus, Elastalert, Alertmanager
Log Aggregation & Search	ELK Stack, Loki, Graylog

Best Practices for Secure Database Network Design

Best Practices for Secure Database Network Design

- **Segment the Network**: Place databases in isolated VLANs or subnets to limit exposure.
- **Use Jump Hosts**: Restrict direct access and enforce authentication via controlled intermediate servers.
- **Implement Least Privilege**: Configure firewalls and access policies for minimum required access only.
- **Apply Microsegmentation**: Use security groups or policies to isolate services even within secure zones.
- **Audit Configurations Regularly**: Review rulesets, ACLs, and routing for misconfigurations.

Real-World Threats and Mitigations

Threat	Mitigation
Unauthorized remote access	VPN, SSH tunneling, IP whitelisting
Man-in-the-middle attacks	SSL/TLS encryption, certificate validation
Port scanning and fingerprinting	Firewall rules, port knocking, IDS integration
Lateral movement after intrusion	Network segmentation, access logging
Intercepted backup restoration	Encrypted protocols (SFTP, HTTPS, rsync+SSH)

Common Mistakes in Network-Level DB Security

Metrics to Track in Network DB Security

- **Blocked connection attempts** (e.g., by Fail2Ban or firewall rules)
- **Unauthorized access alerts** (triggered by login failures or unusual IPs)
- **SSL/TLS certificate expiration** monitoring
- **Backup restore logs** with timestamps and hashes
- **VPN/SSH tunnel usage logs** (active sessions, duration, source IPs)

📌 *Use centralized dashboards (ELK, Grafana, Zabbix,Zenoss) for real-time visibility.*

Summary

🔐 **Network-Level Database Security – Summary**

To protect databases from external threats, implement multiple layers of network-based defense:

- **Firewall & IP Restriction**: Limit access to trusted IPs only and define access control policies.
- **SSL/TLS Encryption**: Secure data in transit to prevent eavesdropping and tampering.
- **VPN & SSH Tunneling**: Enable encrypted remote access over public networks.
- **Secure Backup Restoration**: Use encrypted protocols (SFTP, HTTPS) to avoid exposing sensitive data.
- **Access Monitoring & Blocking**: Log all connection attempts, use tools like **Fail2Ban**, and monitor logs with **GoAccess**, journalctl, or the **ELK Stack`.