# Beyond Trust

## PostgreSQL Client Authentication

Christoph Mönch-Tegeder

2ndQuadrant
http://www.2ndquadrant.com/

2017-06-30

2ndQuadrant®
PostgreSQL

# Teil I

## Authentifizierung

# Warum Authentifizierung?

- *Kompliziert*
- *Die IP-Adressen sind geheim*
- *Wir werden schon nicht angegriffen*

# Datenreichtum

Malwarebytes LABS

CYBERCRIME | HACKING

## COMELEC breach data released online, fully searchable

Posted April 21, 2016 by Christopher Boyd

On March 27, the COMELEC (Philippines' Commission on Elections) website was defaced and data on up to 55 million registered voters in the Philippines was compromised.

https://blog.malwarebytes.com/cybercrime/2016/04/comelec-breach-data-released-online-fully-searchable/

# Datenreichtum (2)


https://www.theregister.co.uk/2016/04/25/mexico_voter_data_breach/

# Datenreichtum für mich, für Dich, für Alle!


`https://www.databreaches.net/191-million-voters-personal-info-exposed-by-misconfigured-database/`

2ndQuadrant® PostgreSQL

# Neue Geschäftsmodelle


10 JAN 17

## Extortionists Wipe Thousands of Databases, Victims Who Pay Up Get Stiffed

https://krebsonsecurity.com/2017/01/extortionists-wipe-thousands-of-databases-victims-who-pay-up-get-stiffed/

# Boomendes Business

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE SECURITY TRANSFORMATION DEVOPS BUSINESS PERSONAL TECH SCIENC

Security

## MongoDB ransom attacks soar, body count hits 27,000 in hours

Aussie comms watchdog reporting exposed databases.

9 Jan 2017 at 02:26, Darren Pauli

https://www.theregister.co.uk/2017/01/09/mongodb/

2ndQuadrant®
PostgreSQL

# Nicht nur die eine Datenbank


https://www.bleepingcomputer.com/news/security/mongodb-hijackers-move-on-to-elasticsearch-servers/

2ndQuadrant® PostgreSQL

# Diversifizierung


https://www.bleepingcomputer.com/news/security/hadoop-servers-expose-over-5-petabytes-of-data/

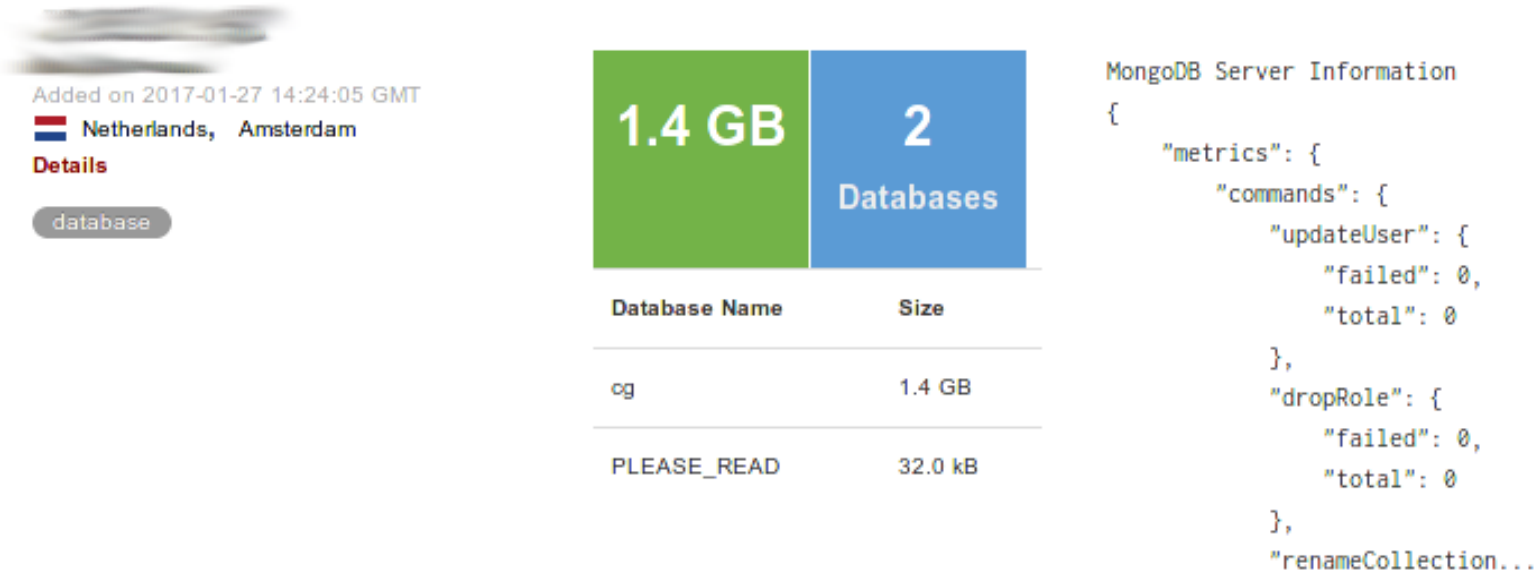
# Warum Authentifizierung?

- Open Data ist schön, aber nicht so
- Nicht der Support, den wir zahlen wollten

# Warum Authentifizierung?

- Open Data ist schön, aber nicht so
- Nicht der Support, den wir zahlen wollten
- Compliance
- Wer sagt es dem Chef?

# Verstecken hilft nicht




- `https://www.shodan.io`
- `https://www.zoomeye.org`

2ndQuadrant® PostgreSQL

# Also. . . , Datenbank absichern!

- Network Security und Firewalls
  - *drinnen* und *draußen*
  - PostgreSQL nur auf `localhost` per default
- `GRANT`/`REVOKE` Access Control
  - Applikation sollte nicht den Superuser (`postgres`) benutzen
- Schwachstellen in der Applikation vermeiden
- alle Benutzer authentifizieren

# Definitionen

- **Identifikation** Behauptung einer Identität
  - "Ich bin `user42`"
- **Authentifizierung** Beweis der Identität
  - Wissen, Besitz, Eigenschaften
- **Authorisierung** erlaubt Zugriff auf Objekte
  - `GRANT SELECT ON tbl TO user42`, RLS, ...
- **Audit Log** wer hat was wann getan?
  - `log_connections`, `pgaudit`

# Geheime Information sicher handhaben

- Shared Keys, Asymmetrische Schlüssel
  - Code, Konfiguration, ...
- Wie Login-Information ändern?
- Keys nicht auf Github ablegen!
- To Backup or Not To Backup?
  - Wie werden Backups geschützt?
  - Wie werden Secrets wiederhergestellt?
- Hardware Security Module (HSM)
  - Was passiert, wenn HSM verloren oder zerstört?

# Sichere Authentifizierung

- Passive Angriffe
  - Lauschangriff: Passwörter vom Netz lesen
  - und alles andere auch
- Aktive Angriffe
  - Man in the Middle (MitM)
  - kann Traffic modifizieren
- *There is no safe authentication unless you authenticate whom you're authenticating against first.* ***(Martin Seeger)***

# PostgreSQL Authentifizierung

- konfigurierbar, gut dokumentiert
- flexibel: bis zu 14 Methoden
- Datenbank, User, Source und Verbindungstyp
  - `pg_hba.conf`, Host Based Authentication
  - First Match Wins
- Konfiguration dokumentieren – und mit Realität abgleichen
- Rollen müssen immer (auch) in PostgreSQL angelegt sein
  - nur Authentifizierung kann extern sein

# Host Based Authentication Konfiguration

```
# IPv4 local connections:
host    all     all     127.0.0.1/32    md5
```

- Art der Verbindung
  - `local` nur auf Unix-Plattformen
- Datenbank (`all`, `@file`, `replication`, `sameuser`, `samerole`)
- User (`all`, `+group`, `@file`)
- Nicht-lokale Verbindung: Client-Adresse (Netzmaske!)
- Authentifzierungsmethode und Optionen

# Identifikations-Mapping

- `pg_ident.conf` externer Benutzername zu PostgreSQL Rolle
- Mappings addressiert durch `map`-Parameter in `pg_hba.conf`
- Reguläre Ausdrücke möglich

```
# MAPNAME    SYSTEM-USERNAME          PG-USERNAME
sslmap       ”Test User”              ssluser
krbmap       /^([^@]+)@MY.KRB5.REALM  \1
```

# Teil II

# Authentifizierungs-Methoden

# Trust – there is none

- keine Authentifizierung – nur Identifikation
- im Allgemeinen **nicht** für Server
- manchmal ok für Tests, embedded Systems
- gelegentlich in der initialen `pg_hba.conf`

# Ident – Remote User

- kontaktiert Client-IP, fragt nach Unix-Usernamen
- *Identification Protocol* (RFC1413) – eher selten
- zusätzliche TCP-Verbindung
- vertraut auf Sicherheit des Client-Hosts

> *The Identification Protocol is not intended as an authorization or access control protocol.*

**RFC1413** *https://www.rfc-editor.org/rfc/rfc1413.txt*

# Peer – Unix Credentials

- nur für lokale (`local`) Verbindungen
- System-Benutzername am Unix-Socket ausgelesen
- gut für lokale Administration
- Sicherheit wie beim Betriebssystem
- keine Unterstützung für Remote-Verbindungen

# Reject – Blacklisting

- verbietet Zugriff
- einen verbieten, `all` erlauben
- temporäres Sperren der Verbindungen während Maintenance
- *nicht-Authentifizierungs-Methode*

# Password – Klar wie Text

- Passwort-Authentifizierung
- Hashes in PostgreSQL gespeichert – `pg_authid`
- (`password_encryption`)
- Passwörter im Klartext auf dem Netz
- **nicht mehr benutzen**
- stattdessen: `md5`, `scram-sha-256` (PostgreSQL 10)

# MD5 – Passwort-Hash

- Passwort-Authentifizierung reloaded
- Hashes in PostgreSQL gespeichert – `pg_authid`
- `'md5' + md5(md5(password + username) + salt)`
- Hash aus Datenbank ausreichend für Authentifizierung
- Replay: 50% nach 2 Milliarden Verbindungen (4 Byte Salt)
- Birthday Attack: 50% nach ca. 77000 Verbindungen
- MD5 Hash sehr schwach – Compliance-Probleme
- Server wird nicht authentifiziert

# SCRAM – Salted Challenge Response Authentication Mechanism

- Passwort-Authentifizierung noch besser
- SCRAM: RFC5802, SCRAM-SHA-256: RFC7677
- Simple Authentication and Security Layer SASL: RFC4422
- ab PostgreSQL 10
- `password_encryption = scram-sha-256`
- Hash-Aufwand einstellbar
- benötigt Client-Unterstützung
- noch keine Server-Verifikation

# LDAP – Lightweight Directory Access Protocol

- authentifiziert gegen LDAP-Server (nicht enthalten)
- *simple*: Credentials authentifizieren *(bind)* am LDAP-Server
- *search+bind*: sucht User/Passwort im LDAP-Baum
- TLS-Verbindung zum LDAP-Server möglich
  - nur STARTTLS (RFC 5413) – nicht LDAP-over-SSL
  - manche Server unterstützen STARTTLS nicht
- Passwort im Klartext, Server wird nicht authentifiziert
- TLS für die PostgreSQL-Verbindung empfohlen

# PAM – Pluggable Authentication Modules

- *Module* (Plugins) übernehmen Authentifizierung (und mehr)
- Konfiguration in `/etc/pam.d/`
- kein Zugriff auf `/etc/shadow` (für PostgreSQL)
- Lockout nach falschem Login, alternative Backends
- Passwort im Klartext, Server wird nicht authentifiziert
- TLS für die PostgreSQL-Verbindung empfohlen

# GSS – Generic Security Services API

- Integration in KerberosV (Infrastruktur nicht enthalten)
  - Realm, Principal, Ticket, TGT
- Client authentifiziert gegen Key Distribution Center (KDC)
- *Service Server* PostgreSQL bekannt beim KDC
- Zeitsynchronisation für alle Teilnehmer nötig
- periodisch neue *Tickets*
- keine Passwörter im Klartext, alle Teilnehmer authentifiziert

# GSS (2)

- KDC Passwort Backend: lokale Datenbank, LDAP, ...
- Client-KDC Auth: Passwort (oft), PKINIT (Public Key), ...
- *keytab* Datei für nicht-interaktive Prozesse: **Stored Secrets**
- Mapping der Kerberos Principals auf PostgreSQL Rollen

```
# MAPNAME    SYSTEM-USERNAME           PG-USERNAME
krbmap       /^([^@]+)@MY.KRB5.REALM \1
```

# GSS (3)

- Einfache Konfiguration für PostgreSQL
- `postgresql.conf`

```
krb_server_keyfile = 'krb5.keytab'
```

- `pg_hba.conf`

```
host all all 10.0.1.0/24 gss krb_realm=MY.KRB5.REALM
```

- DSN: `krbsrvname=postgres`

# Cert – TLS Client Zertifikate

- nur für Verbindungen mit TLS (`hostssl`)
- Client verifiziert Server, Server verifiziert Client (per Zertifikat)
- benötigt: PKI (nicht mitgeliefert)
- manuelle Zertifikatserstellung möglich (wenige Hosts)
- für nicht-triviale Setups: CA-Software

# Cert (2)

- Minimale Server-Konfiguration

```
ssl = on
ssl_cert_file = 'server_cert.pem'
ssl_key_file = 'server_cert.key'
ssl_ca_file = 'user_ca.pem'
```

- Empfehlung: Server-CA nicht für User-Zertifikate benutzen
- keine externe (kommerzielle) CA für User benötigt
- DSN: `sslcert=cert.pem sslkey=cert.key`

# Cert (3)

- Private Keys sind **Stored Secrets**
- Client-Zertifikate laufen ab: einfach austauschen
- Server-Zertifikat und CAs laufen ab: PostgreSQL Restart
- Client-Zertifikat widerrufen: Certificate Revocation List (CRL)
  - `ssl_crl_file`
  - nur mit Restart in PostgreSQL $< 10$
- Client-Zertifikat Common Name (CN) zu PostgreSQL Rolle:

```
# MAPNAME     SYSTEM-USERNAME          PG-USERNAME
sslmap        ''Test User''            ssluser
```

2ndQuadrant® PostgreSQL

# Weitere Methoden

- `radius` Remote Authentication Dial-In User Service
  - RADIUS heute selten, außer bei Telcos
- `sspi` Security Support Provider Interface
  - Windows Single Sign On, Kerberos mit NTLM Fallback
- `bsdauth` OpenBSD Authentication Framework
  - nur für OpenBSD, Prinzip wie PAM

# Teil III

# Praktische Hinweise

# Hinweise für TLS

- Cipherlist muss eingeschränkt werden, `ssl_ciphers`
- Beispiel (vorsicht mit alten Clients):

  `TLSv1.2+HIGH+EECDH:TLSv1.2+HIGH+DHE:!eNULL@STRENGTH`

- Diffie-Hellman-Parameter selbst generieren
- aktuelle PostgreSQL kann ECC (ECDSA)
- Verbindungsaufbau langsam
- mäßiger Overhead auf laufender Verbindung

# Connection Pooler

- Clients authentifizieren am Pooler
- Pooler authentifiziert an PostgreSQL
- Weiterleitung der Authentifizierung: nur mit Klartext-Passwort
- TLS wird am Pooler terminiert

# Zusammenfassung

- Auch Datenbanken müssen abgesichert werden
- Clients (und Server) authentifizieren
- Passwörter (`md5`/`scram-sha-256`) sind das Minimum
- Verbindungen verschlüsseln – TLS
- Nicht vergessen: Zertifikate verifizieren

Fragen?