Beyond Trust

PostgreSQL Client Authentication

Christoph Mönch-Tegeder

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

2017-02-05

2ndQuadrant®
PostgreSQL

Part I

Authentication

Why Authentication?

- *Too complicated*
- *Nobody knows my database*
- *Nobody will attack us*

Rich Data

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/

Rich Data (2)

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

Racketeering

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/

Racketeering (2)

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

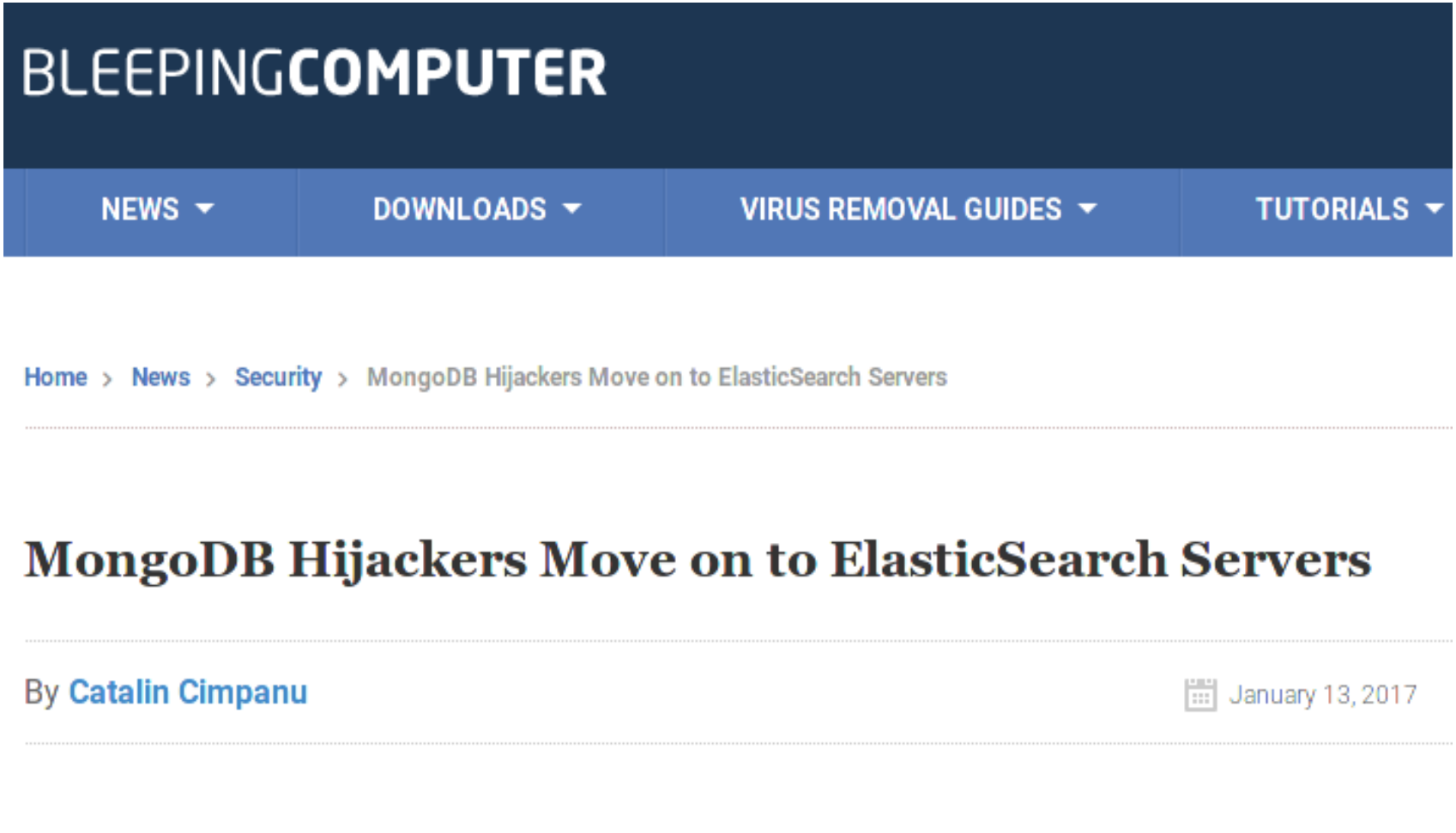
Aussie comms watchdog reporting exposed databases.

9 Jan 2017 at 02:26, Darren Pauli

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

2ndQuadrant®
PostgreSQL

It's not only that database



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

2ndQuadrant® PostgreSQL

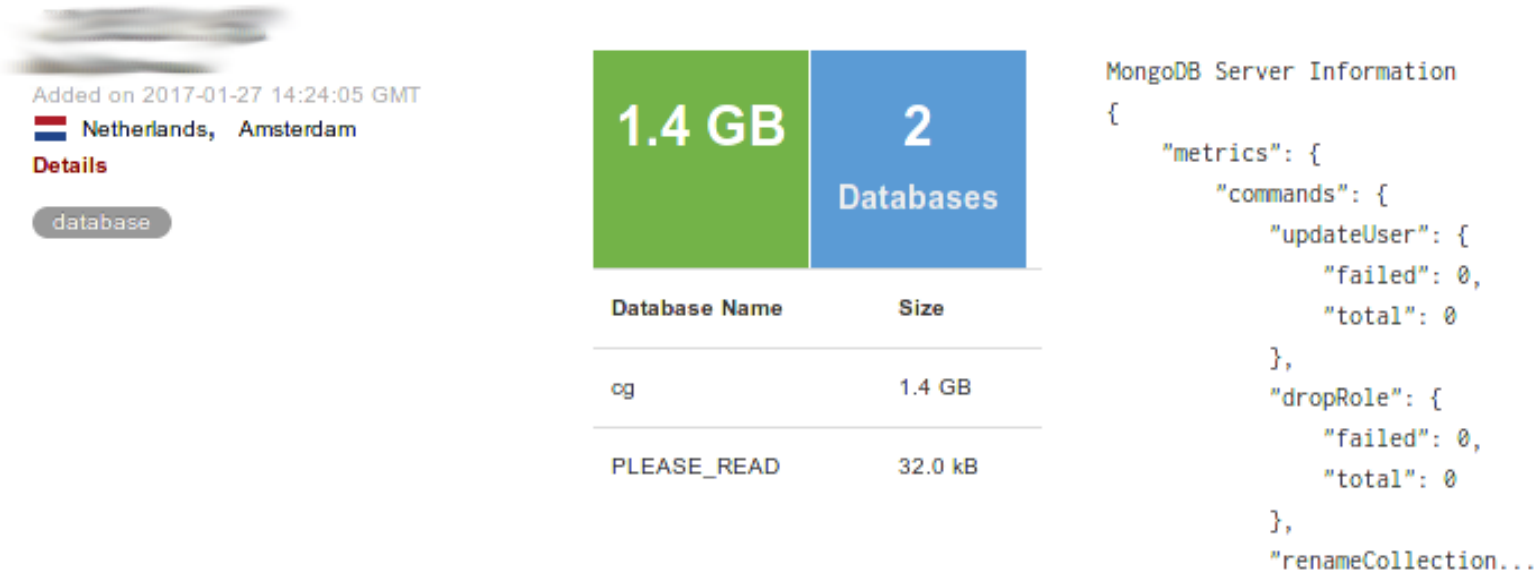
Why Authentication?

- Not the Open Data we wanted!
- Not the kind of support we want to pay!

Why Authentication?

- Not the Open Data we wanted!
- Not the kind of support we want to pay!
- Regulations
- Nobody wants to explain these incidents

You cannot hide



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

2ndQuadrant®
PostgreSQL

So. . . Secure Your Database!

- Network Security and Firewalls
  - keep unauthorized users out
  - PostgreSQL listens on `localhost` only by default
- Fine Grained Access Control
  - do not run your app as `postgres`
- Secure your application
- Authenticate legitimate users

Some Definitions

- **Identification** declaration of identity
  - "I am `user42`"
- **Authentication** proof of identity
  - what (only) you know, have, are
- **Authorization** allows actions on objects
  - `GRANT SELECT ON tbl TO user42`, RLS, ...
- **Audit Log** who did what and when
  - `log_connections`, `pgaudit`

Secret Handling

- Shared Keys, Asymmetric Secrets
  - hardcoded, configuration files, ...
- How to change secrets?
- Do not put secrets on Github!
- To Backup or Not To Backup?
  - how to guard backups? how to restore secrets?
- Hardware Security Module (HSM)
  - what if lost/destroyed?

Authentication Security

- Passive Attacks
  - sniffing authentication info off the net
  - and all other traffic
- Active Attacks
  - Man in the Middle (MitM)
  - may modify traffic
- *There is no safe authentication unless you authenticate whom you're authenticating against first.* **(Martin Seeger)**

PostgreSQL Authentication

- Highly configurable, well documented
- Flexible: up to 13 methods
- Per database, user, source and connection type
  - `pg_hba.conf`, Host Based Authentication
  - checked from top to first match
- Document your configuration, check if it still matches reality
- Users always have to exist in PostgreSQL
  - only authentication can be handled externally

Host Based Authentication Configuration

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

- type of connection
  - `local` on unix-like platforms only
- database (`all`, `@file`, `replication`, `sameuser`, `samerole`)
- user (`all`, `+group`, `@file`)
- non-local: source network
- authentication method and options

Identification mapping

- `pg_ident.conf` allows mapping of external usernames to PostgreSQL usernames
- mappings selected by `map` parameter in `pg_hba.conf`
- regular expression support

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

Part II

Authentication Mechanisms

Trust - there is none

- no authentication - just identification
- do **not** use on servers
- for testing, some embedded systems
- sometimes used in initial `pg_hba.conf`

Ident - ask remote

- contact source host, ask for local user name
- uses *Identification Protocol* (RFC1413) - less common today
- additional TCP connection
- relies on security of source host

> *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

- on local connections only
- get system user name from unix socket
- great for local administration
- as secure as the host OS
- limited use - no remote support, inflexible

Reject - Blacklisting

- explicitly reject access
- reject one, allow `all`
- temporary block during maintenance
- *non-authentication* method

Password - clear as text

- password authentication
- hashes stored in PostgreSQL – `pg_authid`
- sends password in clear
- do **not** use
- use `md5` instead

MD5 - hashed password

- password authentication reloaded
- hashes stored in PostgreSQL – `pg_authid`
- on the wire:
  `'md5' + md5(md5(password + username) + salt)`
- replay: 50% chance after 2 billion connections (4 byte salt)
- MD5 hash considered broken – regulatory problems
- does not authenticate server

LDAP - Lightweight Directory Access Protocol

- authenticates against LDAP backend (not included)
- simple mode: use credentials to *bind* (authenticate) to the LDAP server
- search+bind mode: bind to LDAP server and search for user with given credentials
- can use TLS connection to the LDAP server
  - STARTTLS only (as per RFC 5413) - no LDAP over SSL
  - some servers do not support STARTTLS
- cleartext password, does not authenticate server
- TLS for PostgreSQL connection recommended

PAM – Pluggable Authentication Modules

- *modules* (plugins) handle authentication (and more)
- configuration in `/etc/pam.d/` or similar
- cannot access `/etc/shadow` (when running in PostgreSQL)
- can be used to lock out clients or accounts after failed logins
- cleartext password, does not authenticate server
- TLS for PostgreSQL connection recommended

GSS - Generic Security Services API

- uses KerberosV (Kerberos infrastructure not included)
  - realm, principal, ticket, TGT
- client authenticates Key Distribution Center (KDC)
- *Service Server* PostgreSQL must be known to KDC
- requires accurate time across all systems
- periodic *ticket* renewal
- no clear text passwords on the network, all entities authenticated

GSS (2)

- KDC password store: local databases, LDAP, ...
- Client-KDC authentication: password (most common), PKINIT (public key), ...
- keytab files for non-interactive processes: **stored secrets**
- mapping of Kerberos principals to PostgreSQL users

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

GSS (3)

- Simple configuration on the PostgreSQL side
- `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 Certificate

- only for TLS connections (`hostssl`)
- as the client verifies the server's certificate, the server verifies the client's certificate
- requires: PKI (not included)
- possible to create and sign certificates by hand
- for more than a few hosts, use real CA software

Cert (2)

- minimal server configuration

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

- recommended: do not re-use server CA for user certificates
- does not require external CA for users
- DSN: `sslcert=cert.pem sslkey=cert.key`

Cert (3)

- private keys are **stored secrets**
- client certificates expire: can be changed on the fly
- server certificates and CAs expire: change needs restart
- disable client certificate: Certificate Revocation List (CRL)
  - `ssl_crl_file`
  - requires restart in PostgreSQL < 10
- map certificate Common Name (CN) to PostgreSQL user

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

Other Mechanisms

- `radius` Remote Authentication Dial-In User Service
  - RADIUS uncommon, except for telco environments
- `sspi` Security Support Provider Interface
  - Windows Single Sign On, Kerberos with NTLM fallback
- `bsdauth` OpenBSD authentication framework
  - OpenBSD only, ideas like PAM

Part III

Considerations

Some Notes on TLS

- default cipher list too broad, set `ssl_ciphers`
- generate your own DH-parameters
- recent PostgreSQL supports ECC (ECDSA)
- new connections are expensive
- not that much overhead once connection is up

Connection Pooling

- Clients authenticate to pooler
- Pooler authenticates to PostgreSQL
- Forwarding of authentication with plaintext password only
- Terminates TLS

Summary

- Secure your databases
- Authenticate clients (and servers)
- At least, use passwords
- Encrypt connections (use TLS)
- Always verify certificates

Questions?