MySQL Security

by Wagner Bianchi

BIANCHI

Disclaimer

As informações fornecidas nesta apresentação são apenas para fins educacionais e informativos. Embora todos os esforços tenham sido feitos para garantir a precisão do conteúdo, o apresentador não garante a completude das informações. As práticas e recomendações de segurança discutidas são diretrizes gerais e podem não ser adequadas para todos os ambientes.

Os participantes são aconselhados a consultar os especialistas em segurança de sua organização ou assessoria jurídica antes de implementar qualquer medida de segurança discutida nesta apresentação. O apresentador e as organizações associadas não se responsabilizam por quaisquer danos ou perdas decorrentes do uso das informações apresentadas.

Abstract

Esta apresentação abordará a segurança no MySQL, desde conceitos básicos até avançados. Inclui gerenciamento de contas, autenticação, criptografía de dados, componentes e plugins de segurança, além de políticas de para controle de acesso remoto aos bancos de dados e como gerar credenciais dinâmicas com segmentação de privilégios com o Hashicorp Vault.

O objetivo é fornecer uma visão abrangente para proteger os ambientes de banco de dados MySQL, desde sua instalação até a sua utilização em ambiente de produção.

Wagner Bianchi

Com 20 anos de experiência na otimização de bancos de dados **MySQL** para clientes no Brasil e no exterior, Bianchi acumulou um vasto conhecimento ao trabalhar em corporações de renome como **Oracle**, **Splunk**, **IBM**, **Percona** e **MariaDB Corporation**. Nesta última, liderou um time global de DBAs no departamento de **Managed Services** (RDBA).

Atualmente, **Bianchi** se dedica a arquitetar, documentar e oferecer suporte a soluções complexas para empresas do setor financeiro no Vale do Silício-CA/US, uma função que desempenha nos últimos 3 anos, com foco em **MariaDB**, **MySQL** e **PostgreSQL**, on-premises, **AWS** e **GCP**.

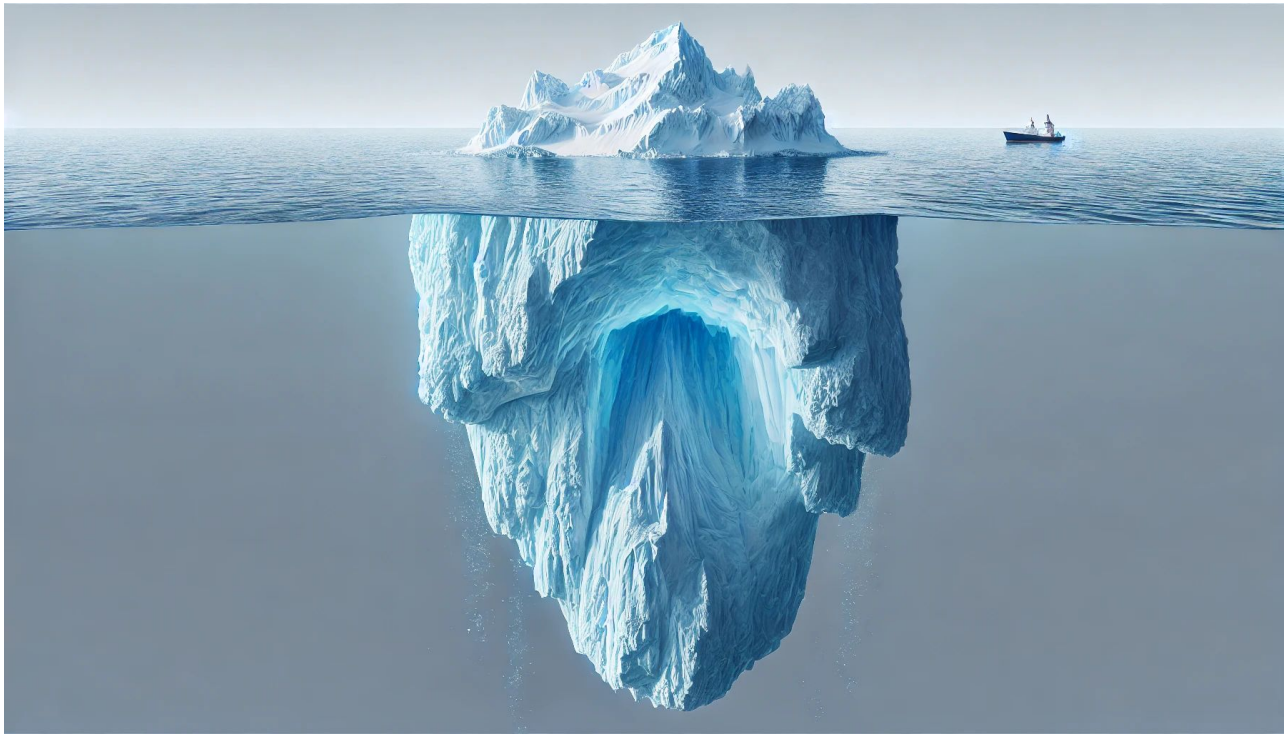
Bianchi é fundador do Grupo **MySQL Brasil**, que desde 2010 acumula usuários para diversas discussões sobre MySQL.

Entre em contato, vamos conversar: linkedin.com/in/wagnerbianchi/

Agenda

- Conceitos básicos de segurança para bancos de dados;
- Pontos de auditoria de segurança para uma nova instância MySQL;
- Mecanismos de autenticação e seus problemas (native x sha2);
- Controle de Acesso e Gerenciamento de Contas (MySQL ACLs);
- Componentes e Plugins de Segurança de Dados (MySQL Enterprise);
- Segurança de Host e de Rede (SSL/TLS e conexões seguras);
- *Authentication Plugins*, o que se deve e o que não se deve fazer;
- Recursos de Segurança Avançados (*SELinux* e *Hashicorp Vault*);
- **DEMO**: credenciais dinâmicas com *MySQL* e *Hashicorp Vault*.

O que você vê…



Conceitos Básicos de Segurança

- Os ataques mais populares (by BCS*) :
  - **Eavesdropping (altering, playback)**: interceptação, alteração de dados;
  - **DDOS -** exaustão de capacidade de recursos de máquina;
  - **SQL Injection -** código SQL para manipular e obter acesso aos dados;
  - **Vulnerabilities Exploitation -** correções são logo disponibilizadas em forma de releases;
  - **Ransomware -** tipo de *malware* que encripta os dados e solicita uma quantia em dinheiro;
  - **Credenciais fracas**, não rotacionadas de tempos em tempos, ou excesso de privilégios;
  - Auditoria de acessos que não funciona ou não é utilizado;
- Code Repositories:
  - Publicação de credenciais em meio à novos commits;
- Image Registries:
  - Docker Hub, Harbor, ECR;

* https://www.bcs.org/articles-opinion-and-research/top-ten-database-attacks/

Conceitos Básicos de Segurança

- Não utilize o usuário "`root`" nas atividades do seu dia-a-dia;
- Não dê privilégios a usuários não administrativos à tabela `mysql.user`;
- O usuário "root" deve ter uma senha forte e ser utilizado quando necessário;
- Crie usuários com a **política do menor privilégio** e sempre com permissões que vão até o tipo de objeto que se deve ter acesso - **1:1 sempre**;
- Não rode MySQL sem autenticação de usuários (`--skip-grant-tables`);
- Evite usuários compartilhados e para múltiplas finalidades;
- Utilize um padrão para definição de senhas;
- Autentique com `caching_sha2_password`- fast reconnection;
- `mysql_secure_installation`antes de iniciar suas instâncias MySQL.

Conceitos Básicos de Segurança

- **Para on-premisses**:
  - invista tempo para garantir configuração ótima de firewalls;
- **Para ambientes cloud**:
  - garanta que seu banco de dados managed ou self-managed não seja exposto publicamente na internet - mantenha-os o mais isolado possível;
- Tome cuidado com portas abertas em seu servidor;
- Somente alterar de 3306 para outra não é suficiente - filtre conexões;

```
➜  ~ nmap 146.190.66.30 #: https://nmap.org/book/port-scanning-tutorial.html
Starting Nmap 7.95 ( https://nmap.org ) at 2024-08-06 20:08 -03
Nmap scan report for 146.190.66.30
PORT     STATE    SERVICE
22/tcp   open     ssh
25/tcp   filtered smtp
3307/tcp open     mysql
```

Conceitos Básicos de Segurança

- Outros comandos você precisa ter defesas contra:
  - nc
  - telnet
  - tcpdump | strings

```
➜  ~ nc -vz 146.190.66.30 3306
Connection to 146.190.66.30 port 3306 [tcp/mysql] succeeded!

➜  ~ telnet 146.190.66.30 3306
Trying 146.190.66.30...
Connected to 146.190.66.30.
Escape character is '^]'.

➜  ~ tcpdump -l -i any -w - src or dst port 3306 | strings
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
<#M@
|AiI
8.4.2 -- Ahá!!! Já sabemos qual é a versão utilizada!!
```

Conceitos Básicos de Segurança

- MySQL:
  - Reforce o `SQL_MODE` em um MySQL de produção - **não confie nos dados *inputados* por usuários da aplicação** - prepare sua aplicação para lidar com ***SQL Injection***;
  - Aqui temos dois problemas a serem resolvidos:
    - Consistência dos dados - `STRICT_ALL_TABLES`, *`STRICT_TRANS_TABLES`;
    - Fazer com que o conversão interna de tipos de dados seja mais restrita;
  - Formulários de entrada de dados com os dados da condicional entre *single-quotes*;
  - Sua aplicação precisa estar atenta aos caracteres especiais;
- On-premises:
  - SELinux, fail2ban, iptables, auditd, …;
- Cloud (AWS like):
  - Firewalls, Network ACLs, Security Groups, Private Subnets, VPN, etc;

Atenção às seguintes variáveis…

`allow-suspicious-udfs`	`FALSE`	Ponto de atenção - recomenda-se não habilitar.
`local-infile`	`FALSE`	Ponto de atenção - recomenda-se não habilitar.
`skip-symbolic-links`	`FALSE`	Ponto de atenção - recomenda-se não habilitar.
`secure_file_priv`	`/path/to/tmp`	Ponto de atenção - recomenda-se não habilitar.
`sql_mode`	`STRCIT_ALL_TABLES`	Ponto de atenção - consistência de dados e desabilitação de conversões internas que podem ser exploradas.
`skip-grant-tables`	`FALSE`	Não suba o MySQL sem as *Grant Tables* / autenticação.
`skip-name-resolve`	`FALSE`	Atua na identificação do host tentando conexão através de um mecanismo de proxy-reverso.

`LOAD DATA` vs `LOAD DATA LOCAL`

- A diferença entre um **comando SQL** e outro é:
  - `LOAD DATA INFILE` para arquivos localizados no servidor de bancos de dados;
    - O usuário precisa ter *`FILE` como privilégio concedido (**Admin-Only Priv**);
    - A localização do arquivo deve seguir o `PATH` configurado em `@@global.secure_file_priv`;
    - **Trabalho para o DBA.**
  - `LOAD DATA` **`LOCAL`** `INFILE` para arquivos localizados no servidor cliente;
    - O usuário não precisa ter `FILE` como privilégio concedido;
    - O servidor MySQL deve ser configurado para aceitar a operação através da variável `@@global.local_infile=1`;
    - Conexão com o banco, interativa ou não, deve ser sobre SSL/TLS (`--ssl-mode=VERIFY_IDENTITY`);
    - **Considere o risco.**

Políticas para definição de Passwords

- Aqui temos dois tipos de passwords (ou secret):
  - Passwords para outros sistemas, armazenados em bancos de dados do usuário;
  - Passwords ou *Authentication Strings* armazenados no banco de dados do sistema (mysql);
- Passwords do usuário do MySQL:
  - Eles são armazenados nas Grant Tables, a qual só devem ser acessadas por DBA;
  - Há pouco tempo (MySQL 5.7.6) a coluna *password* foi renomeada *authentication_string*;
    - O password do usuário pode ter até **65.535** caracteres (TEXT Data Type);
    - **50 caracteres é o bastante**, desde que seja gerado por um gerador de secrets;
  - **MySQL 8.4 LTS Password Validation Component**:
    - O antigo "plugin" passou por reimplementação - componente;
    - Precisa ser devidamente instalado e configurado;

Políticas para definição de Passwords

- Instalação do componente:

```
mysql> INSTALL COMPONENT 'file://component_validate_password' ;
Query OK, 0 rows affected (0.03 sec)
mysql> select * from mysql.component\G
*************************** 1. row ***************************
     component_id: 1
component_group_id: 1
    component_urn: file://component_validate_password
1 row in set (0.01 sec)
```

- Forçar validação dos passwords informados na criação dos usuários;
- Analisa os comandos `CREATE USER`, `ALTER USER`, `SET PASSWORD`;
- Contas bloqueadas (`ACCOUNT LOCK`) tem os passwords verificados;
- Validation Password Component <> Validation Password Plugin;

Políticas para definição de Passwords

- Verifique seus passwords com **`VALIDATE_PASSWORD_STRENGTH()`**

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH("fz^bU4}?dtS.KXB(mgE$9R;V*Dh~WukM@6a%27L:YNG-]`QFPA"\G
***************************1. row ***************************
VALIDATE_PASSWORD_STRENGTH("fz^bU4}?dtS.KXB(mgE$9R;V*Dh~WukM@6a%27L:YNG-]`QFPA": 100
1 row in set (0.00 sec)
```

Password Test	Return Value
Length < 4	0
Length ≥ 4 and < validate_password_length	25
Satisfies policy 1 (LOW)	50
Satisfies policy 2 (MEDIUM)	75
Satisfies policy 3 (STRONG)	100

- Configure o componente antes de usar:

```
[mysqld]
validate_password.changed_characters_percentage = 100 #: 100% dos chars do antigo password são rejeitados no novo
validate_password.check_user_name               = 1   #: o username não pode fazer parte do password
validate_password.dictionary_file               = /usr/share/dict/words #: arquivos com palavras rejeitadas
validate_password.length                        = 50  #: tamanho da string de password
validate_password.mixed_case_count              = 5   #: qtd chars entre maiúsculas e minúsculas
validate_password.number_count                  = 5   #: qtd chars que são números
validate_password.policy                        = 2   #: 0-LOW, 1-MEDIUM, 2-STRONG
validate_password.special_char_count            = 5   #: qtd chars especiais
```

Políticas para definição de Passwords

- MySQL Dual-password:
  - `RETAIN CURRENT PASSWORD` ajuda a migrar contas antigas e **rotacionar passwords**;
  - Mantém o atual, e cria um novo, ao final `DISCARD OLD PASSWORD` descarta o antigo;

```
CREATE USER IF NOT EXISTS `jay`@`172.31.32.0/20255.255.240.0`
IDENTIFIED WITH caching_sha2_password
BY 'z_UyQ5E{3JYe8b[jCcrs' ;

ALTER USER IF EXISTS `jay`@`172.31.32.0/20255.255.240.0`
IDENTIFIED WITH caching_sha2_password
BY 'y_UyQ5E{3JYe8b[jCcrs'  RETAIN CURRENT  PASSWORD;

ALTER USER IF EXISTS `jay`@`172.31.32.0/20255.255.240.0`
DISCARD OLD  PASSWORD;
```

Políticas para definição de Passwords

- Controle mais ajustado para contas:

```
ALTER USER `jay`@`172.31.32.0/20255.255.240.0` PASSWORD EXPIRE NEVER;
ALTER USER `jay`@`172.31.32.0/20255.255.240.0` PASSWORD EXPIRE INTERVAL 90 DAY;
ALTER USER `jay`@`172.31.32.0/20255.255.240.0` PASSWORD EXPIRE INTERVAL DEFAULT;
```

- Padrão de expiração de password:
  - `@@global.default_password_lifetime`
  - `@@global.disconnect_on_expired_password` (enabled by default)

MySQL ACLs - Grant Tables

- GRANT tables são as tabela do banco de dados de sistema mysql;
  - **mysql.user**: usuários, privilégios globais estáticos e outras colunas relacionadas com certificado SSL, password control e detalhes do usuário;
  - **mysql.global_grants**: tabela para armazenamento de privilégios globais dinâmicos (8.0++);
  - **mysql.db**: privilégios a nível de banco de dados;
  - **mysql.tables_priv**: privilégios a nível de tabela;
  - **mysql.columns_priv**: privilégios a nível de coluna;
  - **mysql.procs_priv**: privilégios para procedimentos armazenados e funções;
  - **mysql.proxies_priv**: privilégios de usuários proxy;
  - **mysql.default_roles:** roles ativas lidas com a conexão de um dos usuários USER listados;
  - **mysql.role_edges**: mapeamento role x usuário;
  - **mysql.password_history**: histórico de alterações de senhas.

MySQL ACLs - GRANT e REVOKE

- Não altere as Grant Tables com comandos SQL DML;
- GRANT e REVOKE são para concessão e revogação de privilégios;
  - Um comando GRANT pode popular as tabelas user, db, tables_priv, columns_priv, procs_priv;
  - Um comando REVOKE sobre estes privilégios, remove todos os privilégios de um usuário;
- Exemplos rápidos:

```
GRANT SELECT (first_name, last_name) ON employees.employees TO jak@localhost;
REVOKE SELECT (first_name, last_name) ON employees.employees FROM jak@localhost;
--
GRANT EXECUTE ON PROCEDURE test.sp_test TO `jak`@`localhost`;
REVOKE EXECUTE ON PROCEDURE test.sp_test FROM `jak`@`localhost`;
```

MySQL ACLs - Roles

- **ROLE**s são basicamente **USER**s sem privilégio de autenticação;
- Também, um arcabouço contendo privilégios para um conjuntos de **USER**s;
- **ROLE**s podem ter um não a parte <host> do que vemos no **USER**s;
- **USER**s podem ter várias **ROLE**s e estas podem ser alteradas em uma sessão;
- **Sequência operacional básica:**
  a. Cria-se uma **ROLE** r_a;
  b. Concede-se privilégios a **ROLE** r_a;
  c. Cria-se um novo **USER** u_z;
  d. Concede-se a **ROLE** r_a ao **USER** u_z;
  e. Ativa-se a **ROLE** r_a para o **USER** u_z.
- Utilizar a variável `mandatory_roles` pode criar um problema de segurança!

MySQL ACLs - Criando Usuários

```
-- vamos validar os passwords?
mysql> SELECT VALIDATE_PASSWORD_STRENGTH("wNR>uDr7;b{[38<vHUksP=A@/E~eT'9?(Xmh`GpMY)c24S5BQ6") admin_jim
    ->        , VALIDATE_PASSWORD_STRENGTH("q&}kYMp=#>';7dL[N@4yv65~)/hQPAfc*bVSxH+RX<ea9GKZ-W") devel_joe
    ->        , VALIDATE_PASSWORD_STRENGTH("mePSA<-'k:JY8DhVK@?T`c&H2f/QaZ,6_Uw*tELn#=+pu4G]9{") sdbas_jay\G
*************************** 1. row ***************************
admin_jim: 100
devel_joe: 100
sdbas_jay: 100
1 row in set (0.00 sec)

-- criamos usuários com 50 chars passwords
CREATE USER IF NOT EXISTS `admin_jim`@`192.168.10.0/255.255.254.0` IDENTIFIED WITH caching_sha2_password
BY "wNR>uDr7;b{[38<vHUksP=A@/E~eT'9?(Xmh`GpMY)c24S5BQ6";
CREATE USER IF NOT EXISTS `devel_joe`@`192.168.30.0/255.255.254.0` IDENTIFIED WITH caching_sha2_password
BY "q&}kYMp=#>';7dL[N@4yv65~)/hQPAfc*bVSxH+RX<ea9GKZ-W";
CREATE USER IF NOT EXISTS `sdbas_jay`@`192.168.50.0/255.255.254.0` IDENTIFIED WITH caching_sha2_password
BY "mePSA<-'k:JY8DhVK@?T`c&H2f/QaZ,6_Uw*tELn#=+pu4G]9{";
```

MySQL ACLs - Criando Roles/Grant privilégios

```
-- criamos as roles para agrupar nosso usuários, sendo que cada grupo tem seus próprios privilégios GLOBAIS e de OBJECTOS
CREATE ROLE IF NOT EXISTS `grp_admin`@`192.168.10.0/255.255.254.0`;
GRANT SELECT, INSERT, DELETE, UPDATE, CREATE, INDEX, ALTER, TRIGGER
ON datasys.* TO `grp_admin`@`192.168.10.0/255.255.254.0`;
GRANT REPLICATION SLAVE, PROCESS, FILE
ON *.*  TO `grp_admin`@`192.168.10.0/255.255.254.0`;

CREATE ROLE IF NOT EXISTS `grp_devel`@`192.168.30.0/255.255.254.0`;
GRANT SELECT, INSERT, DELETE, UPDATE, CREATE, INDEX, ALTER
ON datasys.* TO `grp_devel`@`192.168.30.0/255.255.254.0`;

CREATE ROLE IF NOT EXISTS `grp_sdbas`@`192.168.50.0/255.255.254.0`;
GRANT SELECT, INSERT, DELETE, UPDATE, CREATE, INDEX, ALTER, TRIGGER, EXECUTE, REFERENCES
ON datasys.* TO `grp_sdbas`@`192.168.50.0/255.255.254.0` WITH GRANT OPTION;
GRANT REPLICATION SLAVE, PROCESS, FILE, CREATE USER, ROLE_ADMIN, SYSTEM_VARIABLES_ADMIN
ON *.*  TO `grp_sdbas`@`192.168.50.0/255.255.254.0` WITH GRANT OPTION;
```

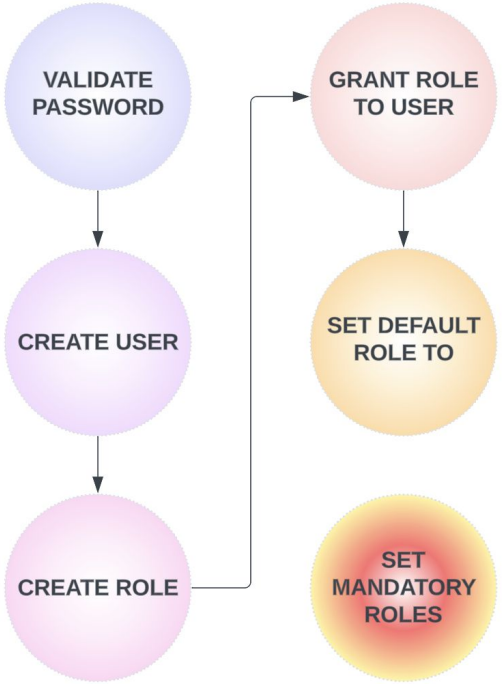
MySQL ACLs - Granting Roles e Mapping Roles to Users

```
-- atribuímos as roles para os usuários
GRANT `grp_admin`@`192.168.10.0/255.255.254.0` TO `admin_jim`@`192.168.10.0/255.255.254.0` WITH ADMIN OPTION;
GRANT `grp_devel`@`192.168.30.0/255.255.254.0` TO `devel_joe`@`192.168.30.0/255.255.254.0`;
GRANT `grp_sdbas`@`192.168.50.0/255.255.254.0` TO `sdbas_jay`@`192.168.50.0/255.255.254.0` WITH ADMIN OPTION;

-- verificamos as permissões dos usuários herdadas pelas roles
SHOW GRANTS FOR `admin_jim`@`192.168.10.0/255.255.254.0` USING `grp_admin`@`192.168.10.0/255.255.254.0`;
SHOW GRANTS FOR `devel_joe`@`192.168.30.0/255.255.254.0` USING `grp_devel`@`192.168.30.0/255.255.254.0`;
SHOW GRANTS FOR `sdbas_jay`@`192.168.50.0/255.255.254.0` using `grp_sdbas`@`192.168.50.0/255.255.254.0`;

-- mapeamos a roles para os usuários: quando eles se conectarem, as roles serão inicializadas
-- essas roles não devem ser adicionadas ao @global.mandatory_roles
SET DEFAULT ROLE `grp_admin`@`192.168.10.0/255.255.254.0` TO `admin_jim`@`192.168.10.0/255.255.254.0`;
SET DEFAULT ROLE `grp_devel`@`192.168.30.0/255.255.254.0` TO `devel_joe`@`192.168.30.0/255.255.254.0`;
SET DEFAULT ROLE `grp_sdbas`@`192.168.50.0/255.255.254.0` TO `sdbas_jay`@`192.168.50.0/255.255.254.0`;
```

MySQL ACLs - Roles



- Ressalvas para esse processo:
  - *mandatory_roles* pode trazer um problema de segurança com um furo na segmentação dos usuários;
  - Roles nomeadas como *mandatory* não podem ser revogadas;
  - Roles podem ser criadas omitindo a parte "host", mas isso faz com que elas sejam criadas com host as "%";
  - Uma ou mais roles podem ser concedidas para um dado usuário - **cuidado com o *spider-web effect***!
  - Usuário podem ter roles concedidas com a declaração `WITH ADMIN OPTION` - mysql.roles_edge;

MySQL ACLs - Atenção

- Não conceda `ALL PRIVS` ou mesmo, `SUPER` ou `PROCESS` para usuários do seu banco de dados MySQL - somente o usuário root deveria tê-los;
  - Um usuário que precisa configurar a variável super_read_only necessitaria ter `SUPER`. Mas, graças aos privilégios dinâmicos, você pode conceder o `SYSTEM_VARIABLES_ADMIN`;
  - Assim como o x, temos outros que são um subconjunto do `SUPER`:
    - `BINLOG ADMIN`,
    - `CONNECTION ADMIN`,
    - `ENCRYPTION KEY ADMIN`,
    - `GROUP REPLICATION ADMIN`,
    - `REPLICATION SLAVE ADMIN`,
    - `SESSION VARIABLES_ADMIN`,
    - `SET USER ID`,
    - `SYSTEM_VARIABLES_ADMIN`
- **Perguntas essenciais:**
  - É necessário criar um novo usuário?
  - Se sim, quais são as atribuições desse novos usuário?
  - Temos uma ROLE definida para ele ou teremos que criar um nova ROLE?
  - Por que esse usuário precisa acessar tais dados/objetos no banco?

Autenticação

- O MySQL 5.7 trouxe a possibilidade de utilizarmos plugins de autenticação:
  - **caching_sha2_password**: O plugin padrão e mais seguro, que utiliza o algoritmo SHA-256;
  - **mysql_native_password**: Um plugin **legado** que usa hashing com SHA-1;
  - **sha256_password**: Usa SHA-256 para maior segurança;
  - **auth_socket**: Permite autenticação baseada no nome do usuário do sistema operacional;
  - **auth_pam**: Integração com sistemas de autenticação PAM (*Pluggable Authentication Modules*);
  - **authentication_ldap**: Permite autenticação usando LDAP.
- Verificando plugins:
  - `SHOW PLUGINS;`
  - `SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_TYPE = 'AUTHENTICATION';`

Autenticação Benchmark

- Benchmark com o mesmo dataset, acessado com dois usuários (20m):
  - Aqui criei um usuário com cada um dos plugins: ***native_mysql x caching_sha2***;
  - Existem ganhos que podem ser maiores se comparados em um ambiente maior;
  - O `caching_sha2_password` além de dar mais performance, entrega mais segurança;
  - O `native_mysql_password` é algo deprecated no MySQL 8.0++, e desabilitado *by default*;
    - Pode ser habilitado para *backward support*: `--mysql_native_password=ON`

Authentication Method	*Native MySQL Password*	*Caching SHA2 Password*
Número de transações	176691 (147.22 per sec.)	204.311 (170.25 per sec.)
Leituras	2.473.674	2.860.354
Escritas	706.764	817.244

SSL/TLS - client/server

- Ficou mais fácil adotar conexões seguras desde o MySQL 5.7, que disponibiliza certificados self-signed na sua instalação:

```
[mysqld]
ssl-capath=/var/lib/mysql
ssl_ca="ca.pem"
ssl_cert="server-cert.pem"
ssl_key="server-key.pem"
require_secure_transport="ON"
```

- No cliente, acesse o mysql com os certificados client:

```
mysqladmin -u bianchi -p \
--ssl-mode=VERIFY_CA --ssl-ca=/var/lib/mysql/ca.pem \
--ssl-cert=/var/lib/mysql/client-cert.pem \
--ssl-key=/var/lib/mysql/client-key.pem -h 172.31.43.231 -P 3306 ping
mysqld is alive
```

SSL/TLS - mysqlx plugin

- Adote conexões seguras para as chamadas à Dev API com mysqlsh*

```
[mysqld]
mysqlx-ssl-capath  = /var/lib/mysql
mysqlx-ssl-ca      = ca-cert.pem
mysqlx-ssl-cert    = server-cert.pem
mysqlx-ssl-key     = server-key.pem
```

- Seus usuários se conectam à API X:

```
mysqlsh --ssl-mode=VERIFY_CA \
        --host=172.31.43.231 \
        --port=33060 \
        --user=bianchi \
        --password
```

MySQL Enterprise

MySQL Enterprise **Masking**
- De-identify, Anonymize Sensitive Data

MySQL Enterprise **TDE**
- AES 256 encryption, Key Management

MySQL Enterprise **Authentication**
- External Authentication Modules

MySQL Enterprise **Encryption**
- Public/Private Key Cryptography, Asymmetric Encryption

MySQL Enterprise **Firewall**
- Block SQL Injection Attacks, Intrusion Detection

MySQL Enterprise **Audit**
- User Activity Auditing, Regulatory Compliance

MySQL Enterprise **Telemetry**
- Telemetry data directly from within MySQL
- Open Telemetry Traces and Metrics

MySQL Enterprise **Backup**
- High Performance, Online Backup
- Secure Backups, AES 256 encryption

MySQL e Hashicorp Vault

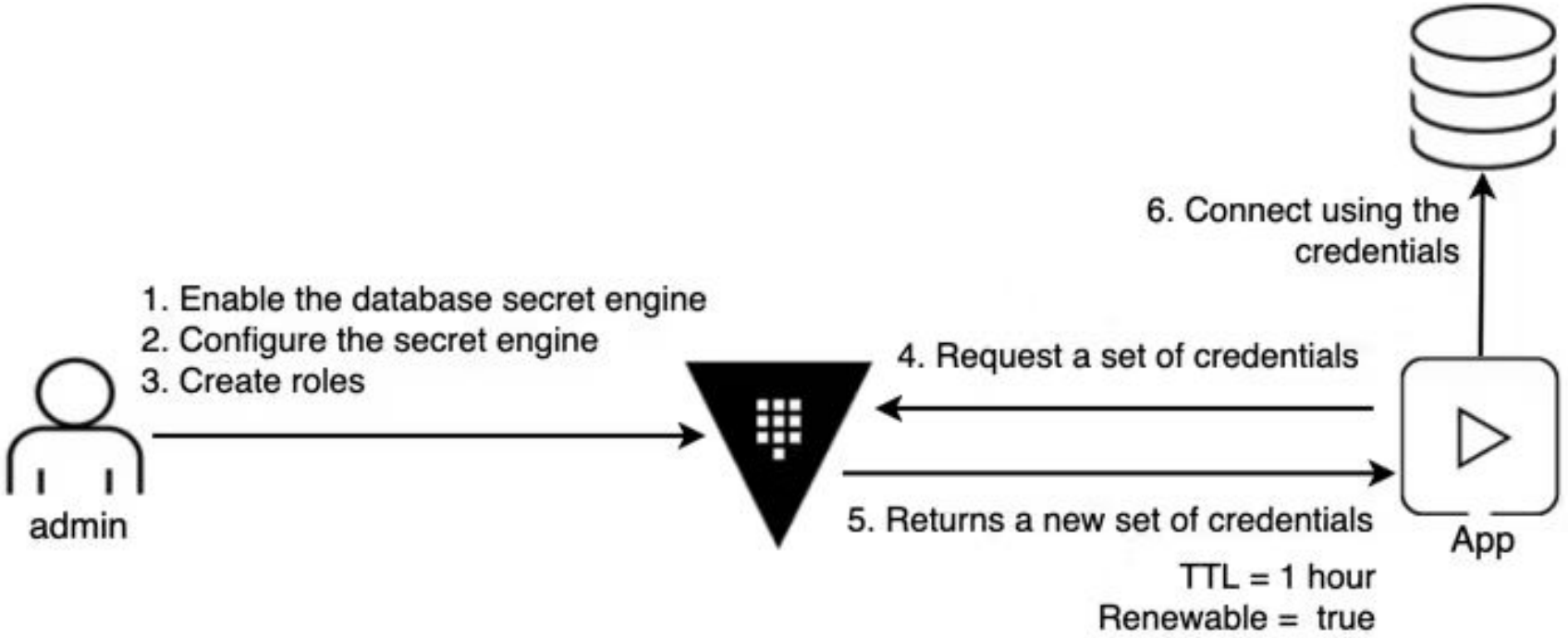
- Antes de pensar na especialização de componentes;
- Antes mesmo de se pensar na quantidade de tempo que dura uma conexão;
  - Usuários individuais eram disponibilizados (LDAP, Active Directory, …);
  - Cada usuário tinha o seu /home/user nas máquinas que se obtinha acesso;
  - Em cada home directory se disponibilizava um arquivo .my.cnf;
  - O usuário não tinha *sudo*.
- Esse tipo de arquitetura hoje passa por vários problemas:
  - Ter arquivos com passwords nas máquinas pode ser um problema grave;
  - O DBA precisa ter sudo e com o tempo esse ambiente começa a ter muitas exceções;
  - As regras então viram as excessões e a segurança do ambiente desanda;
  - Foi preciso pensar em algo diferente.

MySQL e Hashicorp Vault

- Foi preciso pensar em algo diferente.
  - Um componente externo que tem um usuário de serviços para acesso ao banco de dados;
  - O DBA pode requisitar credenciais sempre que é preciso acessar o banco de dados;
  - Essas credenciais são válidas por 60 minutos, podendo ser extendidas por mais 30 minutos;
- Surgiu a ideia de gerarmos **Credenciais Dinâmicas** com o Hashicorp Vault;

```
$ vault write mariadb/roles/store-developers \
db_name="mariadb-dynamic-creads" \
creation_statements="CREATE USER IF NOT EXISTS '{{name}}'@'%' IDENTIFIED BY '{{password}}' REQUIRE SSL; \
GRANT SELECT, INSERT, DELETE, UPDATE, CREATE, ALTER ON store.* TO '{{name}}'@'%'; \
FLUSH PRIVILEGES;" \
default_ttl=30m max_ttl=60m
```

MySQL e Hashicorp Vault



Obrigado.