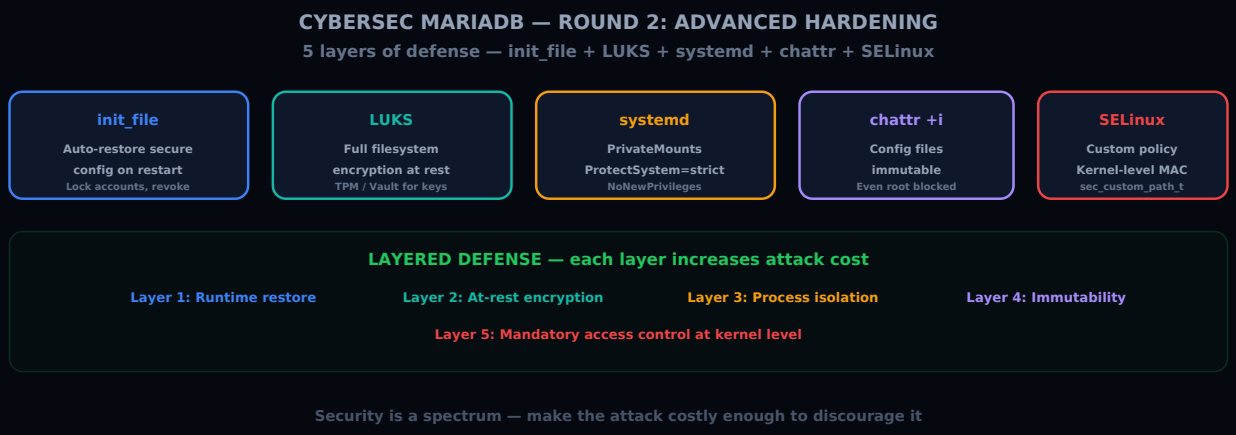
Cybersécurité MariaDB : round 2

Sylvain ARBAUDIE · 8 juin 2025

MARIADB SECURITY HARDENING SYSTEMD SELINUX



Au-delà des fondamentaux

Le premier round de sécurité MariaDB / MySQL couvre les bases : mots de passe forts, utilisateurs minimaux, TLS activé, pare-feu configuré. Le round 2 va plus loin. On entre dans le territoire du hardening avancé — des techniques que peu de DBA implémentent mais qui font la différence face à un attaquant déterminé.

init_file : le script silencieux

La variable `init_file` de MariaDB / MySQL permet de spécifier un fichier SQL qui sera exécuté automatiquement au démarrage du serveur. C'est un outil puissant pour le hardening :

```
[mysqld]
init_file = /etc/mysql/conf.d/init_security.sql
```

Le fichier `init_security.sql` peut contenir :

```
-- Désactiver les comptes par défaut
ALTER USER 'root'@'localhost' ACCOUNT LOCK;

-- Révoquer les privilèges excessifs
```

```
REVOKE ALL PRIVILEGES ON *.* FROM 'app_user'@'%';
GRANT SELECT, INSERT, UPDATE, DELETE ON app_db.* TO 'app_user'@'%';

-- Supprimer les bases de test
DROP DATABASE IF EXISTS test;

-- Activer l'audit
INSTALL SONAME 'server_audit';
SET GLOBAL server_audit_logging = ON;
```

L'avantage : même si un attaquant modifie la base pendant une intrusion, le redémarrage du serveur restaure automatiquement la configuration sécurisée.

LUKS : chiffrement du filesystem

Les données MariaDB au repos doivent être chiffrées. InnoDB supporte le chiffrement de tablespace natif, mais LUKS (Linux Unified Key Setup) offre une protection plus complète : il chiffre tout le filesystem, y compris les logs, les fichiers temporaires et les fichiers de configuration.

```
# Créer un volume chiffré LUKS pour le datadir
cryptsetup luksFormat /dev/sdb1
cryptsetup luksOpen /dev/sdb1 mariadb_data
mkfs.ext4 /dev/mapper/mariadb_data
mount /dev/mapper/mariadb_data /var/lib/mysql
```

La clé LUKS ne doit jamais être stockée sur le même disque que les données. Utilisez un module TPM, un token USB, ou un service de gestion de clés externe (Vault, AWS KMS).

systemd : defaults-file et PrivateMounts

Le fichier unit systemd de MariaDB peut être renforcé de plusieurs façons :

--defaults-file explicite

```
[Service]
ExecStart=/usr/sbin/mariadbd --defaults-file=/etc/mysql/mariadb.cnf
```

Spécifier `--defaults-file` empêche MariaDB de lire d'autres fichiers de configuration (comme un `~/.my.cnf` malveillant déposé par un attaquant).

PrivateMounts et namespaces

```
[Service]
PrivateMounts=yes
ProtectHome=yes
ProtectSystem=strict
ReadWritePaths=/var/lib/mysql /var/run/mysqld /tmp
NoNewPrivileges=yes
PrivateTmp=yes
```

- **PrivateMounts=yes** : MariaDB voit son propre namespace de montage. Les modifications de montage faites par d'autres processus ne sont pas visibles.
- **ProtectHome=yes** : Le répertoire /home est inaccessible.
- **ProtectSystem=strict** : Le filesystem est en lecture seule sauf les chemins explicitement autorisés.
- **NoNewPrivileges=yes** : Le processus MariaDB ne peut pas acquérir de nouveaux privilèges (pas de setuid).
- **PrivateTmp=yes** : MariaDB a son propre /tmp isolé.

chattr : l'immutabilité

L'attribut `+i` (immutable) du filesystem ext4 empêche la modification d'un fichier, même par root :

```
# Rendre le fichier de configuration immutable
chattr +i /etc/mysql/mariadb.cnf
chattr +i /etc/mysql/conf.d/init_security.sql

# Vérifier
lsattr /etc/mysql/mariadb.cnf
# ----i--------e-- /etc/mysql/mariadb.cnf
```

Un attaquant qui obtient un accès root ne pourra pas modifier la configuration de MariaDB sans d'abord retirer l'attribut immutable — ce qui laisse des traces dans les logs d'audit.

Pour modifier le fichier légitimement :

```
chattr -i /etc/mysql/mariadb.cnf
# ... modifier le fichier ...
chattr +i /etc/mysql/mariadb.cnf
systemctl restart mariadb
```

SELinux : politiques personnalisées

SELinux en mode enforcing est la couche de sécurité la plus puissante et la plus négligée. MariaDB est livrée avec une politique SELinux par défaut, mais une politique personnalisée peut aller beaucoup plus loin.

Créer un type SELinux personnalisé

```
# Définir un type pour les fichiers de configuration sensibles
semanage fcontext -a -t sec_custom_path_t "/etc/mysql/conf.d(/.*)?"
restorecon -Rv /etc/mysql/conf.d/
```

Politique de module personnalisée

Créez un fichier `.te` (Type Enforcement) qui restreint les accès de MariaDB :

```
# mariadb_custom.te
module mariadb_custom 1.0;

require {
    type mysqld_t;
    type sec_custom_path_t;
    class file { read open getattr };
}

# MariaDB peut lire les configs mais pas les modifier
allow mysqld_t sec_custom_path_t:file { read open getattr };
# Pas d'écriture autorisée sur les configs
```

Compilez et installez :

```
checkmodule -M -m -o mariadb_custom.mod mariadb_custom.te
semodule_package -o mariadb_custom.pp -m mariadb_custom.mod
semodule -i mariadb_custom.pp
```

Avec cette politique, même si un attaquant compromet le processus MariaDB, il ne peut pas modifier les fichiers de configuration — SELinux bloque l'accès au niveau du noyau.

La défense en couches

Chaque technique présentée ici est une couche de défense. Aucune n'est suffisante seule. Ensemble, elles forment un blindage significatif :

Couche	Protection	Contre
init_file	Restauration automatique	Modifications de config en runtime
LUKS	Chiffrement au repos	Vol de disque physique
systemd namespaces	Isolation du processus	Escalade de privilèges
chattr +i	Immutabilité des configs	Modification par root compromis
SELinux	Contrôle d'accès obligatoire	Exploitation du processus MariaDB

Conclusion

Le hardening avancé de MariaDB demande du temps et de l'expertise. Chaque couche ajoutée rend l'attaque plus difficile, plus lente, et plus détectable.

La sécurité n'est pas un état binaire — c'est un spectre. L'objectif n'est pas d'être invulnérable (c'est impossible), mais de rendre l'attaque suffisamment coûteuse pour que l'attaquant passe à une cible plus facile.

Cet article a été initialement publié sur Medium.