

Referat de securitate

Securitatea Bazelor de Date

PostgreSQL

Autor: Murariu Andrei

Rezumat

În acest referat se efectuează o analiză detaliată a mecanismelor de securitate disponibile în ediția "Community a PostgreSQL", demonstrând cum pot fi utilizate pe o bază de date medicală simplă formată din trei entități principale: "pacient", "personal_medical" și "fișă_medicală". Lucrarea abordează mai multe straturi de securitate, inclusiv securitatea rețelei prin utilizarea metodelor de autentificare sigure "(scram-sha-256)" și configurația fișierului "pg_hba.conf" pentru controlul accesului bazat pe IP. Cele paradigmă fundamentale ale controlului accesului sunt examineate: Gestionaerea accesului bazat pe rol "(RBAC)", controlul accesului discrețional "(DAC)" și controlul accesului obligatoriu "(MAC)" care sunt implementate prin siguranța nivelului rândului "(RLS)". Referatul prezintă modul de implementare a unui sistem de auditare care utilizează trigger-uri PostgreSQL pentru înregistrarea tuturor operațiunilor legate de date sensibile, precum și metode de prevenire a atacurilor de tip "SQL Injection" prin utilizarea declarațiilor pregătite. În ceea ce privește criptarea, există o serie de metode diferite. Acestea includ criptarea la nivel de disc complet folosind programe precum "FileVault", "BitLocker" și "LUKS"; criptarea la nivel de coloană folosind extensia "pgcrypto"; criptarea în tranzit prin "SSL/TLS". În cele din urmă, se arată cum se utilizează vizualizările "(Views)" pentru a îmbunătăți securitatea datelor prin ascunderea structurii tabelelor și limitarea accesului la date. Totul este și vizibil public în repozitoriu-ul meu de Github la urmatorul Link: https://github.com/iamxorum/Master_Unibuc/tree/main/SBD/Report

Cuprins

1 Pregătirea mediului pentru referat	3
1.1 Cerințe	3
1.2 Pornirea mediului	3
2 Schema bazei de date	4
3 Securitatea Rețelei	5
4 Controlul Accesului	7
4.1 Role Based Access Control	7
4.2 Discretionary Access Control	9
4.3 Mandatory Access Control	9
5 Auditare	10
5.1 Trigger	10
5.2 pgAudit	11
6 Prevenirea SQL Injection	12
7 Criptarea Datelor	13
7.1 Full Disk Encryption - La nivel de disk	13
7.2 File System Encryption - La nivel de os	14
7.3 Criptarea la nivel de coloană	14
7.4 Criptare la nivel de rețea	15
7.5 Criptarea parolelor	15
8 Visualizări pentru Securitate	16
9 Concluzii	17

Capitolul 1

Pregătirea mediului pentru referat

Instalarea PostgreSQL Community Edition și configurarea unui mediu de testare sunt necesare pentru a rula exemplele prezentate în acest referat. Pentru a face configurarea mediului proiectului mai simplă, Docker este folosit. Mediul este pe un VPS închiriat (securizat prin regulile de firewall iptables) și accesat doar prin VPN (Wireguard); Subnet-ul de rețea prin care se poace accessa (prin VPN) este 10.80.0.0/16 unde userii coneptați prin VPN sunt pe subnet-ul 10.80.1.0/24 și VPS-urile/Nodurile pe subnet-ul 10.80.0.0/24.

1.1 Cerințe

- **Docker și Docker Compose:** utilizate pentru containerizarea și orchestrarea serviciilor;
- **PostgreSQL Community Edition** (versiunea 14 sau superioară): sistemul de gestiune a bazelor de date relaționale;
- **Acces la linia de comandă:** necesar pentru execuția scripturilor SQL și administrarea containerelor.

1.2 Pornirea mediului

Mediul poate să fie rulat oriunde, important că este instalat Docker și Docker Compose. Comanda necesară pentru pornire este urmatoarea

```
docker-compose up -d
```

Capitolul 2

Schema bazei de date

Pentru referatul acesta (care va fi implementat în continuare pentru aplicația de evaluare SBD) o să mă folosesc de un DB Schema simplu compus de 3 entități:

- pacient;
- personal_medical;
- fisa_medicala;

Relațiile sunt următoarele:

- Pacient (1) - (N) Fisa_medicala;
- Personal_medical (1) - (N) Fisa_medicala;

Jos puteți observa rularea creării a schemei:

```
[2025-12-30 20:06:57] Connected
referat_db,public> CREATE TABLE pacient (
    id_pacient SERIAL PRIMARY KEY,
    nume VARCHAR(50) NOT NULL,
    prenume VARCHAR(50) NOT NULL,
    cnp VARCHAR(13) UNIQUE NOT NULL,
    telefon VARCHAR(15),
    adresa TEXT
)
[2025-12-30 20:06:57] completed in 101 ms
referat_db,public> CREATE TABLE personal_medical (
    id_personal SERIAL PRIMARY KEY,
    nume VARCHAR(50) NOT NULL,
    prenume VARCHAR(50) NOT NULL,
    specializare VARCHAR(50),
    username_db VARCHAR(60) UNIQUE NOT NULL,
    grad_acreditare INT NOT NULL DEFAULT 1
)
[2025-12-30 20:06:58] completed in 92 ms
referat_db,public> CREATE TABLE fisa_medicala (
    id_fisa SERIAL PRIMARY KEY,
    id_pacient INT REFERENCES pacient(id_pacient),
    id_medic INT REFERENCES personal_medical(id_personal),
    diagnostic TEXT NOT NULL,
    tratament TEXT,
    data_consultatie TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    nivel_clasificare INT NOT NULL DEFAULT 1 CHECK (nivel_clasificare IN (1, 2, 3))
)
[2025-12-30 20:06:58] completed in 91 ms
```

Figura 2.1: Creare schema

Capitolul 3

Securitatea Rețelei

Primul pas în securizarea bazei de date este de a determina cine și cum poate accesa baza de date în fișierul pg_hba.conf, unde se stabilește ce intervale de IP sunt permise sau respinse pentru acces. Este util să se protejeze accesul la nivel de firewall, dar este util să existe încă un strat de securitate în cazul în care firewall-ul nu funcționează.

TYPE	DATABASE	USER	ADDRESS	METHOD
local	all	all		scram-sha-256
host	all	all	127.0.0.1/32	scram-sha-256
host	all	all	10.80.0.0/16	scram-sha-256
host	all	all	0.0.0.0/0	reject
host	all	all	::/0	reject

Asta este o configurație minimală care permite acces-ului la baza de date în local cu parolă de autentificare; permite access-ului prin parola la userii care provin din range-ul de IP 10.80.0.0/16 (VPN în cazul acesta) și localhost; în final se dă reject la restul userilor care provin de altundeva la nivelul IPv4 și IPv6.

O vulnerabilitate era să fie trecut pentru tipul de conexiune "local" metoda "trust" unde se permitea conexiunea fară parolă.

O altă vulnerabilitate era folosirea metodei "md5" în loc de "scram-sha-256" deoarece md5 este mai nesigur (și-a pierdut setarea by default de la PostgreSQL 14 unde scram-sha-256 este acum setat default).

MD5 ca să fie considerat sigur trebuie să îndeplinească următoarele: - Ar trebui să fie capabil să convertească ușor informații digitale, cum ar fi un mesaj, într-o valoare hash cu lungime fixă. - Hash-ul trebuie să fie imposibil de decriptat din punct de vedere computational pentru a obține orice informații despre mesajul de intrare. - Din punct de vedere computational, trebuie să fie imposibil să se găsească două fișiere cu un hash identic.

Din păcate al treilea punct nu este îndeplinit deoarece este posibil să se genereze același hash pentru două fișiere diferite; situația asta aduce la atacuri de tip coliziune,

o vulnerabilitate extrem de ridicată, deoarece pot să generez același hash pentru o altă parolă spre exemplu. [md5collisions]

O altă metodă bună pentru securizarea accesului prin rețea este setarea tipului de conexiune în loc de "host" cu "hostssl". Pentru referat nu vom folosi SSL.

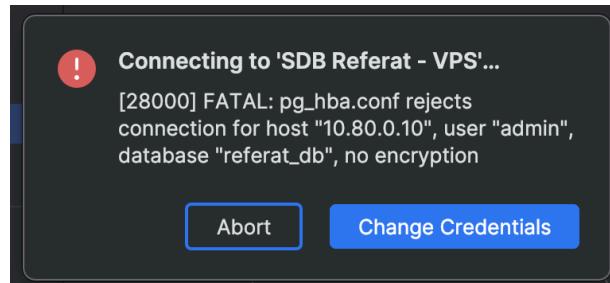


Figura 3.1: Conexiune cu Hostssl

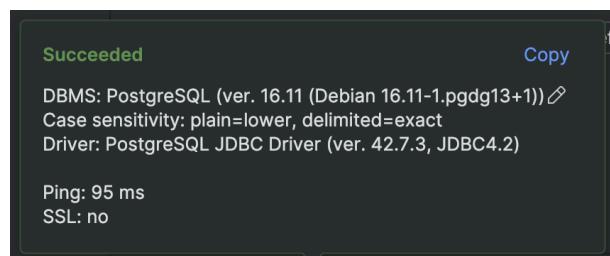


Figura 3.2: Conexiune cu Host

Capitolul 4

Controlul Accesului

Pentru referatul acesta, o să abordez 3 tipuri de access control (menționate și la cursul SBD):

- **RBAC:** Role Based Access Control;
- **MAC:** Mandatory Access Control;
- **DAC:** Discretionary Access Control;

4.1 Role Based Access Control

Acest tip de access control bazează accesul prin roluri. În loc de a configura pentru fiecare personal_medical ce are voie să facă, să vadă, să modifice, să steargă (CRUD), stabilim asta pe baza de rol aşa că daca e nevoie să se steargă/adauge/modifce o regulă de access, este mai ușor de intreținut/mentinut pentru ca se face doar odată schimbarea.

```
referat_db:public> CREATE ROLE rol_medic
[2025-12-30 20:47:03] completed in 86 ms
referat_db:public> CREATE ROLE rol_asistent
[2025-12-30 20:47:03] completed in 83 ms
referat_db:public> CREATE ROLE rol_rezident
[2025-12-30 20:47:03] completed in 83 ms
referat_db:public> GRANT ALL ON fisa_medicala TO rol_medic
[2025-12-30 20:47:04] completed in 84 ms
referat_db:public> GRANT ALL ON pacient TO rol_medic
[2025-12-30 20:47:04] completed in 84 ms
referat_db:public> GRANT ALL ON personal_medical TO rol_medic
[2025-12-30 20:47:04] completed in 83 ms
referat_db:public> GRANT SELECT ON pacient TO rol_asistent
[2025-12-30 20:47:05] completed in 83 ms
referat_db:public> GRANT SELECT ON fisa_medicala TO rol_asistent
[2025-12-30 20:47:05] completed in 85 ms
referat_db:public> GRANT SELECT ON personal_medical TO rol_asistent
[2025-12-30 20:47:05] completed in 85 ms
referat_db:public> GRANT SELECT ON fisa_medicala TO rol_rezident
[2025-12-30 20:47:06] completed in 83 ms
referat_db:public> CREATE USER "ana.popescu" WITH PASSWORD 'parola123'
[2025-12-30 20:47:06] completed in 89 ms
referat_db:public> GRANT rol_medic TO "ana.popescu"
[2025-12-30 20:47:06] completed in 84 ms
referat_db:public> CREATE USER "mihai.marinescu" WITH PASSWORD 'parola123'
[2025-12-30 20:47:07] completed in 90 ms
referat_db:public> GRANT rol_asistent TO "mihai.marinescu"
[2025-12-30 20:47:07] completed in 83 ms
referat_db:public> CREATE USER "elena.constantinescu" WITH PASSWORD 'parola123'
[2025-12-30 20:47:07] completed in 93 ms
referat_db:public> GRANT rol_rezident TO "elena.constantinescu"
[2025-12-30 20:47:08] completed in 83 ms
[2025-12-30 20:47:12] transaction committed: @console [SDB Referat - VPS]
```

Figura 4.1: Configurare RBAC

După ce am creat rolurile/userii pentru acces, putem testa (prin linie de comandă).

Se poate observa că medicul chiar are acces la orice (poate să adauge/șteargă pacienți noi spre exemplu), când asistentul vede totul dar nu are drept de modificare/ștergere/adăugare a datelor. În final, rezidentul are acces doar la fișele medicale.

```

[XRM-SRV01] iomxorum[XRM-SRV01]~ 
+ psql -h 10.80.0.40 -p 5432 -U "ana.popescu" -d referat_db
Password for user ana.popescu:
psql (17.6 (Debian 17.6-0+deb13u1), server 16.11 (Debian 16.11-1.pgdg13+1))
Type "help" for help.

referat_db=> \dp pacient
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | pacient | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> SELECT * FROM pacient LIMIT 1;
 id_pacient | nume | prenume | cnp | telefon |           adresu
-----+-----+-----+-----+-----+-----+
 1 | Popescu | Ion | 19800123456789 | 07123456789 | Strada Mihai Eminescu, Nr. 10, Bucuresti
(1 row)

referat_db=> INSERT INTO pacient (nume, prenume, cnp, telefon, adresu)
VALUES ('Test', 'Medic', '20000123456789', '0711111111', 'Test Address')
RETURNING *;
ERROR: permission denied for sequence pacient_id_pacient_seq
referat_db=> INSERT INTO pacient (id_pacient, nume, prenume, cnp, telefon, adresu)
VALUES (4, 'Test', 'Medic', '20000123456789', '0711111111', 'Test Address')
RETURNING *;
 id_pacient | nume | prenume |   cnp | telefon |           adresu
-----+-----+-----+-----+-----+-----+
 4 | Test | Medic | 20000123456789 | 0711111111 | Test Address
(1 row)

INSERT 0 1
referat_db=> DELETE FROM pacient WHERE cnp = '20000123456789';
DELETE 1
referat_db=> \dp fisa_medicala
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | fisa_medicala | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> \dp personal_medical
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | personal_medical | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> \dp
referat_db=> \q

[XRM-SRV01] iomxorum[XRM-SRV01]~ 
+ psql -h 10.80.0.40 -p 5432 -U "mihai.morinescu" -d referat_db
Password for user mihai.morinescu:
psql (17.6 (Debian 17.6-0+deb13u1), server 16.11 (Debian 16.11-1.pgdg13+1))
Type "help" for help.

referat_db=> \dp pacient
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | pacient | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> SELECT * FROM pacient;
 id_pacient | nume | prenume |   cnp | telefon |           adresu
-----+-----+-----+-----+-----+-----+
 1 | Popescu | Ion | 19800123456789 | 07123456789 | Strada Mihai Eminescu, Nr. 10, Bucuresti
 2 | Ionescu | Maria | 19980507890123 | 0723456789 | Bulevardul Unirii, Nr. 25, Cluj-Napoca
 3 | Georgescu | Andrei | 1985123456789 | 0734567890 | Strada Republicii, Nr. 5, Timisoara
(3 rows)

referat_db=> \dp fisa_medicala
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | fisa_medicala | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> SELECT * FROM fisa_medicala;
referat_db=> SELECT * FROM fisa_medicala;
referat_db=> INSERT INTO pacient (id_pacient, nume, prenume, cnp, telefon, adresu) VALUES (4, 'Test', 'Asistent', '2000123456789', '0711111111', 'Test Address')
ERROR: permission denied for table pacient
referat_db=> \q

```

Figura 4.2: Rol Medic

Figura 4.3: Rol Asistent

```

[XRM-SRV01] iomxorum[XRM-SRV01]~ 
+ psql -h 10.80.0.40 -p 5432 -U "eleno.constantinescu" -d referat_db
Password for user eleno.constantinescu:
psql (17.6 (Debian 17.6-0+deb13u1), server 16.11 (Debian 16.11-1.pgdg13+1))
Type "help" for help.

referat_db=> \dp fisa_medicala
          Access privileges
Schema | Name | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+
public | fisa_medicala | table | admin=arwdxt/admin +| | 
       |         |         | rol_medic=arwdxt/admin+| | 
       |         |         | rol_asistent=r/admin +| | 
       |         |         | rol_rezident=r/admin | | 
(1 row)

referat_db=> SELECT * FROM fisa_medicala;
referat_db=> SELECT * FROM pacient;
ERROR: permission denied for table pacient
referat_db=> SELECT * FROM personal_medical;
ERROR: permission denied for table personal_medical
referat_db=> \q

```

Figura 4.4: Rol Rezident

```

referat_db=# SELECT r.role_name AS role_name, m.member_name AS member_name
FROM pg_roles r
JOIN pg_auth_members m ON r.oid = m.roleid
JOIN pg_roles om ON m.member = om.oid
WHERE r.role_name LIKE 'rol_%'
ORDER BY r.role_name, m.member_name;
role_name | member_name
-----+-----
rol_asistent | mihai.morinescu
rol_medic | ana.popescu
rol_rezident | eleno.constantinescu
(3 rows)

referat_db=# SELECT
    grantee AS role_name,
    table_name,
    string_agg(privilege_type, ', ' ORDER BY privilege_type) AS privileges
FROM information_schema.role_table_grants
WHERE grantee LIKE 'rol_%'
AND table_name NOT IN ('pg_temp_1', 'pg_temp_2')
GROUP BY grantee, table_name
ORDER BY grantee, table_name;
role_name | table_name | privileges
-----+-----+-----+
rol_asistent | fisa_medicala | SELECT
rol_asistent | pacient | SELECT
rol_asistent | personal_medical | SELECT
rol_medic | fisa_medicala | DELETE, INSERT, REFERENCES, SELECT, TRIGGER, TRUNCATE, UPDATE
rol_medic | pacient | DELETE, INSERT, REFERENCES, SELECT, TRIGGER, TRUNCATE, UPDATE
rol_medic | personal_medical | DELETE, INSERT, REFERENCES, SELECT, TRIGGER, TRUNCATE, UPDATE
rol_rezident | fisa_medicala | SELECT
(7 rows)

referat_db#

```

Figura 4.5: Verificare Roluri

4.2 Discretionary Access Control

Acest tip de access control permite proprietarului unui obiect (tabel, înregistrare) să decidă direct cine are acces. Permișunile sunt acordate individual fiecărui utilizator, oferind control granular asupra accesului la date.

Cum se poate observa, după ce medicul a primit rolul de SELECT access pe ”fisa_medicala” cu opțiune de a da mai departe, userul ”elena.constantinescu” (asistent) a primit permisiunea de SELECT pe același tabel și se poate observa că a primit accesul.

```

[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> GRANT SELECT ON *.* TO "elena.popescu";
Query OK, 0 rows affected (0.000 sec)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 1 | 1 | 1 | Hipertensiune arteriala | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 2 | 1 | 1 | Diabet tip 2 | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 3 | 1 | 1 | Infecție respiratorie superioară | Antibiotice 500mg de 3 ori pe zi timp de 7 zile | 2024-01-25 08:30:00 | 3 |
+-----+-----+-----+-----+
(3 rows)

referat_db> SELECT * FROM pacient;
+-----+
| id_pacient |
+-----+
| 1 |
+-----+
(1 row)

referat_db> SELECT * FROM medic;
+-----+
| id_medic |
+-----+
| 1 |
+-----+
(1 row)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 1 | 1 | 1 | Hipertensiune arteriala | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 2 | 1 | 1 | Diabet tip 2 | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 3 | 1 | 1 | Infecție respiratorie superioară | Antibiotice 500mg de 3 ori pe zi timp de 7 zile | 2024-01-25 08:30:00 | 3 |
+-----+-----+-----+-----+
(3 rows)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 1 | 1 | 1 | Hipertensiune arteriala | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 2 | 1 | 1 | Diabet tip 2 | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 3 | 1 | 1 | Infecție respiratorie superioară | Antibiotice 500mg de 3 ori pe zi timp de 7 zile | 2024-01-25 08:30:00 | 3 |
+-----+-----+-----+-----+
(3 rows)

referat_db>
```

Figura 4.6: Testare DAC

4.3 Mandatory Access Control

Acest tip de access control este impus de sistem pe baza unor etichete de securitate și reguli stricte (în cazul acesta ”grad_acreditare” pentru personal și ”nivel_clasificare” pentru fișele medicale). Accesul este determinat de nivelul de clasificare al datelor și al utilizatorului, fără posibilitatea modificării de către utilizatori (prin intermediul RLS - Row Level Security).

După ce s-a rulat acces-ul pentru RLS, se poate vedea cum fiecare user poate să vadă doar înregistrările care au ”nivel_clasificare” mai mic sau egal cu propriul lor ”grad_acreditare”.

```

[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> GRANT SELECT ON *.* TO "elena.popescu";
Query OK, 0 rows affected (0.000 sec)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 1 | 1 | 1 | Hipertensiune arteriala | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 2 | 1 | 1 | Diabet tip 2 | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 3 | 1 | 1 | Infecție respiratorie superioară | Antibiotice 500mg de 3 ori pe zi timp de 7 zile | 2024-01-25 08:30:00 | 3 |
+-----+-----+-----+-----+
(3 rows)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 1 | 1 | 1 | Hipertensiune arteriala | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 2 | 1 | 1 | Diabet tip 2 | Litoterapie 10mg zilnic, monitorizare tensiune | 2024-01-25 08:30:00 | 3 |
| 3 | 1 | 1 | Infecție respiratorie superioară | Antibiotice 500mg de 3 ori pe zi timp de 7 zile | 2024-01-25 08:30:00 | 3 |
+-----+-----+-----+-----+
(3 rows)

referat_db>
[192.168.1.1] [elena@DESKTOP-3KJH1V1] ~
+ root => 10:00:49 p $432 -l "elena.popescu" -d referat_db
mysql> SELECT * FROM fisa_medicala;
+-----+-----+-----+-----+
| id_fisa | id_pacient | id_medic | diagnostic | tratament | data_consultatie | nivel_clasificare |
+-----+-----+-----+-----+
| 2 | 1 | 2 | Infecție respiratorie superioară | Antibiotice 400mg la noapte, infuzii de relaxare | 2024-01-25 08:30:00 | 2 |
+-----+-----+-----+-----+
(1 rows)

referat_db>
```

Figura 4.7: Testare MAC

Capitolul 5

Auditare

5.1 Trigger

Auditarea permite înregistrarea tuturor operațiunilor efectuate asupra datelor sensibile pentru a putea urmări și analiza accesul la informații. În PostgreSQL, aceasta poate fi implementată folosind trigger-uri care înregistrează automat fiecare operațiune (INSERT, UPDATE, DELETE) într-un tabel de audit.

Implementarea folosește un tabel "audit_log" care stochează:

- **Tabelul afectat;**
- **Tipul operației (INSERT, UPDATE, DELETE);**
- **Utilizatorul care a efectuat operația (session_user);**
- **Timestamp-ul acțiunii;**
- **Datele vechi și noi (pentru UPDATE);**
- **Adresa IP a utilizatorului;**

Trigger-ele sunt configurate pentru toate tabelele sensibile (fisa_medicala, pacient, personal_medical) și se declanșează automat la fiecare modificare.

```
referat_db>> INSERT INTO personal_medical (id_personal, nume, prenume, specializare, username_db, grad_acreditor) VALUES ('5', 'Dr. Popescu', 'Andrei', 'Cardiologie', 'and.popescu', 3);
INSERT 0 1
referat_db>>
```

Figura 5.1: Testare Insert

id	id_personal	username_db	grad_acreditor	date_modificare	ip
1	1	and.popescu	3	2023-12-26 20:24:41.968000	127.0.0.1
2	2	and.popescu	3	2023-12-26 20:24:41.968000	127.0.0.1

Figura 5.2: Verificare Audit

5.2 pgAudit

Se poate folosi extensia "pgAudit" pentru medii de producție, care oferă auditare detaliată la nivel de sesiune și obiect. Toate operațiunile PostgreSQL sunt înregistrate de pgAudit și pot fi analizate folosind pgAudit Analyze, care încarcă datele într-un schema de bază de date pentru analiză. [pgaudit_analyze_doc]

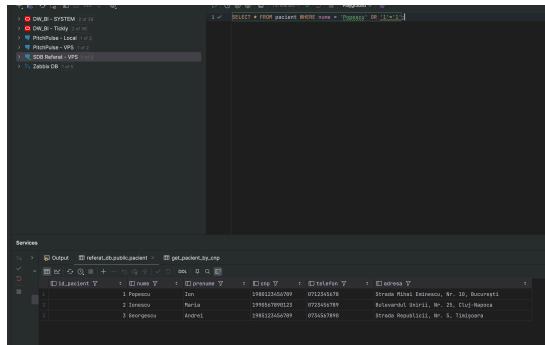
Beneficiile pgAudit includ auditare la nivel de sesiune (toate comenzi SQL), auditare la nivel de obiect (DDL, DML pe anumite tabele) și integrare cu sistemul de logare PostgreSQL. De asemenea, pgAudit Analyze permite analiză avansată. [pgaudit_analyze_doc]

Capitolul 6

Prevenirea SQL Injection

SQL Injection este printre cele mai frecvente atacuri asupra bazelor de date, în care un atacator injectează cod SQL fals în interogări prin concatenarea string-urilor. Soluția implică interogările parametrizate pregătite, care separă codul SQL de datele utilizatorului.

În PostgreSQL, se pot folosi prepared statements cu "PREPARE" și "EXECUTE", sau funcții PL/pgSQL care acceptă parametri. Acestea previne injectarea de cod SQL deoarece parametrii sunt tratați ca date, nu ca parte a comenzi SQL.



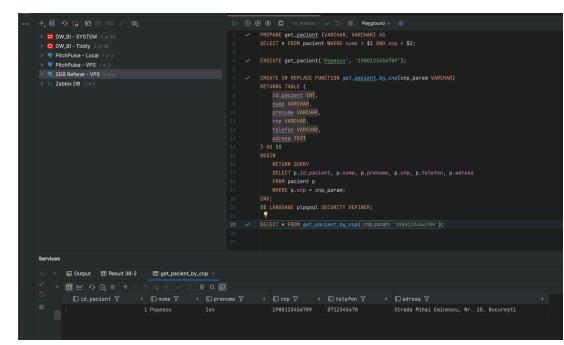
The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
SELECT * FROM patient WHERE name = 'Thomas' OR 1=1;
```

The results pane shows a single row with the following data:

id_patient	name	sex	prename	exp	telefon	adresa
1	Popescu	Ion	Maria	198812246789	072124678	Strada Mihai Eminescu, Nr. 10, București

Figura 6.1: Testare Injection



The screenshot shows the pgAdmin interface with a query editor containing the following SQL code:

```
CREATE OR REPLACE FUNCTION get_patient_by_exp(exp_patient VARCHAR) AS
$BODY$
    SELECT * FROM patient WHERE name = $1 AND exp = $2;
$BODY$;
CREATE OR REPLACE FUNCTION get_patient_by_name(name_patient VARCHAR) AS
$BODY$
    SELECT * FROM patient WHERE name = $1 AND exp = $2;
$BODY$;
```

The results pane shows a single row with the following data:

id_patient	name	sex	prename	exp	telefon	adresa
1	Popescu	Ion	Maria	198812246789	072124678	Strada Mihai Eminescu, Nr. 10, București

Figura 6.2: Verificare Injection

Capitolul 7

Criptarea Datelor

Criptarea datelor este necesară pentru protejarea informațiilor sensibile împotriva accesului neautorizat.

7.1 Full Disk Encryption - La nivel de disk

Una dintre cele mai bune metode de protejare a datelor este criptarea completă a discului sau a partii. Această tehnică asigură protecția atât a fiecărui fișier, cât și a stocării temporare care poate conține părți ale acestora. Nu trebuie să te îngrijorezi să alegi ce fișier vrei să protejezi, deoarece criptarea completă a discului protejează toate fișierele. [edb_security_2020]

Avem diverse opțiuni (depinde de OS-ul pe care îl folosim): Implementarea folosește un tabel "audit_log" care stochează:

- **MacOS - FileVault;**
- **Windows - BitLocker;**
- **Linux - LUKS (Linux Unified Key Setup).**



Figura 7.1: FileVault

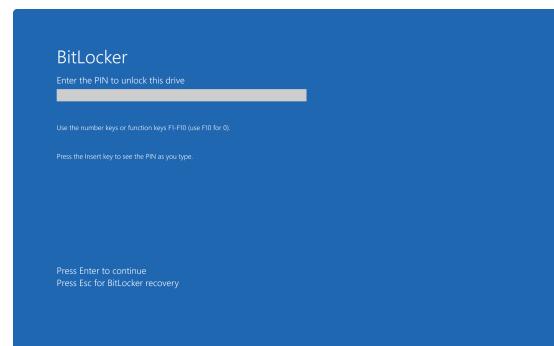


Figura 7.2: BitLocker

7.2 File System Encryption - La nivel de os

Criptarea la nivel de sistem de fișiere, denumită și criptare de fișiere sau directoare, este atunci când sistemul de fișiere însuși criptează fișierele sau directoarele. [edb_security_2020]

7.3 Criptarea la nivel de coloană

Pentru criptare la nivel de coloană, se poate folosi extensia "pgcrypto". Aceasta permite criptarea selectivă a coloanelor cu date sensibile, cum ar fi numerele CNP și cardurile de credit. [edb_security_2020]

```
RETURNS BYTEA AS $$  
BEGIN  
    RETURN pgp_sym_encrypt(cnp_plain, encryption_key);  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE OR REPLACE FUNCTION decrypt_cnp(cnp_encrypted BYTEA, encryption_key TEXT)  
RETURNS VARCHAR AS $$  
BEGIN  
    RETURN pgp_sym_decrypt(cnp_encrypted, encryption_key);  
END;  
$$ LANGUAGE plpgsql;  
  
INSERT INTO pacient_encrypted (nume, prenume, cnp_encrypted, telefon, adresa)  
VALUES (  
    'Popescu',  
    'Ion',  
    encrypt_cnp('1980123456789', 'portocal12'),  
    '0712345678',  
    'Strada Mihai Eminescu, Nr. 10, București'  
)  
  
SELECT  
    id_pacient,  
    nume,  
    prenume,  
    decrypt_cnp(cnp_encrypted, 'portocal12') AS cnp_decriptat,  
    telefon,  
    adresa  
FROM pacient_encrypted  
WHERE nume = 'Popescu';  
  
ALTER TABLE pacient  
ALTER COLUMN cnp TYPE BYTEA  
USING encrypt_cnp(cnp, 'portocal12');  
| %L to chat, %K to generate
```

Figura 7.3: encryption.sql

În cazul tabelelor pe care le avem, am creat o entitate nouă pentru a arăta diferența între a citi având cheia de decriptare și fără cheie.

După ce am creat uneltele necesare (verificați "encryption.sql" din imagine), mai jos se poate observa diferența când, după ce s-a introdus pacientul cu criptarea CNP-ului, se face citirea fără sau cu cheie.

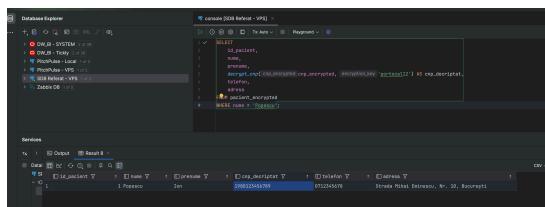


Figura 7.4: Cu cheie

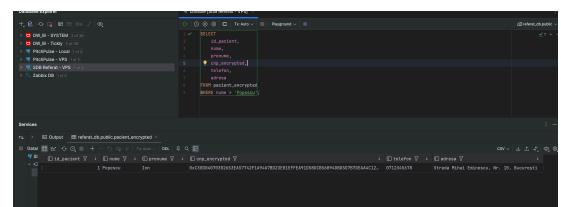


Figura 7.5: Fără cheie

Pentru tabelul curent "pacient", am criptat coloana prin comanda următoare:

```

1 ALTER TABLE pacient
2 ALTER COLUMN cnp TYPE BYTEA
3 USING encrypt_cnp(cnp, 'portocal12');

```

Listing 7.1: Migrarea coloanei CNP la format criptat (BYTEA)

Jos se poate observa coloana ”cnp” din tabelul ”pacient” criptat după rularea comenzi de mai sus:

The screenshot shows the pgAdmin interface with the 'pacient' table selected. The 'cnp' column is highlighted, showing its data type as 'BYTEA'. The table has three rows: 1. Pacient, Ion; 2. Dordea, Maria; 3. Stoica, Andreea. The 'cnp' values are displayed as long strings of characters.

	cnp	nume	prenume	telefon	adresa
1	0x5f505440475500871a0f0332f5a5d20201f3970d34810978803708813504fb_	Pacient	Ion	0773546768	Strada Mihai Eminescu, Nr. 10, Bucuresti
2	0x5f505440475500871a0f0332f5a5d20201f3970d34810978803708813504fb_	Dordea	Maria	0773546769	Bulevardul Unirii, Nr. 21, Cluj-Napoca
3	0x5f505440475500871a0f0332f5a5d20201f3970d34810978803708813504fb_	Stoica	Andreea	0773546769	Strada Radulescu Nr. 3, Timisoara

Figura 7.6: CNP Criptat

7.4 Criptare la nivel de rețea

Criptarea la nivel de rețea protejează datele în tranzit între client și server. În PostgreSQL, aceasta se realizează prin SSL/TLS. Am menționat anterior opțiunea ”hostssl” în ”pg_hba.conf” care necesită conexiuni SSL.

7.5 Criptarea parolelor

PostgreSQL folosește deja criptare pentru stocarea parolelor. Am menționat anterior folosirea ”scram-sha-256” în loc de ”md5” pentru autentificare.

Capitolul 8

Visualizări pentru Securitate

Views pot îmbunătăți securitatea datelor prin ascunderea structurii tabelelor și limitarea accesului. Putem face vizualizări care expun doar coloanele necesare și aplică filtrare automată în loc să acordăm permisiuni direct tabelelor.

```
psql: C:\Users\mihai\Documents\GitHub\referat\referat (1.0)
1 CREATE OR REPLACE VIEW pacient_public AS
2   SELECT
3     id_pacient,
4       nume,
5       prenume,
6       telefon
7   FROM pacient;
8
9 CREATE OR REPLACE VIEW personal_public AS
10  SELECT
11    id_personal,
12      nume,
13      prenume,
14      specializare
15  FROM personal_medical;
16
17 CREATE OR REPLACE VIEW fisa_medicala_personal AS
18  SELECT
19    f.id_fisa,
20    f.id_pacient,
21    p.nume AS nume_pacient,
22    p.prenume AS prenume_pacient,
23    f.diagnostic,
24    f.tratament,
25    f.data_consultatie
26  FROM fisa_medicala f
27  JOIN pacient p ON f.id_pacient = p.id_pacient
28  JOIN personal_medical pm ON f.id_medic = pm.id_personal
29  WHERE pm.username_db = current_user;
30
31 GRANT SELECT ON pacient_public TO rol_asistent;
32 GRANT SELECT ON personal_public TO rol_asistent;
33 GRANT SELECT ON fisa_medicala_personal TO rol_medic;
34
35
```

```
[XRM-SRV01] iamxorum@XRM-SRV01:~ ✓
→ psql -h 10.80.0.40 -p 5432 -U "mihai.marinescu" -d referat_db
Password for user mihai.marinescu:
psql (17.6 (Debian 17.6-0+deb13u1), server 16.11 (Debian 16.11-1.pgdg13+1))
Type "help" for help.

referat_db=> SELECT * FROM pacient;
ERROR: permission denied for table pacient
referat_db=> SELECT * FROM pacient_public;
id_pacient | nume | prenume | telefon
-----+-----+-----+-----
1 | Popescu | Ion | 0712345678
2 | Ionescu | Maria | 0723456789
3 | Georgescu | Andrei | 0734567890
(3 rows)

referat_db=>
```

Figura 8.2: Testare Visualizare

Figura 8.1: Creare Visualizări

Exemplu În loc să acordăm ”SELECT” pe întregul tabel ”pacient”, creăm un view ”pacient_public” care expune doar nume, prenume și telefon, ascunzând CNP-ul și adresa completă. (vedeți mai sus la imaginea 8.2)

Capitolul 9

Concluzii

Acest referat a demonstrat implementarea practică a multiple straturi de securitate în PostgreSQL, utilizând o bază de date medicală ca studiu de caz. Principalele concluzii sunt:

- **Securitatea rețelei:** prin configurarea corectă a "pg_hba.conf" și utilizarea metodelor de autentificare sigure precum "scram-sha-256";
- **Controlul accesului:** poate fi implementat prin trei paradigmă complementare (RBAC, DAC, MAC);
- **Auditare:** pentru urmărirea și analiza accesului la date sensibile, iar implementarea prin trigger-uri;
- **Prevenirea SQL Injection:** prin utilizarea prepared statements;
- **Criptarea:** la multiple niveluri (disc, coloană, rețea);
- **Visualizări:** pentru ascunderea structurii bazei de date și limitarea accesului la date sensibile.