



**UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET**



SIGURNOST POSTGRESQL BAZE PODATAKA
-Seminarski rad-

Mentor:
Prof. dr. Aleksandar Stanimirović

Student:
Sena Savić, br.ind. 1570

Niš, 2023.

SAŽETAK

Sigurnost baza podataka, uključujući i PostgreSQL bazu podataka, je neizostavni element poslovanja u današnje vreme. Sam pojam sigurnosti baze podataka, odnosi se na sve vidove bezbednosti, uključujući opštu sigurnost baze podataka, jednako kao i naprednije tehnike i metode.

U seminarskom radu „Sigurnost PostgreSQL baze podataka“, kroz pet poglavlja, prikazani su koncepti kojima se PostgreSQL baza podataka služi, kako bi postigla odgovarajući nivo zaštite i sigurnosti. Rad je koncipiran tako da se u uvodnom poglavlju, na koncizan način pruža uvid u strukturu seminarskog rada, ali i razloge detaljnog opisa ostalih poglavlja, kao i važnosti sigurnosti za baze podataka. Poglavlje posvećeno opštoj sigurnosti bavi se opisom i definicijom pojma sigurnosti baza podataka, ali i najčešćim pretnjama za sistem baza podataka, kao i razlozima zašto je važno očuvati sigurnost. Naredno poglavlje, fokusira se konkretno na proces postizanja sigurnosti kod PostgreSQL baze podataka i sadrži opise i primere važnih metoda, poput sigurnosti na mrežnom nivou, autentifikacionih tehnika, bezbednosti na nivou baze podataka i enkripcije. Finalno poglavlje bavi se opisom i savetima šta je od opisanih tehnika, najbolje primeniti u praksi, kako bi se garantovala bezbednost PostgreSQL baze podataka.

Poglavlje koje predstavlja zaključak, sumira celokupni prikaz seminarskog rada, ponovo ukazujući na to koliko je važna i kompleksna tema sigurnosti PostgreSQL baze podataka.

SADRŽAJ

1.	UVOD	4
2.	SIGURNOST BAZA PODATAKA	5
2.1	Formalna definicija pojma sigurnosti baza podataka.....	5
2.2	Najčešće pretnje po sigurnost baze podataka i važnost sigurnosti baza podataka	5
3.	SIGURNOST KOD POSTGRESQL BAZE PODATAKA	7
3.1	Sigurnost na mrežnom nivou (Network- Level Security)	7
	Unix Domain Soketi (UDS)	7
	TCP/IP soketi	8
	Firewalls.....	9
3.2	Autentifikacija.....	10
	pg_hba.conf fajl	11
	Trust autentifikacija	11
	Autentifikacija korišćenjem šifre	11
	Peer i Ident	12
	Kerberos	12
	TLS Sertifikati (Transport Level Security Certificates).....	13
3.3	Bezbednost na nivou PostgreSQL baze podataka	15
	Uloge.....	15
	Sigurnost na nivou redova.....	18
	Auditing	22
	Pogledi	24
3.4	Enkripcija.....	24
	Pgcrypto	25
	Funkcije za heširanje.....	25
	Enkripcija na nivou tabele.....	26
4.	PRAKTIČNI PREGLED KORIŠĆENIH TEHNIKA ZA POSTIZANJE SIGURNOSTI POSTGRESQL BAZE PODATAKA	29
5.	ZAKLJUČAK.....	31
6.	SPISAK KORIŠĆENE LITERATURE.....	32

1. UVOD

Sigurnost baze podataka predstavlja ključni aspekt u očuvanju poverljivosti, integriteta i dostupnosti podataka u različitim organizacijama.

Baze podataka, vrlo često sadrže osetljive informacije, poput ličnih podataka, finansijskih ili zdravstvenih informacija i samim tim su vrlo pogodne za izvršenje zlonamernih aktivnosti od strane organizacija ili pojedinaca. Ugrožavanje sigurnosti i bezbednosti baze podataka, može dovesti do ozbiljnih posledica, kao što su krađa identiteta, finansijske prevare, gubitak poverljivih informacija i mnoge druge.

Iz tog razloga sigurnost baza podataka treba da bude jedan od glavnih prioriteta svih organizacija koje rade i koriste poverljive podatke. Oblast sigurnosti baza podataka obuhvata različite aspekte, kao što su autentifikacija i autorizacija korisnika, enkripcija podataka, kontrola pristupa bazi podataka i upravljanje lozinkama, održavanje bezbedonosnih kopija i mnoge druge mere.

U seminarskom radu biće reči o sigurnosti baza podataka, razlozima i važnosti ove oblasti za sisteme baza podataka, zatim će biti dat detaljan uvid u mehanizme kojima se postiže sigurnost PostgreSQL baze podataka kako bi se osigurala dostupnost podataka, uključujući sigurnost na mrežnom nivou, autentifikacione mehanizme, bezbednost na nivou same baze i enkripciju. Na kraju rada dat je uvid u neke od najčešće korišćenih tehnika za postizanje sigurnosti kod PostgreSQL baze podataka, koje se primenjuju u praksi.

2. SIGURNOST BAZA PODATAKA

Sigurnost baza podataka obuhvata niz mera koje se koriste za obezbeđenje sistema za upravljanjem bazama podataka od zlonamernih napada i nezakonite upotrebe [1]. Bezbedonosni programi koji baze podataka koriste su dizajnirani tako da ne štite samo podatke u bazi podataka, već i sam sistem za upravljanjem podacima, ali i aplikacije koje koriste te podatke. Sigurnost baza podataka obuhvata različite tehnike i metode uspostavljanja bezbedonosnih mehanizama unutar samog sistema, ali i sistema povezanih sa konkretnom bazom.

2.1 Formalna definicija pojma sigurnosti baza podataka

„Sigurnost baze podataka odnosi se na kolektivne mere koje se koriste za zaštitu i obezbeđivanje baze podataka ili softvera za upravljanjem bazom podataka od nezakonitih upotreba ili sajber pretnji i napada [2]“.

Sa druge strane, „sigurnost baza podataka se odnosi na niz alata, kontrola i mera, dizajniranih za uspostavljanje i očuvanje poverljivosti, integriteta i dostupnosti baze podataka [3]“.

Bez obzira na spektar definicija sigurnosti baza podataka, ovaj pojam je izuzetno širok i uključuje mnoštvo procesa, alata i metodologija koje pružaju sigurnost u okruženju baze podataka.

2.2 Najčešće pretnje po sigurnost baze podataka i važnost sigurnosti baza podataka

U uvodnom delu seminarskog rada, rečeno je kako baze podataka skladište poverljive informacije, pa je neophodno da sistem garantuje sigurnost i integritet podataka koji se u njemu skladište. Međutim, neretko se dešava, da upravo baze podataka postanu mete različitih sajber napada ili budu ugrožene na razne načine, jednako zbog ranjivosti samog softverskog sistema koji upravlja sistemom baza podataka, kao i zbog pogrešne konfiguracije ili nepažnje od strane korisnika. Samim tim, sigurnost baze podataka je narušena, a neki od najčešćih uzroka su:

- Neovlašćeni pristup- ovaj tip narušavanja sigurnosti baze podataka, podrazumeva zlonamerne akcije korisnika ili hakera, koji pokušavaju da na nezakonit način dobiju pristup bazi podataka. Uglavnom se delovanje zlonamernih korisnika svodi na proces dešifrovanja slabih korisničkih imena i lozinke i njihovu zloupotrebu u razne svrhe. Ovakvi problemi mogu biti sprečeni primenom snažnih autentifikacionih i autorizacionih mehanizama, poput kreiranja snažnih lozinke i dvofaktorske autentifikacije.
- Ljudski faktor – najčešći način slabljenja sigurnosti baze podataka, upravo se vezuje za ljudski faktor. Vrlo često korisnici nenamerno obrišu ili oštete relevantne podatke u sistemu baze podataka, postavljaju slabe korisničke lozinke koje se lako dešifruju ili ih svesno podele sa drugim korisnicima, koji tada imaju neograničen pristup podacima.
- SQL/NoSQL injections- specifična pretnja po sigurnost baze podataka. Podrazumeva izvršenje zlonamernih SQL i NoSQL komandi (u zavisnosti od vrste baze podataka). Obično se u upit, ubaci zlonamerni string koji predstavlja SQL/NoSQL injekciju i prilikom izvršenja ovako formiranog upita se naruši bezbednost, međutim ovaj napad se može pravovremeno izbeći sanitizacijom ulaznih parametara, ili korišćenjem parametrizovanih upita i ograničenjem pristupa.
- Korišćenje zastarelog softvera- stare verzije softvera i operativnih sistema mogu sadržati ranjivosti, koje mogu biti iskorišćene za napade, a prevencija ovakvih scenarija se postiže redovnim ažuriranjem softvera i operativnog sistema.
- Malware- softver napisan tako da iskoristi ranjivost baze podataka ili bilo kog drugog sistema kome je namenjen. Ovako kreiran softver, može da dospe do baze podataka koristeći bilo koji uređaj koji je povezan na mrežu na kojoj se nalazi i sama baza.

Prevenција od ovakve vrste ugrožavanja sigurnosti se može postići visokim nivoom zaštite endpoint-a, pogotovu na serverima baze podataka.

- DoS napadi (eng. Denial of Service)- server baze podataka dobija veliku količinu lažnih zahteva, koji ga preplave, pa nije u stanju da obradi prave korisničke zahteve. Kao rezultat, sistem postaje nestabilan i vrlo često je sklon ka „rušenju“.

Pored nabrojanih uzroka destabilizacije sigurnosti baze podataka, postoji još dosta načina koji mogu doprineti ugroženju bezbednosti baze.

Samim tim, posmatrano iz ugla sistema baza podataka, sigurnost predstavlja izuzetno važnu kariku iz nekoliko razloga:

- Zaštita podataka – održavanje sigurnosti baze podataka je ključno radi zaštite osetljivih informacija koje se u bazi skladište, ali i radi sprečavanja neovlašćenog pristupa i krađe podataka.
- Usklađenost sa zakonskim propisima- niz sigurnosnih funkcija koje određene baze podataka pružaju, omogućavaju olakšanu integraciju i interoperabilnost skladištenih informacija u raznim zemljama.
- Zaštita integriteta podataka – sigurnost baza podataka utiče na visok nivo sprečavanja neželjenih promena podataka.
- Zaštita sistema- održavanje sigurnosti je vrlo važno kako bi se prevazišla bilo kakva ranjivost sistema, koju bi mogli da iskoriste hakeri ili drugi zlonamerni korisnici.

Iz svega navedenog, može se zaključiti koliko je visok stepen važnosti sigurnosti baza podataka i koliko je sama sigurnost relevantan element projektovanja i održavanja sistema koji skladišti poverljive informacije. Stoga je važno primeniti odgovarajuće mere, poput autentifikacionih procesa, autorizacije, enkripcije podataka, ali i praćenja aktivnosti korisnika kao i izvršiti adekvatnu obuku zaposlenih u vezi sa pravilima upravljanja sigurnošću podataka. Svakako, svaka baza podataka, podržava sebi svojstvene mehanizme kojima obezbeđuje i garantuje sigurnost podataka, a u nastavku seminarskog rada biće reči o tehnikama sigurnosti koje koristi PostgreSQL baza podataka.

3. SIGURNOST KOD POSTGRESQL BAZE PODATAKA

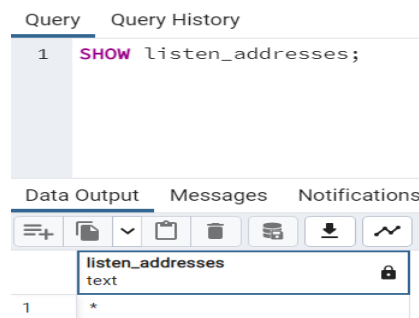
PostgreSQL je objektno-relaciona baza podataka otvorenog koda, koja predstavlja proširiv i izuzetno funkcionalan skladišteni sistem koji sadrži pregršt korisnih funkcija. Mnoge svetski poznate kompanije poput Netflix-a, Apple-a i Spotify-a, koriste upravo PostgreSQL, a vrlo često je ova baza podataka podrazumevani izbor za Web i mobilne aplikacije, gde je potrebno skladištiti velike količine vrednih i osetljivih podataka [4]. Iz tih razloga, neophodna je garancija sigurnosti PostgreSQL sistema, koja se može ostvariti na mrežnom nivou, korišćenjem autentifikacionih i autorizacionih mehanizama, enkripcionim tehnikama i tehnikama koje se fokusiraju na sigurnost na nivou same baze podataka.

Ovo poglavlje seminarskog rada, posvećeno je načinima i metodama kojima PostgreSQL baza podataka postiže sigurnost.

3.1 Sigurnost na mrežnom nivou (Network- Level Security)

Glavne komponente kojima PostgreSQL baza podataka postiže sigurnost na nivou mreže su Unix domain soketi, TCP/IP soketi i Firewall postavljen na mašini na kojoj se nalazi skladišteni sistem (na glavnoj mašini) [5].

Unix Domain Soketi (UDS) predstavljaju podrazumevanu funkciju koja omogućava povezivanje PostgreSQL baze podataka sa drugim procesima koji rade u okruženju zasnovanom na Unix-u [5]. UDS izgleda kao posebna datoteka na fajl sistemu i pristup se može izvršiti direktno sa mašine na kojoj je obavljena instalacija. Vlasnikom soketa smatra se operativni sistem koji PostgreSQL server koristi prilikom pokretanja. Podrazumevano, UDS-u su dodeljenje iste kontrole pristupa kao i svim ostalim datotekama na sistemu, iako on inicijalno zahteva samo dozvole za upis. Upravljanje UDS-om se se može postići korišćenjem konfiguracionih opcija poput `unix_socket_group` i `unix_socket_permissions`. Takođe, moguće je izmeniti i inicijalne kontrole pristupa u direktorijumu u kome se nalazi soket, kako bi se uticalo na promenu njegovih dozvola. PostgreSQL omogućava kreiranje više soketa korišćenjem opcije `unix_socket_directories`. Moguće je navesti nekoliko direktorijuma, gde svaki ima sopstvene dozvole, kako bi odgovarali zasebnim aplikacijama ili grupama korisnika [5]. UDS se vrlo često koriste u kombinaciji sa TCP/IP soketima, a iz ugla sigurnosti baze podataka, UDS su važni jer je inicijalno PostgreSQL konfigurisan da prihvata dolazne konekcije na bilo kojoj IP adresi, kao što je prikazano na narednoj slici.



Slika 1- Prikaz prihvatanja dolazne konekcije sa bilo koje dostupne IP adrese na serveru

Predstavljeno osluškivanje na svim mrežnim interfejsima, može biti jako rizično jer otvara mogućnost neovlašćenog upada zlonamernih korisnika ili aplikacija u sistem baze podataka, čime se ugrožava sigurnost. Zbog toga je preporučljivo ograničiti slušanje samo na IP adrese

koje su neophodne za rad baze podataka, kao što su lokalne IP adrese ili IP adrese aplikacija koje imaju dozvoljen pristup bazi podataka.

Ovo se konkretno kod Unix Domain soketa postiže podešavanjem samog soketa u `postgresql.conf` datoteci. U pomenutoj datoteci, neophodno je pronaći liniju koja sadrži IP adrese kojima je dozvoljeno osluškivanje i kako je ta linija u konfiguracionom fajlu, inicijalno zakomentarisana, potrebno je otkomentarisati je i dodati soket. Nakon toga, PostgreSQL server je neophodno restartovati i izvršiti konektovanje na bazu podataka korišćenjem soketa u odgovarajućem formatu. Opisani postupak prikazan je na narednim slikama.

```
#listen_addresses = 'localhost'      listen_addresses= 'localhost, /var/run/postgresql'
```

Slika 2 – Podešavanje Unix Domain soketa

```
host= /var/run/postgresql dbname= sistemibp user= student password= sifra
```

Slika 3- Konekcija na bazu korišćenjem Unix Domain soketa

TCP/IP soketi omogućavaju udaljenim sistemima pristup PostgreSQL serveru. Obično su potrebni kod aplikacija koje nude više različitih usluga, kao i za omogućavanje udaljenje administracije pomoću alata kao što je pgAdmin [5].

U slučaju bezbednosti i sigurnosti baze podataka, cilj je smanjenje područja potencijalnog napada za svakog ko pokuša da dobije pristup sistemu. Način postizanja sigurnosti zavisi u mnogome i od toga kako je server hostovan na mreži. Ukoliko se nalazi unutar korporativne mreže, može biti hostovan na više VLAN-ova ili fizičkih mreža, koje se mogu koristiti u različite svrhe [6]. Sistem treba da bude konfigurisan samo da sluša i prihvata veze na mrežama koje su zaista neophodne. Podrazumevano, izvorni kod PostgreSQL-a, formiran je tako da osluškuje samo na lokalnom hostu, međutim verije PostgreSQL-a koje su prethodno upakovane, imaju drugačiju konfiguraciju, pa je potrebno izvršiti proveru nakon instalacije. Proces provere se postiže na isti način kao i kod UDS-a (to je zato što se vrlo često kombinuju) i time se može uticati na pojačanje sigurnosti, jer se PostgreSQL bazi nameće da sluša i prihvata veze sa određenih mrežnih adresa[6].

Obezbeđenje TCP/IP konekcije se kod PostgreSQL baze postiže na još jedan način, pokretanjem bezbednog tunela preko ssh [7]. Ovaj proces se obavlja u dva koraka. U prvom koraku se uspostavlja tunel na mašini na kojoj se sistem nalazi, tako što se argumentom `-L`, navodi željeni broj porta koji predstavlja jedan kraj tunela, dok je drugi kraj tunela predstavlja broj porta koji PostgreSQL inicijalno koristi (5432). Između brojeva koji su navedeni kao krajevi tunela, nalazi se ime ili adresa koja pripada serverskoj mašini, kao i poslednji argument za ssh koji uključuje opciono korisničko ime. Bez navođenja ovog opcionog dela, ssh će pokušati da izvrši ovaj proces korišćenjem imena pod kojim je korisnik trenutno prijavljen. Nakon ovoga, sledi drugi korak, koji podrazumeva povezivanje klijenta sa lokalnim hostom, korišćenjem broja porta koji je naveden u prethodnom koraku. U ovom koraku vrlo je važno prilikom ostvarivanja konekcije, navesti argument `-h`, kako bi klijent iskoristio TCP/IP soket umesto Unix soketa. Argument porta se može izostaviti, ako je za drugi kraj tunela iskorišćen port 5432 [7]. U novijim verzijama PostgreSQL-a ovaj proces se može izvršiti direktno korišćenjem alata poput pgAdmin4.

```
ssh -L 3333:wit.msc.anl.gov:5432 postgres@wit.msc.anl.gov  
psql -h localhost -p 3333 -d mwp
```

Slika 4- Obezbeđenje TCP/IP konekcije kod PostgreSQL baze podataka

Slika 5- Obezbeđenje TCP/IP konekcije kod PostgreSQL baze podataka direktno iz pgAdmin4 alata

TCP/IP i Unix Domain soketi se mogu kombinovati, ali je važno istaći da ukoliko čvor na kome je pokrenut PostgreSQL server, koristi nekoliko mrežnih interfejsa, potrebno je izvršiti ograničenje kojim se serveru nameće komunikacija samo sa adresama koje klijenti koriste za povezivanje. To se postiže podešavanjem parametra `listen_addresses` [4]. Ukoliko su svi korisnici PostgreSQL baze podataka na istom mrežnom čvoru, sigurnije je onemogućiti slušanje TCP soketa. Server će tada koristiti i prihvatati samo mreže sa Unix Domain soketa, a ne sa mreže, pa je sistem sigurniji. Ovo se postiže navođenjem praznog reda umesto adrese u `listen_addresses` parametru.

```
listen_addresses= 'localhost, 192.168.0.1'
listen_addresses= ' '
```

Slika 6- Ograničenje adresa kod TCP/IP i Unix Domain soketa

Firewalls predstavljaju još jedan način kojim je moguće poboljšati sigurnost servera PostgreSQL baze podataka. U idealnom svetu, PostgreSQL server bi bio potpuno izolovan i ne bi dozvoljavao nikakve dolazne veze, međutim ova vrsta podešavanja nije nešto što PostgreSQL podržava [8]. Iz tog razloga, korišćenjem firewall-a, postiže se zaključavanje pristupa na nivou porta čvora na kome baza podataka radi, čime se povećava sigurnost same baze podataka [8]. Po defaultu, PostgreSQL baza podataka osluškuje konekcije na TCP portu 5432. Tipično, firewall-i će omogućiti definisanje ulaznih i izlaznih pravila koja određuju dozvoljeni saobraćaj. Ova pravila u razmatranje pored porta, uzimaju i mrežni protokol (TCP ili Ipv6), kao i izvorišnu adresu odakle dolazi konekcija [6]. U zavisnosti od operativnog sistema, postoje različiti načini blokiranja porta. Operativni sistem Windows koristi Windows Defender Firewall, dok operativni sistem Linux koristi iptables funkcionalnosti za regulisanje sigurnosti kod PostgreSQL baze podataka.

Bez obzira o kom se operativnom sistemu ili firewall-u radi, ono što je primarni cilj obezbeđivanja sigurnosti jeste smanjenje pristupa PostgreSQL-u. Samim tim, blokiranje porta podrazumeva da se definiše skup pravila kojim se odbija sav saobraćaj koji dolazi na defaultni PostgreSQL port, ukoliko ne dolazi sa adrese servera aplikacije sa kojom se očekuje komunikacija. Izvorna adresa saobraćaja može biti i lista adresa ali i podmreža, kojima je pristup dozvoljen. Takođe, ukoliko server ima instalirane data wrapere (eng. data wrappers) ili slične ekstenzije, poželjno je definisati unapred odlazna pravila kako bi se sprečilo njihovo povezivanje sa drugim skupom servera, osim sa prethodno definisanim.

Kod Linux operativnog sistema, kao što je već rečeno, obezbeđenje sigurnosti postiže se `iptables` `firewall` funkcionalnostima i ovaj proces je nešto složeniji od procesa konfiguracije Windows Defender-a kod Windows operativnog sistema.

```
iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT
iptables -A INPUT -p tcp -m state --state NEW --dport 5432 -j ACCEPT
```

Slika 7- Podešavanje validne konekcije, SSH i dozvola korišćenja PostgreSQL-a korišćenjem iptables kod Linux-a

```
iptables -A OUTPUT -j ACCEPT
iptables -A INPUT -j DROP
iptables -A FORWARD -j DROP
```

Slika 8- Dozvola odlaznog saobraćaja i blokiranje svega ostalog

Pravilima prikazanim na slikama, PostgreSQL omogućava sav odlazni saobraćaj, ali takođe pruža mogućnost povezivanja bilo kog korisnika na port 5432. Firewall ima opciju striktnije sigurnosti i podrazumeva prihvatanje konekcija samo sa određenih mreža ili podmreža.

```
iptables -A INPUT -p tcp -m state --state NEW --dport 5432 -s 192.168.1.0/24 -j ACCEPT
```

Slika 9- Dozvola pristupa PostgreSQL portu samo sa lokalne podmreže

Prethodno prikazanom komandom, sprečava se konektovanje na podrazumevani PostgreSQL port. Međutim, ponekad je za ovaj pristup obezbeđenja sigurnosti kod PostgreSQL baze, neophodna uloga tzv. „lokalnog agenta“ [4]. Ovaj lokalni agent, uspostavlja vezu sa klijentskim čvorovima i ponaša se kao proxy za raspodelu saobraćaja do lokalne PostgreSQL instance. Ovaj metod je poznat kao metod „obrnutog tunelovanja“ i podrazumeva isti proces kreiranja tunela, prethodno opisanog kod TCP/IP soketa.

Konfigurisanje firewall-a za pružanje sigurnosti kod PostgreSQL baze podataka, na Windows operativnom sistemu je nešto jednostavnije nego na drugim operativnim sistemima. Ukoliko se PostgreSQL koristi na Windows operativnom sistemu, moguće je iskoristiti ugrađeni Windows Defender Firewall, i upravljanje njime se može izvršiti ili preko grafičkog interfejsa ili preko komandne linije. U narednom primeru, na Windows operativnom sistemu, korišćenjem firewall-a, dozvoljen je pristup PostgreSQL bazi podataka samo sa navedene IP adrese i to samo preko podrazumevanog porta baze podataka. Ovako konfigurisan firewall, osigurava PostgreSQL bazu podataka, tako što dozvoljava pristup samo određenim IP adresama, čime se smanjuje rizik od neovlašćenog pristupa i krađe podataka.

```
netsh advfirewall firewall add rule name= "PostgreSQL Port" dir= in action= allow protocol= TCP localport=5432 remoteip= 192.168.1.100
```

Slika 10- Konfigurisanje Windows Firewall-a i obezbeđenje sigurnosti PostgreSQL baze podataka

3.2 Autentifikacija

Autentifikacija je proces kojim server baze podataka uspostavlja identitet klijenta i utvrđuje da li je klijentskoj aplikaciji dozvoljeno da se poveže sa bazom podataka, korišćenjem zahtevanog korisničkog imena [9].

PostgreSQL nudi različite metode autentifikacije klijenata, a koji metod će se koristiti zavisi od klijentske adrese hosta, baze podataka i korisnika. Korisnička imena koja se zahtevaju prilikom autentifikacije, su kod PostgreSQL baze podataka logički odvojena od korisničkih imena operativnog sistema na kome radi server baze podataka. Razlog ovakvoj logičkoj odvojenosti korisničkih imena, rezultat je činjenice da veliki broj korisnika baze podataka, može pristupiti bazi preko udaljene veze i samim tim takvi korisnici nemaju nalog lokalnog operativnog sistema [9].

Autentifikacija je jedan od važnih koncepata sigurnosti baze podataka i kod PostgreSQL baze podataka postoji nekoliko tehnika koje se mogu koristiti kako bi se na adekvatan način obezbedili ovi mehanizmi.

Ukoliko se govori o klijentskoj autentifikaciji kod PostgreSQL baze, prvi pojam na koji se nailazi je pg_hba.conf fajl.

pg_hba.conf fajl je konfiguracioni fajl koji kontroliše klijentsku autentifikaciju i čuva se u klasteru baze podataka, u okviru direktorijuma podataka. Instalira se kada se direktorijum podataka inicijalizuje `initdb-om` [9]. Generalni format ovog fajla je set slogova, gde svaka posebna linija predstavlja jedan slog. Blanko linije i karakteri iza karaktera `#` se ignorišu, a jedan slog se može nastaviti i u narednoj liniji korišćenjem `/`. Ovim slogovima se definišu pravila pristupa i metode autentifikacije klijenta na server baze podataka. Linije tj. slogovi u datoteci se obrađuju sekvencijalno kada se uspostavi veza, a prvi slog koji odgovara svojstvima veze koristi se za određivanje metoda autentifikacije [6]. Postoji sedam različitih formata za linije u datoteci, od kojih su glavne tri varijante, a ostale imaju strukturu formiranu kombinacijom prethodnih linija, samo sa drugačijim tipom veze u prvom polju. Pored tipa veze u prvom polju, svaki zapis dalje sadrži opseg IP adresa, ime baze, ime korisnika i autentifikacioni metod.

<code>local</code>	<code>database</code>	<code>user</code>	<code>auth-method</code>	<code>[auth-options]</code>		
<code>host</code>	<code>database</code>	<code>user</code>	<code>address</code>	<code>auth-method</code>	<code>[auth-options]</code>	
<code>hostssl</code>	<code>database</code>	<code>user</code>	<code>address</code>	<code>auth-method</code>	<code>[auth-options]</code>	
<code>hostnossl</code>	<code>database</code>	<code>user</code>	<code>address</code>	<code>auth-method</code>	<code>[auth-options]</code>	
<code>hostgssenc</code>	<code>database</code>	<code>user</code>	<code>address</code>	<code>auth-method</code>	<code>[auth-options]</code>	
<code>hostnogssenc</code>	<code>database</code>	<code>user</code>	<code>address</code>	<code>auth-method</code>	<code>[auth-options]</code>	
<code>host</code>	<code>database</code>	<code>user</code>	<code>IP-address</code>	<code>IP-mask</code>	<code>auth-method</code>	<code>[auth-options]</code>
<code>hostssl</code>	<code>database</code>	<code>user</code>	<code>IP-address</code>	<code>IP-mask</code>	<code>auth-method</code>	<code>[auth-options]</code>
<code>hostnossl</code>	<code>database</code>	<code>user</code>	<code>IP-address</code>	<code>IP-mask</code>	<code>auth-method</code>	<code>[auth-options]</code>
<code>hostgssenc</code>	<code>database</code>	<code>user</code>	<code>IP-address</code>	<code>IP-mask</code>	<code>auth-method</code>	<code>[auth-options]</code>
<code>hostnogssenc</code>	<code>database</code>	<code>user</code>	<code>IP-address</code>	<code>IP-mask</code>	<code>auth-method</code>	<code>[auth-options]</code>

Slika 11 – Prikaz formata sloga `pg_hba.conf` fajla kod PostgreSQL baze podataka¹
 Formati `local` i `host`, su najčešći formati ostvarivanja bezbedne konekcije, gde se prvi odnosi na povezivanje korišćenjem Unix Domain soketa, a drugi na formiranje konekcije preko TCP/IP-a. Treći format, `hostssl` je autentifikacioni slog za povezivanje preko TCP/IP-a, ali jedino kada je konekcija ostvarena SSL enkripcijom. Primeri `local` i `host` formata dati su u nastavku, gde svaki od formata respektivno, koriste `scram-sha-256` i `md5` kao autentifikacione metode.

```
local sistemibp korisnik scram-sha-256
host sistemibp korisnik 172.16.253.47/32 md5
```

Slika 12- Prikaz `local` i `host` formata za autentifikaciju `pg_hba.conf` fajla kod PostgreSQL baze podataka

Trust autentifikacija je autentifikacioni mehanizam koji se koristi samo u izuzetnim okolnostima, jer kod PostgreSQL baze podataka, omogućava klijentu da se poveže sa serverom bez dalje provere autentičnosti. Ovaj autentifikacioni mehanizam je koristan za razvoj i testiranje aplikacija na lokalnoj mašini, kada se konekcija vrši preko UDS soketa i pristup imaju samo korisnici koji su apsolutno pouzdani, a bezbednost podataka ne predstavlja problem.

Autentifikacija korišćenjem šifre kod PostgreSQL baze podataka se bazira na nekoliko metoda, koje funkcionišu prilično slično, ali se razlikuju u načinu čuvanja samih šifri i njihovoj distribuciji kroz mrežu. Metodi poput `md5` i `scram-sha-256` su opisani kao deo autentifikacionog mehanizma `pg_hba.conf`. `md5` metoda sprečava tzv. „njuškanje lozinki“ i izbegava čuvanje lozinki na serveru, ali ne pruža dublju zaštitu ukoliko napadač uspe da ukrade heš vrednost lozinke. Ne može se koristiti bez `db_user_namespace` funkcije PostgreSQL baze. Bezbedniji metod je `scram-sha-256`, koja podržava sve što i `md5` metod, ali dozvoljava skladištenje lozinki

¹ Izvor slike 11- <https://www.postgresql.org/docs/current/auth-pg-hba-conf.html>

na serveru u kriptografski heširanom obliku. Trenutno je najbezbedniji autentifikacioni metod ovog tipa, ali ga starije verzije PostgreSQL baze podataka ne podržavaju. Autentifikacija korišćenjem samo šifre je najslabiji metod, jer predstavlja čist tekstualni format i izuzetno je kao takav podložan napadima. Ipak, moguće je i ovako formiranu šifru korisnika koristiti na bezbedan način, ukoliko se konekcija zaštiti SSL enkripcijom[10]. Šifre PostgreSQL baze podataka su odvojene od korisničkih šifri operativnog sistema na kome se baza nalazi. Šifre koje se kreiraju na klasičan način čuvaju se u pg_audit sistemskom katalogu i njima se može upravljati korišćenjem naredbi CREATE i ALTER USER.

```
CREATE USER korisnik WITH PASSWORD 'passwordKorisnik'; SET ROLE korisnik;
SELECT 1;
```

	?column? integer	
1		1

Slika 13- Prikaz kreiranja korisnika i postavljanja šifre klasičnom metodom kao i rezultata provere postojanosti korisnika sa zadatom šifrom

Prikazani koraci na slici su neophodni, jer PostgreSQL baza nema direktan način kojim se može proveriti da li je korisnik kreiran i koja mu je lozinka. To se može postići prilikom naredne konekcije na bazu ili kao što je prikazano na slici, prilikom izvršenja neke akcije koja zahteva autentifikaciju. Ukoliko je akcija uspešno izvršena, autentifikacija je odgovarajuća i nema pretnji po sigurnost baze podataka.

Peer i Ident su takođe autentifikacioni metodi, koji omogućavaju autentifikaciju korisnika od strane operativnog sistema i mnogi PostgreSQL paketi dolaze unapred konfigurisani da koriste peer metod. Peer metod je dostupan samo za lokalne veze i kada se koristi, server dobija korisničko ime klijenta od operativnog sistema i pritom proverava da li to ime odgovara traženom korisničkom imenu iz baze podataka[6]. Metod potvrde identiteta (eng. ident) dostupan je samo za mrežne veze. Funkcioniše na sličan način kao i peer autentifikacija, osim što se oslanja na ident server kako bi potvrdio korisničko ime, međutim iz ugla totalne sigurnosti ne treba se previše uzdati u ovaj metod, jer se pokretanje vrši na klijentskoj strani, pa samim tim već nije previše pouzdan [6].

Kerberos autentifikacija se oslanja na Kerberos infrastrukturu za autentifikaciju korisnika, tako što korisnici dobijaju Kerberos tikete koji se koriste za kasniju potvrdu od strane PostgreSQL baze. Kerberos se vrlo često koristi u korporativnim okruženjima i integrisan je sa sistemima za jednokratnu prijavu [6]. U takvim sistemima, PostgreSQL server je konfigurisan da autentifikuje korisnike preko Kerberos infrastrukture, što automatski isključuje metod potvrde kreiranjem šifre. Kerberos autentifikacija je dostupna putem gssapi metode autentifikacije u PostgreSQL-u i smatra se izuzetno sigurnim metodom, uključujući i automatsku autentifikaciju za klijentski softver koji ga podržava.

Host name/addresses	testServer
Port	5432
Maintenance database	postgres
Username	postgres
Kerberos authentication?	<input checked="" type="checkbox"/>

Slika 14- Prikaz Kerberos autentifikacije kod PostgreSQL baze podataka

TLS Sertifikati (Transport Level Security Certificates) koriste se kao deo autentifikacionog mehanizma Transport Level Security protokola, koji služi da obezbedi bezbednu komunikaciju preko Interneta. Izvorno, ovaj protokol je podržan od strane PostgreSQL-a i koristi se za bezbedan pristup korisnika bazi podataka. Da bi se postigla pomenuta sigurnost, neophodno je da postoje ključ i sertifikat, koji se mogu uneti manuelno ili automatski, pisanjem skripte koja koristi konfiguracioni parametar PostgreSQL-a `ssl_passphrase_command`. Pored ovog parametra važno je proveriti i parametre koji se odnose na SSL šifrovanje: `ssl_ciphers`, `ssl_min_protocol_version`, `ssl_dh_params_file` i `ssl_ecdh_curve` [5].

Autentifikacija sertifikatima funkcioniše, na taj način što sistem veruje sertifikatu najvišeg nivoa (eng. root certificate) ili neke od njegovih direktnih potomaka, kako bi se klijentima izdao sertifikat od poverenja. Klijenti koji poseduju sertifikat i ključ, koji je izdat od sertifikata višeg nivoa, mogu se smatrati bezbednim. Primer opisanog postupka, podrazumeva da se prvo kreiraju sertifikat i ključ autoriteta za izdavanje sertifikata (eng. Certificate Authority), koji zbog visokog stepena osetljivosti zahtevaju čuvanje cele konfiguracije u potpunosti. Nakon ovoga, kreira se sertifikat i ključ za PostgreSQL server, koji se potpisuje korišćenjem prethodno kreiranog autoriteta za izdavanje sertifikata. Na kraju se i sertifikat i ključ servera instaliraju na PostgreSQL serveru, zajedno sa kopijom sertifikata koji predstavlja autoritet. Tada je moguće kreirati i klijentske sertifikate i ključeve, koji se po potrebi mogu potpisati sertifikovanim autoritetom. Kada se klijent poveže sa PostgreSQL-om i koristi TLS sertifikate kao autentifikacioni metod, PostgreSQL server će prvo proveriti da li je klijentov sertifikat pouzdan i da li Common Name polje sertifikata odgovara korisničkom imenu klijenta. Mapiranje korisničkih imena se postiže na isti način kao i kod peer i ident metoda. Dodatnu zaštitu od lažiranja sertifikata, pruža mogućnost navođenja brojnih opcija koje utiču na verodostojnost klijentskih sertifikata, čime se postiže viši nivo sigurnosti. Sertifikati predstavljaju idealan način za autentifikaciju automatizovanih sistema koji koriste PostgreSQL.

Opisani proces kreiranja sertifikata i konfiguracije PostgreSQL-a prikazan je na narednim slikama.

```
openssl req -sha256 -new -x509 -days 365 -nodes \
-out server-ca.crt \
-keyout server-ca.key

openssl req -sha256 -new -nodes \
-subj "/CN=postgres.primersistemiBP.com" \
-out server.csr \
-keyout server.key

openssl x509 -req -sha256 -days 365 \
-in server.csr \
-CA server-ca.crt \
-CAkey server-ca.key \
-CAcreateserial \
-out server.crt
```

```
openssl req -sha256 -new -x509 -days 365 -nodes \
-out client-ca.crt \
-keyout client-ca.key

openssl req -sha256 -new -nodes \
-subj "/CN=korisnik" \
-out client.csr \
-keyout server.key

openssl x509 -req -sha256 -days 365 \
-in client.csr \
-CA client-ca.crt \
-CAkey client-ca.key \
-CAcreateserial \
-out client.crt
```

Slika 15- Prikaz kreiranja serverskog i klijentskog TLS sertifikata
Sertifikati prikazani na slici, kreirani su lokalno. Za oba sertifikata navedeno je vreme za koje su važeći. Klijentski sertifikat, biće korišćen od strane korisnika kada se budu povezivali na PostgreSQL bazu podataka, kao vid autentifikacionog mehanizma. Ono što je važno uočiti i znati prilikom kreiranja klijentskog sertifikata, jeste da u polju CN treba da stoji ime uloge baze na koju se korisnik povezuje, jer će upravo to ime PostgreSQL iskoristiti za proveru identiteta

klijenta. Kada su sertifikati kreirani, konfigurirše se i PostgreSQL server kako bi prihvatio TLS konekciju. To se postiže podešavanjem parametara postgresql.conf fajla.

```
ssl = on
ssl_ca_file = 'client-ca.crt'
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_ecdh_curve = 'prime256v1'
ssl_min_protocol_version = 'TLSv1.3'
```

Slika 16- Prikaz konfigurisanja najvažnijih parametara postgresql.conf fajla za prihvatanje TLS konekcije

```
psql "host=postgres.PrimerSistemiBP.com \
user=korisnik \
dbname=postgres \
sslmode=verify-full \
sslrootcert=server-ca.crt \
sslcert=client.crt \
sslkey=client.key"
```

Slika 17- Prikaz konekcije klijenta korišćenjem validnog sertifikata
Nakon svih preduzetih koraka, korisnici se mogu konektovati na bazu samo navođenjem imena servisa i imena baze na koju se konektuju.

```
psql "service=PrimerSistemiBP dbname=postgres"
```

Slika 18- Prikaz konekcije na bazu

PostgreSQL nudi i mogućnost manuelne konfiguracije autentifikacije klijenta, korišćenjem pgAdmin alata. Od svih polja koja je potrebno popuniti na taj način, jedino koje se ne sme izostaviti je polje Root Certificate, jer se ono koristi za verifikaciju servera. Svakako, korišćenje ovih polja nema smisla, ukoliko nisu kreirani sertifikati.

SSL mode	Prefer	▼
Client certificate	C:\Users\korisnik\client\ca.crt	📁
Client certificate key	C:\Users\korisnik\server.key	📁
Root certificate	C:\Users\korisnik\server-ca.crt	📁
Certificate revocation list		📁

Slika 19- Manuelno podešavanje TLS-a kod PostgreSQL baze podataka kroz pgAdmin alat

3.3 Bezbednost na nivou PostgreSQL baze podataka

Sigurnost kod PostgreSQL baze podataka, se pored prethodno opisanih načina, može postići i na nivou same baze, primenom sigurnosnih mera kao što su kreiranje uloga i dozvola, bezbednost na nivou redova i proces revizije.

U prethodnim delovima seminarskog rada, bilo je reči o načinu zaštite servera baze podataka od neovlašćenog pristupa, kao i o procesu autentifikacije korisnika prilikom pristupa bazi, međutim postavlja se pitanje kontrole pristupa nakon što korisnici ostvare sigurnu konekciju sa bazom, odnosno šta je određenim korisnicima dozvoljeno da rade sa podacima koji su sačuvani u bazi. Ovaj koncept se naziva autorizacijom ili kontrolom pristupa [8]. PostgreSQL dolazi sa ugrađenim sistemom korisničkih dozvola, dizajniranih oko koncepta uloga.

Uloge se od PostgreSQL verzije 8.1, definišu kao „korisnici“, odnosno ova dva pojma se izjednačuju. To znači da se ime naloga baze podataka, tretira kao uloga sa atributom LOGIN, koji omogućava povezivanje sa bazom podataka [5]. Iz tog razloga, sledeće SQL komande imaju isto značenje kod PostgreSQL baze podataka.

```
CREATE USER student;  
CREATE ROLE student LOGIN;
```

Slika 20- Kreiranje novog korisnika/uloge u PostgreSQL bazi podataka
Obe komande kreiraju novog korisnika, što direktno ukazuje na to da su „korisnik“ i „uloga“ sinonimi u PostgreSQL bazi podataka. Jedina razlika između ove dve komande je u privilegijama, tačnije prva komanda kreira korisnika koji ima samo osnovne interakcije sa bazom podataka (čitanje/upis), dok se drugom komandom kreiranom korisniku omogućava autentifikacija ka bazi korišćenjem korisničkog imena i lozinke, čime se korisniku pružaju naprednije privilegije. Pored privilegije koju pruža atribut LOGIN, uloge mogu imati i druge attribute koji utiču na kreiranje specifičnih dozvola, poput atributa SUPERUSER (podreazumevani super user u PostgreSQL-u je postgres), koji omogućava zaobilazanje svih provera dozvola. Kada se govori o dozvolama koje se mogu dodeliti ulogama, one se kod PostgreSQL baze podataka mogu podeliti u dve grupe- dozvole koje dopunjuju članstvo privilegija vezanih za određenu ulogu i privilegije objekata baze podataka [8]. Kada se kreira objekat u bazi podataka (npr. tabela), dodeljuje se vlasnik. Vlasnik objekta je zapravo uloga koja je izvršila upit o kreiranju i za većinu kreiranih objekata inicijalno stanje je da samo vlasnik može da modifikuje kreirani objekat [11]. Da bi se drugim ulogama omogućilo korišćenje kreiranog objekta, moraju se dodeliti privilegije. Privilegije koje se primenjuju nad određenim objektom variraju u zavisnosti od tipa objekta, ali najčešće korišćene su ALTER, GRANT i REVOKE. ALTER komandom se kreirani objekat može dodeliti novom vlasniku, pri čemu superuser-i uvek mogu da stave novi objekat pod svoju kontrolu, dok kod običnih uloga to zavisi od toga da li su trenutni vlasnici objekta ili se nalaze u okviru grupe nove vlasničke uloge [11].

```
ALTER TABLE proizvod OWNER TO korisnik;
```

Slika 21- Format ALTER naredbe i prikaz njenog korišćenja za dodelu vlasništva ulozi korisnik nad tabelom proizvod

Za eksplicitnu dodelu privilegija koristi se komanda GRANT. Ova komanda ima dve osnovne varijante, koje odgovaraju podeli dozvola uloga, gde se prva varijanta ove komande odnosi na davanje privilegija objektima (tabelama, kolonama, itd...), a druga na dodelu članstva uloga.

Varijanta komande GRANT, kojom se daju privilegije objektima u bazi podataka, odnosi se na jednu ili više uloga koje su kreirane. Ukoliko se iskoristi ključna reč PUBLIC, sve privilegije koje se dodeljuju objektu, biće dodeljene svim ulogama, čak i onim koje se tek kasnije kreiraju. Ukoliko se iskoristi parametar WITH GRANT, primalac privilegije, tu istu privilegiju može dodeliti drugim ulogama. Ova prva varijanta se vezuje za objekete, jer se smatra da nema potrebe dodeljivati privilegije vlasnicima objekta, s obzirom da se podrazumeva da on već prethodno poseduje sve privilegije. Neke od najčešće korišćenih privilegija sa narednom GRANT su : SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE...

GRANT INSERT ON proizvod TO PUBLIC;

Slika 22- Prikaz primera GRANT naredbe dodele privilegije INSERT svim korisnicima nad objektom(tabelom) PostgreSQL baze podataka

```
SELECT grantee, privilege_type
FROM information_schema.table_privileges
WHERE table_name= 'proizvod';
```

	grantee name	privilege_type character varying
1	postgres	INSERT
2	postgres	SELECT
3	postgres	UPDATE
4	postgres	DELETE
5	postgres	TRUNCATE
6	postgres	REFERENCES
7	postgres	TRIGGER
8	PUBLIC	INSERT

Slika 23- Prikaz rezultata provere GRANT naredbe

GRANT ALL PRIVILEGES ON narudzbina TO korisnik;

	grantee name	privilege_type character varying
1	postgres	INSERT
2	postgres	SELECT
3	postgres	UPDATE
4	postgres	DELETE
5	postgres	TRUNCATE
6	postgres	REFERENCES
7	postgres	TRIGGER
8	korisnik	INSERT
9	korisnik	SELECT
10	korisnik	UPDATE
11	korisnik	DELETE
12	korisnik	TRUNCATE
13	korisnik	REFERENCES
14	korisnik	TRIGGER

Slika 24 – Prikaz primera GRANT naredbe dodele svih privilegija nad objektom(tabelom) narudžbina ulozi korisnik i rezultat provere izvršenja naredbe

Druga varijanta komande GRANT, dodeljuje članstvo u ulozi jednoj ili većem broju novokreiranih uloga. Ovo je važna opcija kada se govori o sigurnosti baze podataka, jer se privilegije date jednoj ulozi, prenose na sve članove grupe u kojoj se ona nalazi, čime se utiče na kontrolu pristupa podacima. Ako je navedena opcija WITH ADMIN, svaki član grupe uloga koji sadrži ovu opciju, može dodeliti članstvo ostalim ulogama ili ga opozvati. Ukoliko je navedena privilegija CREATEROLE, sve uloge koje je poseduju, mogu odobriti ili opozvati članstvo bilo kojoj ulozi koja nije superuser. Za superusere (kod PostgreSQL baze podrazumevani superuser je postgres uloga) ovo ne važi, odnosno njima se ne može osporiti ni jedna privilegija, kao ni dodeliti od strane drugih korisnika uloga, ali oni mogu izvršiti te operacije nad svim ostalim ulogama. Parametar GRANT BY, označava evidentiranje izdavanja odobrenja članstva novoj ulozi od strane navedene uloge i ova opcija je uglavnom dostupna samo superuser-ima.

```
GRANT ALL PRIVILEGES ON studenti TO profesori;
GRANT profesori TO studenti;
```

Slika 24- prikaz primera druge varijante GRANT naredbe kojom se prenosi članstvo i pristup privilegijama nad objektom sa jedne na drugu ulogu

```
SistemiBP=# \du studenti
List of roles
Role name | Attributes | Member of
-----+-----+-----
studenti | Cannot login | {profesori}
```

Slika 25- Prikaz pripadnosti uloge studenti grupi uloge profesori

Na slici iznad je primetno da uloga studenti kao vrednost polja atribut ima sadržaj „ Cannot login“ i to znači da postavljena uloga nema pravo pristupa terminalu ili drugim sistemskim resursima, već je ograničena samo na PostgreSQL bazu podataka.

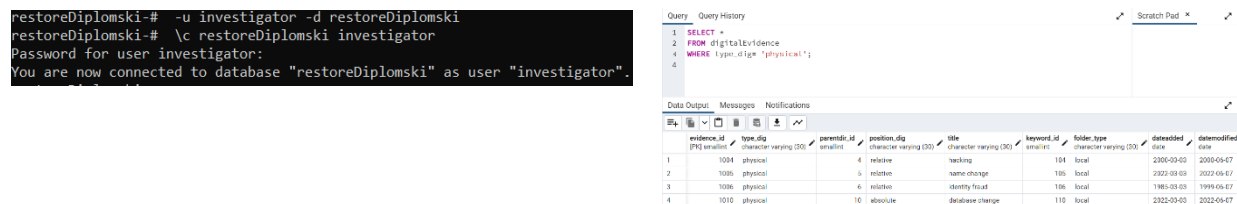
Pored opisanih svojstava, komanda GRANT omogućava proširenje druge varijante dodele privilegija korisnicima, u smislu da umesto da svakom korisniku ponaosob dodeljuje uloge, to može izvršiti grupno. Na primer, kreirana je uloga „investigator“ kojoj je omogućena privilegija SELECT nad tabelom „digitalevidence“ u PostgreSQL bazi podataka. Nakon toga su kreirane dve nove uloge koje nasleđuju sva svojstva prethodno kreirane uloge, ali za razliku od nje nemaju prava izmene podataka, već samo pregleda istih. Ovim primerom obuhvaćena su oba koncepta GRANT komande, ali i princip nasleđivanja, koji podrazumeva da novokreirane uloge naslede i šifru, pa to treba detaljno razmotriti prilikom korišćenja ovih metoda.

```
CREATE ROLE investigator;
GRANT SELECT ON digitalevidence TO investigator;
CREATE ROLE juniorInvestigator1 LOGIN INHERIT;
CREATE ROLE juniorInvestigator2 LOGIN INHERIT;
GRANT investigator TO juniorInvestigator1,juniorInvestigator2;
ALTER USER investigator WITH PASSWORD '1234';
```

Slika 26- Prikaz grupisanja uloga sa privilegijama

Nakon izvršenja ovih naredbi, moguće je logovati se na bazu podataka kao jedan od kreiranih korisnika i izvršiti klasičan SELECT upit nad tabelom. SELECT privilegija se podrazumevano prenosi na sve kolone tabele nad kojom je pozvana, ali to ne mora biti slučaj, tačnije moguće je

ograničiti nove uloge koje će se kreirati na određene kolone u tabeli. Opisani elementi prikazani su na narednim slikama.



Slika 27- Prikaz pristupa bazi podataka u svojstvu jedne od kreiranih uloga i rezultat izvršenja jednostavnog upita od strane ulogovanog korisnika

```
CREATE ROLE intern LOGIN;
GRANT SELECT (evidence_id, title, folder_type) ON digitalevidence TO intern;
CREATE ROLE sena LOGIN INHERIT;
GRANT intern TO sena;
```

Slika 28- Ograničenje SELECT privilegije na određene kolone tabele kojoj pristupa novi korisnik

Pored ALTER i GRANT naredbi, na početku priče o sigurnosti PostgreSQL baze korišćenjem uloga, pomenuta je i naredba REVOKE. Njome se sve prethodno izdate privilegije, povlače. Obično to može učiniti samo vlasnik privilegije nad objektom baze podataka, međutim ukoliko je prethodno iskorišćena naredba WITH GRANT, koja i ostalim primaocima omogućava da podele privilegije (onima koji nisu vlasnici objekta baze podataka), onda se i kod takvih uloga može iskoristiti REVOKE komanda za povlačenje privilegija [11].

```
REVOKE ALL ON proizvod FROM PUBLIC;
```

Slika 29- Prikaz REVOKE komande kojom se povlače sve privilegije prethodno izdate nad tabelom „proizvod“

Sigurnost na nivou redova (eng. Row Level Security) je napredna opcija PostgreSQL baze podataka, kojom se obezbeđuje dodatna bezbednost podataka sačuvanih u bazi. Sigurnost na nivou redova, skraćeno RLS, administratorima PostgreSQL baze podataka omogućava da definišu dodatnu kontrolu pristupa podacima u bazi, određujući način na koji se određeni redovi prikazuju korisnicima i šta oni mogu da urade sa prikazanim podacima. U suštini, RLS, predstavlja dodatni filter koji se može primeniti nad redovima tabele sačuvane u PostgreSQL bazi podataka. Kada ulogovani korisnik pokuša da izvrši neku radnju nad podacima u tabeli, primenjuje se RLS filter i to pre kriterijuma zadatog upita, a podaci se automatski sužavaju u željeni skup ili se odbijaju ukoliko nisu obuhvaćeni bezbedonosnom politikom RLS filtera [12]. Kako bi sigurnost na nivou redova bila korišćena, potrebno je učiniti sledeće dve stvari- omogućiti RLS za tabelu nad čijim će se redovima primenjivati i definisati politiku koja će kontrolisati pristup na nivou reda [8]. Nakon što je dozvoljena sigurnost na nivou redova u tabeli korišćenjem formata naredbe ALTER TABLE ime_tabele ENABLE ROW LEVEL SECURITY, obavezno se mora ispuniti drugi kriterijum, odnosno obavezno se mora kreirati politika bezbednosti redova, jer ona definiše normalan način pristupa tabeli i modifikovanje redova. Ukoliko se ovaj deo izostavi, kod PostgreSQL baze podataka se podrazumevano koristi smernica odbijanja, što znači da ni jedan red nije vidljiv niti su moguće promene nad njim [13]. Važno je napomenuti da operacije poput TRUNCATE i REFERENCE ne podležu sigurnosti na nivou redova, ali da kreirane politike mogu biti specifične za određene komande kao što su SELECT, INSERT, UPDATE i DELETE, jednako kao i za kreirane uloge u bazi podataka.

Da bi se odredilo koji su redovi vidljivi ili modifikovani u skladu sa politikom, potrebno je navesti jedan izraz koji vraća Bulov rezultat. Taj izraz će biti procenjen za svaki red pre bilo kakvih uslova ili funkcija koje se nalaze u korisničkom upitu, pri čemu svi redovi koji ne vraćaju true kao vrednost Bulovog izraza, neće biti obrađeni [13].

Pored svega navedenog, najviše pažnje, ipak treba pokloniti aspektima i parametrima koji utiču na kreiranje RLS politike kontrole pristupa podacima. Neki ključni aspekti su već objašnjeni, ali i pored toga treba znati da su sve smernice koje se koriste za formiranje RLS politika selektivne. To znači da se primenjuju samo nad onim redovima koji odgovaraju unapred definisanom SQL izrazu. Takođe, nazivi smernica se mogu ponavljati u tabelama, odnosno moguće je kreirati RLS politiku sa jednim imenom i ponovo je koristiti za više različitih tabela. Parametri povezani sa politikom, pored njenog imena i imena tabele nad kojom će se primeniti, uključuju i [permissive](#) i [restrictive](#) opcije, komande na koje će politika direktno uticati, imena kreiranih uloga, ali i [using_expression](#) i [check_expression](#) konstrukcije. Permissive parametar je difoltni parametar kod PostgreSQL baze podataka i dozvoljava primenu većeg broja smernica na jedan upit, dok restrictive definiše restriktivnu politiku koja ograničava pristup određenim tabelama.

Using_expression funkcija je zapravo SQL izraz koji vraća logičke vrednosti, pri čemu se svaki red proverava u odnosu na ovaj parametar i od njega zavisi da li korisnik može da vidi neki red ili ne. Check_expression parametar je SQL izraz koji vraća boolean i koristi se kada se nad tabelom vrše INSERT ili UPDATE klauzole. Ukoliko je vrednost ovog izraza true, pomenute operacije se mogu izvršiti nad redovima, a ukoliko je false, podaci se odbacuju i operacije nad njima se onemogućavaju [13].

Prethodno opisani elementi, prikazani su u okviru narednog primera, koji takođe obuhvata i ranije predstavljene koncepte sigurnosti na nivou baze podataka korišćenjem uloga. Za demonstraciju praktičnog dela, kreirana je tabela passwdFile, koja sadrži sva relevantna polja koja bi se našla u realnom Windows password fajlu, kao što su korisničko i puno ime korisnika, lozinka, status koji je ili onemogućen ili zaključan, datum poslednje promene lozinke i istorijat lozinki. Takođe se u okviru ove tabele, nalazi i polje password_reset_url, koje je korisno za Windows, ukoliko treba iskoristiti opciju vraćanja lozinke putem web stranice.

```
CREATE TABLE passwdFile (  
    user_name          text UNIQUE NOT NULL,  
    full_name          text NOT NULL,  
    password           text NOT NULL,  
    disabled           boolean NOT NULL,  
    locked             boolean NOT NULL,  
    password_expired   boolean NOT NULL,  
    password_last_set  timestamp with time zone NOT NULL,  
    password_expiration timestamp with time zone NOT NULL,  
    account_expiration timestamp with time zone NOT NULL,  
    password_history   text[] NOT NULL,  
    password_reset_url text,  
    PRIMARY KEY (user_name)  
);  
  
INSERT INTO passwdFile  
(user_name, full_name, password, disabled, locked, password_expired, password_last_set, password_expiration, account_expiration, password_history,  
password_reset_url)  
VALUES  
(  
    'admin', 'Administrator', md5('sifra'), false, false, false, now(), now() + INTERVAL '90 days', now() + INTERVAL '180 days', ARRAY[md5('sifra')],  
    'https://example.com/reset/admin')  
(  
    'janedoe', 'Jane Doe', md5('1234567'), false, false, false, now(), now() + INTERVAL '90 days', now() + INTERVAL '180 days', ARRAY[md5('1234567')],  
    'https://example.com/reset/janedoe'),  
(  
    'bobsmith', 'Bob Smith', md5('123456'), false, false, false, now(), now() + INTERVAL '90 days', now() + INTERVAL '180 days', ARRAY[md5('123456')],  
    'https://example.com/reset/bobsmith');
```

Slika 30- Prikaz kreiranja i popunjavanja tabele „passwdFile“

Nakon kreiranja tabele, kreirane su i tri uloge-administrator i dva korisnika, kojima se dodeljuju politike prava pristupa nad određenim redovima tabele. Kako bi se video efekat korišćenja kreiranih politika, prethodno je neophodno dozvoliti korišćenje sigurnosti na nivou redova nad tabelom passwdfile.

```
CREATE ROLE admin; ALTER TABLE passwdfile ENABLE ROW LEVEL SECURITY;
CREATE ROLE jane;
CREATE ROLE bob;
```

Slika 31- Kreiranje uloga i omogućavanje sigurnosti na nivou redova nad kreiranom tabelom

```
CREATE POLICY admin_all ON passwdfile TO admin USING (true) WITH CHECK (true);
CREATE POLICY all_view ON passwdfile FOR SELECT USING (true);
CREATE POLICY user_modify ON passwdfile FOR UPDATE
USING (current_user = user_name)
WITH CHECK (current_user = user_name AND password_reset_url IN ('https://example.com/reset/'));
```

Slika 32- Kreiranje politika pristupa redovima od strane uloga korisnika

Posmatranjem slike, jasno se može uočiti da je prvom komadnom, administratoru omogućeno da vidi postojeće i doda nove redove, a da je ostalim korisnicima dozvoljeno da vide sve redove u tabeli (druga komanda), ali mogu da izmene samo svoje podatke, odnosno imaju ograničen pristup podacima i takođe ograničene mogućnosti modifikacije polja koje sadrži reset url password-e. Međutim, da bi efekti ovako kreiranih politika kontrole pristupa, bili vidljivi, neophodno je dodeliti dozvole kreiranim ulogama. To se postiže komandom GRANT, koja je ranije detaljno opisana u radu.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON passwdfile TO admin;
GRANT SELECT (user_name, full_name, disabled, locked, password_last_set) ON passwdfile TO PUBLIC;
GRANT SELECT (full_name, password) ON passwdfile TO PUBLIC;
GRANT UPDATE (full_name, password) ON passwdfile TO PUBLIC;
```

Slika 33- Definisanje dozvola i privilegija ulogama administratora i korisnika

Demonstracija izvršenja opisanih koraka i komadi, data je na narednim slikama.

```
SET ROLE admin; SELECT * FROM passwdfile;
```

user_name [PK] text	full_name text	password text	disabled boolean	locked boolean	password_expired boolean	password_last_set timestamp with time zone	password_expiration timestamp with time zone	account_expiration timestamp with time zone	password_history text[]	password_reset_url text
admin	Administr...	92335228...	false	false	false	2023-05-08 11:08:25.058...	2023-08-06 11:08:25.058...	2023-11-04 11:08:25.058...	{9233522847674...	https://example.co...
janedoe	Jane Doe	508df4cb...	false	false	false	2023-05-08 11:08:25.058...	2023-08-06 11:08:25.058...	2023-11-04 11:08:25.058...	{508df4cb2f4d8f...	https://example.co...
bobsmith	Bob Smith	e10adc39...	false	false	false	2023-05-08 11:08:25.058...	2023-08-06 11:08:25.058...	2023-11-04 11:08:25.058...	{e10adc3949ba5...	https://example.co...

Slika 34- Prikaz sposobnosti administratora da pristupi svim redovima u tabeli

```
SET ROLE jane;
```

Query
Query History

1 SELECT * FROM passwdfile;
2

Data Output
Messages
Notifications

ERROR: permission denied for table passwdfile
SQL state: 42501

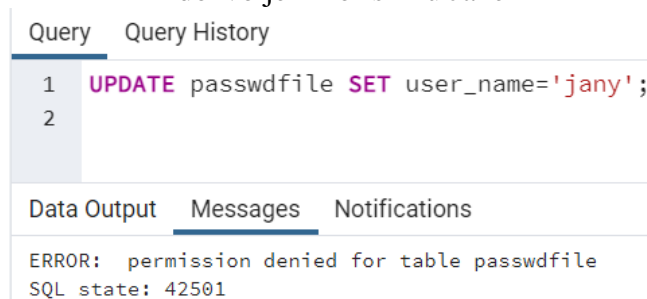
Slika 35- Postavljanje uloge korisnika „Jane“ i rezultat izvršenja upita kojim se zahteva prikaz svih redova u tabeli

Rezultat izvršenja ovog upita od strane korisnika Jane, je negativan iz razloga što je kreiranjem politike na nivou reda i dodelom privilegija orisničkim ulogama, omogućen pristup samo određenim poljima. Iz tog razloga sistem projavljuje grešku i korisniku Jane se ne prikazuju podaci, politika kontrole na nivu redova i data privilegija, sprečavaju uvid u nedozvoljene podatke. Ovo se može prevazići pravilnim navođenjem polja u upitu, za koja se zna da korisnik sa određenom ulogom ima dozvolu pristupa. Još jedna od stvari koja nije moguća korisniku sa ulogom Jane, je i izmena ličnih podataka drugog korisnika, korisnik može da promeni samo svoje podatke i to one redove koji su obuhvaćeni dozvolom politike. RLS u tom slučaju, izvrši promenu polja odgovarajućeg reda i u „tišini“ spreči promenu drugih redova i polja.

SELECT user_name, full_name, disabled, **locked**, password_last_set **FROM** passwdfile;

user_name [PK] text	full_name text	disabled boolean	locked boolean	password_last_set timestamp with time zone
admin	Administrator	false	false	2023-05-08 11:08:25.058762+02
janedoe	Jane Doe	false	false	2023-05-08 11:08:25.058762+02
bobsmith	Bob Smith	false	false	2023-05-08 11:08:25.058762+02

Slika 36- Prikaz pravilnog navođenja elemenata u SQL upitu koji su RLS politikom i ulogama dozvoljeni korisniku Jane



Slika 37- Prikaz neadekvatne upotrebe UPDATE naredbe od strane korisnika Jane koji pokušava da promeni user_name drugom korisniku

UPDATE passwdfile **SET** full_name= 'Jane';
UPDATE 3

Slika 38- Prikaz adekvatne upotrebe UPDATE naredbe od strane korisnika Jane koji menja svoje puno ime

Razlog zabrane izvršenja UPDATE naredbe, predstavljene na gornjoj slici, ne nalazi se samo u činjenici da korisnik pokušava da promeni tuđe ime, već i u tome da korisnicima uopšte nije omogućena UPDATE opcija nad poljem user_name (RLS politikom je stavljeno ograničenje). Svi prikazani primeri do sada, odnosili su se na permissive politiku, što znači da je korišćen podrazumevani PostgreSQL mehanizam korišćenja „ILI“ operatora prilikom prolaska kroz zadate smernice. Restrictive opcijom, koristi se logički „AND“ operator, koji zahteva ispunjenje svih smernica istovremeno da bi se realizovala RLS politika. Primer koji demonstrira ovu opciju, dat je na narednoj slici, koja prikazuje restriktivnu politiku kojom se zahteva povezivanje administratora na Unix socket kako bi pristupio zapisima u tabeli.

CREATE POLICY admin_local **ON** passwdfile **AS** RESTRICTIVE **TO** admin **USING** (pg_catalog.inet_client_addr() **IS** NULL);

Slika 39 – Prikaz primene Restrictive mere nad ulogom administratora

U današnje vreme, krajnjim korisnicima se sve češće daje više slobode u pristupu bazi podataka, u svrhu upravljanja nalogom, dobijanja korisničke podrške ili obavljanja nekih funkcija poput online kupovine i praćenja narudžbine. Iz ugla korisnika, sve ove opcije su izuzetno povoljne i korisne, ali iz ugla sigurnosti baze podataka, svaka funkcija koja korisnicima omogućava pretragu ili izmenu sadržaja u bazi podataka, predstavlja rizik. Vrlo često, napadači koriste ranjivost sistema upravo kroz zloupotrebu tih funkcija i taj metod ugrožavanja sigurnosti baze podataka, poznat je kao SQL injekcija [14]. SQL injekcije predstavljaju zlonamerne napade u kojima se u SQL upit „ubrizgavaju“ određene destruktivne fraze. Napadači vrlo često koriste ovu tehniku kako bi pribavili poverljive podatke i lične informacije, ili kako bi izvršili modifikaciju ili uklanjanje podataka iz baze [14]. Destruktivne fraze kojima se ugrožava sigurnost podataka, odnose se na DROP TABLE, DELETE FROM, INSERT i UPDATE klauzole, jednako kao i na ubacivanje - i ; u upit. PostgreSQL ima nekoliko načina kojima održava sigurnost svog jezgra i štiti se od ove vrste napada, a to su: validacija unosa, proaktivni monitoring, prilagođenje podrazumevanih podešavanja i auditing.

Auditing (revizija baze podataka) je sistematsko praćenje sveobuhvatnih promena i aktivnosti, ali i organizacione strukture i bezbedonosnih politika i definicija korisničkih naloga, kako bi se održala sigurnost sistema. PostgreSQL baza podataka podržava auditing sa opcijama[15] :

- **log_statement** = all- omogućava detaljno evidentiranje svih aktivnosti u PostgreSQL bazi podataka;
- **pgaudit**- pruža mogućnost filtriranja, ali i kovertovanja evidencije aktivnosti u format koji odgovara revizionim tehnikama;
- **custom triggers**- kreiranje sopstvenih revizionih tokova.

Log_statement parametar u konfiguraciji PostgreSQL baze podataka određuje koji se izrazi evidentiraju, pa utiče ne obezbeđenje sigurnosti i prevenciju SQL injekcija, na taj način što sadrži sve informacije o kreiranim upitima, njihovom sadržaju, podacima na koje utiču. Karakteristično za ovaj parametar je da samo superuseri mogu da ga menjaju. Od opcija podržava sledeće:

- none- nema evidentiranja. Ovo je kod PostgreSQL podrazumevana opcija;
- ddl- evidentiranje svih operacija koje podrazumevaju CREATE, DROP, i ALTER naredbe;
- mod- radi isto što i ddl, ali za naredbe INSERT, DELETE i UPDATE;
- all- evidentira apsolutno sve SQL upite i uglavnom se radi postizanja veće sigurnosti, preporučuje korišćenje ove opcije.

Pgaudit je parametar koji pruža dublji nivo detaljnog evidentiranja sesije i promena nad objektima baze podataka, u odnosu na standardno beleženje aktivnosti kod PostgreSQL baze podataka. Korišćenjem ovog parametra, mogu se ustanoviti neobični ili nepravilni upiti, ali i prikupiti informacije o tome koje tabele, pogledi, procedure i funkcije više nisu u upotrebi. Uklanjanje zastarelih objekata ili sumnjivih upita, smanjuje mogućnost injektiranja nepravilnosti u sistem baze podataka [14].

Primer primene auditinga i njegovog parametra pgAudit za obezbeđenje sigurnosti baze podataka dat je u nastavku.

```
SELECT * FROM proizvod WHERE proizvod_id='10 UNION SELECT ime, prezime, email FROM korisnik --';
```

Slika 40- Prikaz upita sa SQL injekcijom

Na slici je dat jednostavan upit kojim se zahteva prikaz proizvoda čiji je id pod rednim brojem 10, ali je u sam upit integrisan nevalidni SQL kod za spajanje dve SQL naredbe putem naredbe

UNION, sa dodatkom -- na kraju upita, kako bise izbegla sintaksna greška. Druga naredba, zahteva lične informacije o korisniku proizvoda, pa će rezultat na izgled bezazlenog upita biti i podaci o korisniku, što može biti prilično opasno. Međutim korišćenjem pgAudita, SQL injekcioni napad se može sprečiti na sledeći način. Prvo je neophodno omogućiti pgAudit ekstenziju, korišćenjem naredbe:

```
CREATE EXTENSION pgaudit;
```

Slika 41- Omogućavanje ekstenzije parametra pgAudit

Nakon instaliranja ekstenzije, neophodno je podesiti odgovarajuće opcije u postgresql.conf fajlu. Neophodne opcije se dodaju na kraju fajla, ili ručno preko tekst editora ili korišćenjem ALTER SYSTEM SET naredbe.

```
ALTER SYSTEM SET pgaudit.log_catalog = on;  
ALTER SYSTEM SET pgaudit.log_level = log;  
ALTER SYSTEM SET pgaudit.log_parameter_values = on;  
ALTER SYSTEM SET pgaudit.log_statement_once = on;  
ALTER SYSTEM SET pgaudit.log = 'ddl, role, read, write';
```

Slika 42 – Podešavanje parametara pgAudit konfiguracije u postgresql.conf fajlu

Sa slike se može uočiti da su omogućene određene komande kroz ddl opciju, kao i kreiranje uloga, ali i čitanje i upis. Međutim da bi sigurnost bila podržana, treba se izvršiti specifikacija tabela koje će biti auditovane i to se postiže obezbeđenjem RLS politike, o kojoj je ranije u radu bilo reči. Na kraju je važno podesiti pgAudit da beleži izvršenje svih SQL komandi, kako bi bilo moguće zabeležiti sve komande koje će se izvršiti nad podacima tabele „proizvod“ i omogućiti njihov pregled u logovima. Ukoliko se i nakon toga pokuša sa prikazanim SQL inketivnim upadom, taj napad će biti evidentiran i obrađen u cilju sprečavanja urušavanja sigurnosti PostgreSQL baze podataka.

```
ALTER TABLE proizvod ENABLE ROW LEVEL SECURITY;  
SELECT set_config('pgaudit.log','all', true);
```

Slika 43- Podešavanje tabele za auditovanje i konfigurisanje pgAudit-a za beleženje izvršenih SQL komandi

Ukoliko se izvrši poređenje revizionih parametara, iz ugla sigurnosti PostgreSQL baze podataka, dolazi se do zaključka da je u većini slučajeva bolje iskoristiti pgAudit parametar u odnosu na log_statement i to iz više razloga. Prvi od njih je mogućnost filtriranja određenih operacija i generisanje evidencije u odgovarajućem revizionom formatu. Zatim je, korišćenjem pgAudit-a dozvoljeno specificirati evidenciju određenih objekata u bazi i reviziju izvršiti detaljno korišćenjem struktuiranog izlaza, pogodnog i za naredne procese auditovanja.

```
-- SQL statement  
DO $$  
BEGIN  
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';  
END $$;  
-- log_statementall generates this log info:  
2020-12-20 23:48:11 UTC:157.230.232.139(53064):sppostgres@test:[9091]: LOG: statement:  
DO $$  
    BEGIN  
        EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';  
    END $$;  
-- pgAudit generates this expanded log info:  
2020-12-20 23:48:11 UTC:157.230.232.139(53064):sppostgres@test:[9091]: LOG: AUDIT:  
SESSION,4,1,FUNCTION,DO,,,DO $$  
    BEGIN  
        EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';  
    END $$;--not logged  
2020-12-20 23:48:11 UTC:157.230.232.139(53064):sppostgres@test:[9091]: LOG: AUDIT:  
SESSION,4,2,DOL,CREATE TABLE, TABLE,public,important_table,CREATE TABLE  
important_table (id INT),not logged
```

Slika 44- Poređenje pgAudit-a i log_statement-a (izvor slike: PostgreSecurity)

Pogledi , koji su izuzetno korisni za enkapsulaciju najčešće korišćenih upita, pokazali su se kao efikasan alat za obezbeđenje sigurnosti na nivou baze podataka. Oni sprečavaju neovlašćeni pristup podacima, na taj način što osiguravaju da uloge koje su kreirane nemaju mogućnost pristupa podacima iz osnovnih tabela, već iz kreiranog view-a. PostgreSQL koristi `pg_catalog.pg_audith`, koji sadrži red za svaku kreiranu ulogu u bazi podataka, uključujući i kolone koje čuvaju heširane lozinke za postavljenu ulogu. Korišćenjem pogleda, a s obzirom da se heš vrednosti smatraju osetljivim informacijama, tabele nemaju SELECT privilegije ni za jednu ulogu, osim za superuser-e, koji su inicijalno prisutni u bazi podataka. Ukoliko se koriste pogledi koji se mogu ažurirati u nekom trenutku, opcija provere podataka je dostupna samo kada se definiše prikaz tj. pogled. Kada se izostavi, pogled omogućava korisniku da ubaci ili ažurira zapise, ali tako da oni onda nisu vidljivi kroz pogled, a da bi umetanje ili ažuriranje bilo moguće, red mora biti vidljiv korisniku. Ukoliko se radi o lokalnim proverama, vidljivost reda se proverava samo u odnosu na uslove pogleda koji se direktno koristi [6].

3.4 Enkripcija

PostgreSQL nudi enkripciju na nekoliko nivoa, u svrhu postizanja sigurnosti baze podataka, ali i radi pružanja fleksibilnosti prilikom zaštite podataka usled različitih napada na server baze podataka, nesigurnosti mreže i sličnih nedozvoljenih aktivnosti. Takođe, enkripcija može biti neophodna kako bi se zaštitili osetljivi podaci korisnika, poput medicinskih kartona ili finansijskih transakcija [16].

Razlog podele enkripcije u više nivoa kod PostgreSQL baze podataka, nalazi se u fizičkoj strukturi storage sistema, pa se samim tim enkripcija odvija na sledećim nivoima, prateći fizičku strukturu PostgreSQL sistema [17]:

- Enkripcija na nivou servera- podrazumeva da celokupan proces enkripcije obavlja server, uključujući i izbor tipa enkripcije, kao i segmenata nad kojima će se izvršiti ovaj proces. Cena korišćenja enkripcije na nivou servera je daleko niža u odnosu na enkriptovanje na nivou klijenta, jer podrazumeva samo prilagođenje baze podataka bez modifikacije aplikativne logike. Međutim, upravo zbog toga performanse su slabije, a nivo sigurnosti nešto niži u odnosu na sigurnost ostalih enkripcionih nivoa;
- Enkripcija na nivou klastera- enkriptuje se ceo klaster baze podataka, pri čemu se odluka o klasterizaciji donosi tokom procesa inicijalizacije sistema. Arhitektura i cena krošćenja su niske, svi objekti unutar klastera su šifrirani, ali to utiče na degradaciju performansi;
- Enkripcija na klijentskom nivou- podrazumeva generisanje ključa za šifriranje segmenta podataka od strane klijenta. To znači da je granularnost šifriranja mala i da se količina šifrovanih podataka može kontrolisati. PostgreSQL vrlo često enkripciju na ovom nivou, obavlja korišćenjem pgcrypto ekstenzije, o kojoj su dati detalji u nastavku seminarskog rada;
- Enkripcija na nivou baze podataka- koristi biblioteke za enkripciju kojima se vrši šifriranje podataka. Arhitektura je jednostavna i cena korišćenja i održavanja efikasnosti je niska. Problem sa ovom vrstom enkripcije je nedostatak garancije oporavka rezervnih kopija baze, odnosno proces oporavka je prilično složen;
- Enkripcija na nivou tabele- podrazumeva primenu enkripcije nad celom tabelom postavljanjem određenog atributa šifriranja. Korišćenjem enkripcije na ovom nivou, postiže se bolja kontrola nad šifriranim podacima. Enkripcija na nivou tabele se dalje može podeliti na transparentno enkriptovanje podataka, enkriptovanje pojedinačnih kolona i aplikativno enkriptovanje.

Pgcrypto je standardna ekstenzija PostgreSQL baze podataka i svrha ove ekstenzije je da omogući korišćenje SQL funkcija za enkripciju i heširanje podataka, kako bi se povećala sigurnost same baze podataka.

Kada se paket instalira na serveru, neophodno je označiti ga kao superuser-a u željenoj bazi podataka, kao što je prikazano na slici.

```
CREATE EXTENSION pgcrypto;
```

Slika 45- Kreiranje pgcrypto ekstenzije

Nakon uključivanja ekstenzije u željenu PostgreSQL bazu podataka, moguće je koristiti funkcije za heširanje podataka ili za formiranje enkripcije na nivou tabela baze podataka (misli se na klasične enkripcione funkcije simetričnog i asimetričnog postupka).

Najčešće korišćene kriptografske funkcije pgcrypto modula su `crypt()` i `gen_salt()` funkcija, koje su specijalno dizajnirane za heširanje šifri.

Funkcije za heširanje, `crypt()` i `gen_salt()`, predstavljaju deo enkripcionog mehanizma i to tako da prva među njima, `crypt()` vrši heširanje jednim od algoritama koji se koriste u PostgreSQL bazi u ove svrhe (prikazani su na slici), a `gen_salt()` priprema parametre za algoritam po kome će se izvršiti heširanje. Ranije u seminarskom radu, opisani su md5 i scram-sha-265 algoritam, u okviru poglavlja posvećenom autentifikaciji, ali se ovi algoritmi efikasno primenjuju i u procesu enkripcije. Pored njih, koriste se i drugi algoritmi, a svi zajedno prate sledeće principe:brzinu, korišćenje nasumičnih vrednosti, uključivanje tipa algoritma u rezultat i mogućnost prilagođenja performansama računara na kome se nalazi baza u kojoj se primenjuju [16].

Algorithm	Max password length	Adaptive?	Salt bits	Description
bf	72	yes	128	Blowfish-based, variant 2a
md5	unlimited	no	48	MD5-based crypt
xdes	8	yes	24	Extended DES
des	8	no	12	Original UNIX crypt

Slika 46- Prikaz algoritama koje koristi `crypt()` funkcija²

U nastavku je dat primer PostgreSQL enkripcije korišćenjem pgcrypto funkcija za heširanje. Ekstenzija pgcrypto je prethodno omogućena (slika 45) i ona se koristi kako bi se omogućio enkripcijski metod heširanja korišćenjem funkcija `crypt()` i `gen_salt()`. Enkripcija je izvršena nad poljem „email“, tabele „korisnik“ prilikom unošenja podataka u bazu , što znači da unet email, nakon heširanja neće biti prikazan u svom izvornom obliku, već u vidu heš vrednosti, pa je samim tim proces ireverzibilan. Kao algoritam za enkriptovanje, iskorišćen je md5, ali je dat i prikaz korišćenja bf algoritma, koji prolazi kroz 8 rundi enkriptovanja email polja.

```
INSERT INTO korisnik (ime, prezime, email)
VALUES
('student','student',crypt('st.@elfak.rs',gen_salt('md5')));
```

Slika 47- Primer enkripcije funkcijama heširanja i md5 algoritma

² Izvor slike 46- <https://www.postgresql.org/docs>

```
INSERT INTO korisnik (ime, prezime, email)
VALUES
('studentNovi','studentNovi',crypt('student.@elfak.rs',gen_salt('bf',8)));
```

Slika 48- Primer enkripcije funkcijama heširanja i bf algoritma

```
SELECT * FROM korisnik
WHERE ime = 'student' AND email=crypt('st.@elfak.rs',email);
```

korisnik_id [PK] integer	ime character varying (50)	prezime character varying (50)	email character varying (50)
10004	student	student	\$1\$DjlzrcVa\$II4v9N9vPcJf7k4AiF9y30

Slika 49- Prikaz provere regularnosti izvršene enkripcije nad poljem email

Klauzolom SELECT izvršena je provera ispravnosti enkriptovanog email-a, koji je dodeljen korisniku sa imenom „student“. Jednostavnim upitom zahteva se prikaz korisnika, pri čemu se u upitu ponovo koristi crypt funkcija kako bi se izvršilo ponovno prevođenje novozadate vrednosti email-a, a kao drugi parametar navodi se polje email po kome će se poređenje enkriptovanih vrednosti izvršiti. Sa slike se može uočiti da email naveden u SELECT delu upita, odnosno njegova hash vrednost, odgovara vrednosti koja se nalazi u bazi podataka. U suprotnom bi rezultat upita bio prikaz praznog reda tabele, tačnije ako nema podudaranja hash vrednosti u polju tabele i vrednosti zadate upitom, ne prikazuje se ništa.

Enkripcija na nivou tabele je jedna od mogućih opcija kojom se postiže sigurnost podataka u PostgreSQL bazi podataka i kako je navedeno u početnom delu poglavlja seminarskog rada posvećenom enkripciji, može biti u vidu transparentnog enkriptovanja podataka, enkriptovanja pojedinačne kolone ili na nivou aplikacije. U nastavku će na konkretnim primerima biti objašnjeni transparentno enkriptovanje podataka i enkriptovanje vrednosti pojedinačne kolone.

Transparentno enkriptovanje podataka, kao deo enkripcije tabele baze podataka, podrazumeva da se enkripcija odvija na nivou diska, što znači da je konkretna tabela enkriptovana na fizičkom nivou. Kod PostgreSQL baze podataka, ovo je moguće postići kreiranjem enkriptovanog tablespace prostora, u kome će se fizički smestiti tabela iz baze podataka. Enkripcijom tablespace-a (dostupan za sve verzije PostgreSQL baze nakon verzije 8.0), svi objekti koji se kasnije budu skladištili u okviru njega uključujući tabele, indeksne strukture, backup-ove, biće enkriptovani na isti način kao i tablespace. Proces transparentnog enkriptovanja podataka, podrazumeva da je prvo neophodno konfigurisati postgresql.conf fajl, gde se navode lokacija na kojoj će se čuvati enkripcioni ključ i algoritam koji će se koristiti za enkripciju (none kao podrazumevani, AES128 i AES256 prema izboru)[17]. Podešavanje u okviru konfiguracionog fajla učinjeno je narednim komandama:

```
keystore_location = ' C:\Program Files\PostgreSQL\15\data\base\keyLocation';
tablespace_encryption_algorithm= 'AES256';
```

Dalje se izvršava SQL funkcija za kreiranje glavnog ključa kojim se vrši enkriptovanje i navodi se simboličko ime koje se koristi prilikom restartovanja servera kako bi se PostgreSQL baza povezala i uspela da otvori ključ.

```
SELECT pgx_set_master_key('podaci');
```

Slika 50- Kreiranje master ključa

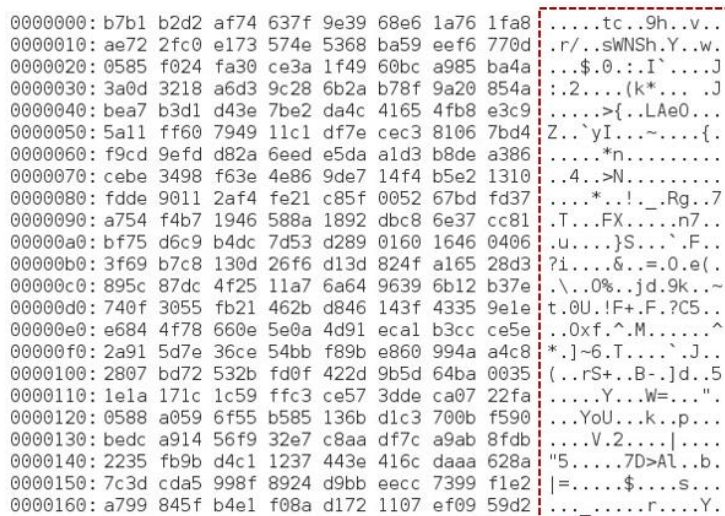
Ovom komandom se kreira datoteka keystore.ks na lokaciji navedenoj u postgresql.conf fajlu. Restartovanje servera vrši se komandom:

```
pg_ctl -keystore-passphrase restart -D \Program Files\PostgreSQL\15\data\base\keyLocation';
```

Nakon restartovanja servera, potrebno je još samo kreirati tablespace na lokaciji na kojoj se nalazi ključ, čime je proces transparentne enkripcije podataka završen:

```
CREATE TABLESPACE primerEncrypt LOCATION 'C:\Program Files\PostgreSQL\15\data\base\tbspodaci';
```

Slika 51- Kreiranje tablespace enkriptovanog prostora



```
00000000: b7b1 b2d2 af74 637f 9e39 68e6 1a76 1fa8 .....tc..9h..v..
00000010: ae72 2fc0 e173 574e 5368 ba59 eef6 770d .r/..sWNSH.Y..w.
00000020: 0585 f024 fa30 ce3a 1f49 60bc a985 ba4a ...$.0...I`....J
00000030: 3a0d 3218 a6d3 9c28 6b2a b78f 9a20 854a :.2....(k*...J
00000040: bea7 b3d1 d43e 7be2 da4c 4165 4fb8 e3c9 .....>{...LAe0...
00000050: 5a11 ff60 7949 11c1 df7e cec3 8106 7bd4 Z..`yI...~....{.
00000060: f9cd 9efd d82a 6eed e5da ald3 b8de a386 .....*n.....
00000070: cebe 3498 f63e 4e86 9de7 14f4 b5e2 1310 ..4..>N.....
00000080: fdde 9011 2af4 fe21 c85f 0052 67bd fd37 ....*...!..Rg..7
00000090: a754 f4b7 1946 588a 1892 dbc8 6e37 cc81 .T...FX.....n7..
000000a0: bf75 d6c9 b4dc 7d53 d289 0160 1646 0406 .u....}S...`F.
000000b0: 3f69 b7c8 130d 26f6 d13d 824f a165 28d3 ?i....&...=.0.e(.
000000c0: 895c 87dc 4f25 11a7 6a64 9639 6b12 b37e .\..0%...jd.9K..~
000000d0: 740f 3055 fb21 462b d846 143f 4335 9e1e t.0U.!F+.F.?C5..
000000e0: e684 4f78 660e 5e0a 4d91 ecal b3cc ce5e ..0xf.^..M.....^
000000f0: 2a91 5d7e 36ce 54bb f89b e860 994a a4c8 *.]~6.T.....J..
00000100: 2807 bd72 532b fd0f 422d 9b5d 64ba 0035 (...rS+...B-..]d..5
00000110: 1e1a 171c 1c59 ffc3 ce57 3dde ca07 22fa .....Y...W=...".
00000120: 0588 a059 6f55 b585 136b d1c3 700b f590 ...YoU...k..p..
00000130: bedc a914 56f9 32e7 c8aa df7c a9ab 8fdb ....V.2....|....
00000140: 2235 fb9b d4c1 1237 443e 416c daaa 628a "5.....7D>A1..b.
00000150: 7c3d cda5 998f 8924 d9bb eecc 7399 f1e2 |=.....$.s...s...
00000160: a799 845f b4e1 f08a d172 1107 ef09 59d2 ..._.....r....Y.
```

Slika 52- Prikaz primera kako enkriptovani podaci izgledaju napadaču³

Enkriptovanje pojedinačne kolone, kao deo enkripcije tabele baze podataka, može se izvršiti simetričnom enkripcijom ili korišćenjem pgcrypto funkcionalnosti (pgcrypto se može iskoristiti i za ovakvu vrstu enkripcije) poznatijih kao PGP funkcije. PGP funkcije se favorizuju u odnosu na klasično simetrično enkriptovanje(koje je takođe deo PGP-a), koje omogućava korisniku da sam obezbedi ključ kao cipher ključ, bez direktne provera integriteta podataka [6].

Simetrično enkriptovanje koristi simetrični ključ, koji služi i za enkripciju i za dekripciju podataka. Tekst se enkriptuje u bajt formatu, a ne u obliku teksta, ali se nakon procesa dekripcije, dobija željeni format. Simetrično enkriptovanje je daleko lakše za upotrebu, jer ne zahteva generisanje random PGP ključa, ali je nivo sigurnosti niži. Ova vrsta enkripcije, može se primeniti i nad drugim objektima baze podataka.

```
SELECT pgp_sym_decrypt(pgp_sym_encrypt('student', 'password'), 'password');
```

pgp_sym_encrypt	pgp_sym_decrypt
bytea	text
[binary data]	student

Slika 53- Proces simetrične enkripcije/dekripcije i prikaz rezultata izvršenja

Iz ugla sigurnosti, PGP funkcije predstavljaju bolji izbor kada se govori o izvršenju enkripcije. PGP funkcije, kod PostgreSQL baze podataka, koriste se kao deo OpenPGP standarda i podržavaju i prethodno opisano simetrično enkriptovanje, ali i public-key enkriptovanje. PGP enkriptovani sadržaj, sastoji se iz dva dela, odnosno 2 paketa: paketa koji sadrži ključ i paketa u kome se nalaze podaci enkriptovani tim ključem. Ranije je opisano simetrično enkriptovanje, koje koristi PGP funkcije za generisanje simetričnog ključa, ali ukoliko se koristi enkripcija sa public ključem, PGP funkcije generišu random vrednost ključa koji se koristi u procesu

³ Izvor slike 52- <https://arctype.com/blog/transparent-data-encryption/>

enkripcije podataka unutar kolone. Generisanje ključa može se izvršiti korišćenjem GnuPG, pri čemu PostgreSQL preferira tipova ključa kao što su „DSA ili Elgama“. Kada se ključ generiše, potrebno ga je eksportovati i tek tada može da se iskoristi za enkripciju ili dekripciju podataka [6].

```
PS C:\Users\korisnik> gpg -- gen key
```

```
PS C:\Users\korisnik> gpg -a --export KEYID > public.key
```

Slika 54 – Generisanje i eksportovanje public ključa

Na slici je dat prikaz generisanja i eksportovanja public ključa korišćenjem gpg komande. Prilikom generisanja ključa, gpg može zahtevati dodatne informacije, kao što su ime i email adresa, nakon čega je potrebno kreirati lozinku za zaštitu ključa. Kada je ključ kreiran, može se eksportovati u ASCII format, ili se taj deo može izostaviti, jednostavnim uklanjanjem `-a` iz komande eksportovanja. Kreirani ključ, može se koristiti u procesu enkripcije/dekripcije u sklopu `pgp_public_encrypt` funkcije, pri čemu se zbog specifičnog formata ključa, koristi `dearmor()` funkcija, kako bi izdvojila vrednost ključa i predala ga enkripcionoj funkciji [16].

```
SELECT pgp_pub_encrypt('student@elfak.rs',dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
mQENBF53PQoBCAC30B/Hg9K0f+iGwzF+R04eKZpL0h8Twvl01q9Esxt15jvH2cWn
vyBqt2ez/DnLhHiZiCjZLLn5y0d9f1Yi5DhW5r80/JIQmwrp1j67GmKM4x4impID
QjXZZ/Mm0dTVS8eS3jg95lN6AIHvPQxUMV0G/6KhtXV7wpU6HsU6BqKw6pyti7Ff
98Z52+iaVU9KjXG7VUkE6U/pCCVTR62ecw47zTYQQSWIay+Gr4aIdf2QJj7l+Tx
dFKmySjbIDoQ05A0m0Zz1hE8zyfPFzh/tB9Y1vdt5m5W5hDkrwWhrh68RzvFjKxW
om0qcyLgSTTWEEJhQLEkN0ilw2IDBXzNTtF1ABEBAAG0IUxhcnJpbmEgUm9ja3Mg
PGxhcnJpbmFyb2Nrc0BlcGFtcGxlLnVvbT6JAVQEEwEIAQ4WIQT/3Ywuc0yycIhP
fuN6+/0Q20gUCXnc9CgIJCgIbAwYLCQgHAWIGFQgCCQoLBBYCAwECHgECF4AACgkQ
N6+/0Q20gVH03Af/SxxuM7Vzg6+IMn6XZU6C4L4AYp4Yq+18tbjyJJe6QcPNhSLL
3qAzN9Z0hugajQx+zIN0Cc/79ahTJyRf+Gc8rUh/bhzlMuwhhZmLHMYpgOC8C+oO
QJwpm/o51ChWz8Fvrt+01xg8fDnBzPjTh9GtZkwW8eojk1lzMc+kzYQJzh2xBbwo
rqvQzwJtbCGhX9ZqTAslLuzUedxIUPRfTVTnTz4hRp4e6fc+/R6UuXcl/9biJhSj
iQzYwYagN3q5j50BL7j/Nkn0sCfiZ9PQ/eWmKv7UASdHw+Vz5Lx5x5gJn5/h+uZ9
DZLaB/s8fVJgkvY+10mVz5AFp4JFhntUcAhnU9
-----END PGP PUBLIC KEY BLOCK-----')) AS email
FROM korisnik
WHERE korisnik_id = 10005;
```

Slika 55- Primer korišćenja javnog enkripcionog ključa kreiranog gpg komandama. Upit sa slike predstavlja jednostavan zahtev za izdvajanjem korisnika sa zadatim id-em. Međutim, dodatno je kolona email, odnosno polje koje se odnosi na tog korisnika, enkriptovana korišćenjem `pgp_pub_encrypt` funkcije, kojoj se kao prvi parametar prosleđuje vrednost polja kolone email, a kao drugi javni ključ. Funkcija `dearmor` se koristi da iz formata javnog ključa ukloni delove koji nisu sadržaj ključa, kao što su `BEGIN` i `END`. Nakon izvršenja upita, polje kolone email je enkriptovano i predstavljeno u ASCII formatu. Važno je naglasiti da s obzirom da je ključ javni, svako ponavljanje izvršenja ovog upita nad istim poljem kolone, daće različitu enkripcionu vrednost.

Enkripcija je izuzetno korisna, pogotovu ako se skladište osetljivi podaci kojimase vrlo često pristupa. Enkripcija korišćenjem javnog ključa ima mnogo više efekta i smisla, kada se podaci razmenjuju van sistema, dok simetrična enkripcija predstavlja bolji izbor ukoliko se baza koristi za jednu, samostalnu aplikaciju [6].

4. PRAKTIČNI PREGLED KORIŠĆENIH TEHNIKA ZA POSTIZANJE SIGURNOSTI POSTGRESQL BAZE PODATAKA

U prethodnom poglavlju seminarskog rada, detaljno su opisane neke od tehnika kojima PostgreSQL baza podataka postiže i održava sigurnost. Ovo poglavlje rada posvećeno je načinima koji se najčešće koriste u praksi za obezbeđivanje sigurnosti PostgreSQL baze podataka.

Jedna od dobrih praksi koja se vrlo često primenjuje, kako bi se održala sigurnost PostgreSQL baze podataka, jeste odluka o smeru enkriptovanja podataka. U praksi se velika pažnja poklanja tome da li podaci treba da budu jednosmerno enkriptovani (ne mogu se dekriptovati) ili dvosmerno enkriptovani (mogu se vratiti u izvorni oblik). Dvosmerno se enkriptuju podaci koji služe za analizu i najčešće se koristi AES enkripciona tehnika, dok se jednosmerno enkriptuju podaci koji su osetljivi, kao što su korisničke lozinke [18]. Pored toga, tehnike šifriranja zasnovane na heš-u, kao jednosmerno enkriptovanje, sve ređe koriste md5 kriptografski algoritam, a okreću se sigurnijem i efikasnijem scram-sha-256 algoritmu [19].

Sledeća stvar koju administratori PostgreSQL baze podataka primenjuju u realnosti, jeste fizičko izolovanje osetljivih skupova podataka, koje se postiže korišćenjem pg_hba, koji je ranije opisan u radu u delu poglavlja namenjenog autentifikaciji. Ovim se osigurava da se podacima u dve različite tabele ne može pristupiti istovremeno u toku jedne sesije projavljivanja od strane jednog korisnika. Takođe, ovaj proces utiče na prekid SQL join-a, pa ga treba koristiti samo u odgovarajućim situacijama.

Sprečavanje eksternih veza sa bazom podataka važan je princip sigurnosti i odnosi se na konfiguraciju listen_addresses parametra na adresu localhost-a ili hosta mašine koja pokreće aplikaciju koja koristi podatke iz PostgreSQL baze podataka. Ovakva konfiguracija, primorava operativni sistem da odbije sve pokušaje konektovanja sa bilo koje mašine, osim one koja je navedena u konfiguraciji i ovim utiče na eliminisanje nedozvoljenog pristupa podacima [18].

`Set listen_addresses= 'localhost'`

`Set listen_addresses = 'localhost, <IP of network device>'`

Drugi način sprečavanja eksternih veza je postići onemogućen udaljeni pristup konfiguracijom pg_hba.conf fajla ili formiranjem SSH tunelovanja kod PostgreSQL baze podataka, kao što je opisano ranije u seminarskom radu.

U praksi se veliki napor ulaže u to kako da evidencija baze podataka zadrži integritet, odnosno kako da baza ne otkrije više informacija od onoga što je predviđeno. Jedan od načina kako se u realnosti ovaj problem prevazilazi je izbegavanje korišćenja klasične SQL create user naredbe za kreiranje korisnika, jer se time korisničke šifre u evidencionim podacima prikazuju u vidu običnog teksta. Umesto toga, poželjnije je korišćenje crateuser komande operativnog sistema, koja lozinke odmah prikazuje u šifrovanom formatu [18].

U PostgreSQL bazi podataka postoji PUBLIC notacija, koja vrlo često izaziva ozbiljne probleme, jer podrazumevano, svaki kreirani objekat ima pristup svim drugim objektima koji su takođe specifikirani kao PUBLIC. Na primer, kreiranjem jednog korisnika u jednog bazi podataka, korisniku neće biti dozvoljeno da sam kreira novu bazu u okviru šeme u kojoj se nalazi, ali će imati mogućnost da pristupi nekoj drugoj bazi podataka, ali će po podrazumevanim PostgreSQL podešavanjima javne šeme, moći da kreira tabelu. U ovakvim situacijama, generalna preporuka je ukloniti PUBLIC iz privilegija baze podataka i javne šeme, kao i izvršiti restrikciju privilegija datih korisnicima [19].

Ranije je u radu opisana auditing tehnika kao jedan od načina uklanjanja SQL injekcija u PostgreSQL bazi. Pored ove tehnike, generalne preporuke stručnjaka su korišćenje format()

funkcije za konstrukciju SQL query stringa, korišćenje string tipa podataka kao parametra samo ukoliko je to neophodno, kao i upotreba dinamičkog SQL-a jedino u izuzetnim i neizbežnim situacijama.

Pored ovde prikazanih tehnika u praksi postoji još dosta drugih načina kojima se održava sigurnost podataka, u toku rada su takođe prikazani važni elementi, ali ono što je izuzetno važno je da stalno ostanete u toku sa napretkom i razvojem PostgreSQL sistema i sigurnosnih podešavanja, kako biste na pravi način iskoristili puni potencijal baze i garantovali integritet podacima koji se nalaze unutar nje.

```
REVOKE ALL ON DATABASE SistemBP FROM PUBLIC;  
REVOKE ALL ON SCHEMA PUBLIC FROM PUBLIC;
```

Slika 56- Primer koji se koristi u praksi kod PostgreSQL baze podataka kako bi se uklonio pristup korisnicima svim privilegijama

5. ZAKLJUČAK

Pregledom seminarskog rada i generalnim sticanjem novih znanja o PostgreSQL bazi podataka, uočava se da je sigurnost baze glavni prioritet i prilikom rada sa bazom podataka, ali i u toku procesa projektovanja i održavanja iste.

Očuvanje sigurnosti zahteva primenu određenih mera i tehnika, koje ukoliko se pravovremeno izaberu i pravilno iskoriste, utiču na kreiranje izuzetno visokog nivoa bezbednosti, pružajući veliki otpor napadačima, a garantujući jak integritet podataka korisnicima PostgreSQL sistema.

PostgreSQL baza podataka, sigurnost postiže i održava na razne načine, kako je i prikazano u radu, počevši od ostvarivanja sigurnosti na mrežnom nivou korišćenjem Unix Domain i TCP/IP soketa, firewall-a, preko autentifikacionih mehanizama i kreiranja TLS sertifikata, do kreiranja uloga i privilegija kojima se kontroliše pristup podacima, uključujući i bezbednost na nivou redova tabele baze podataka, ali i primene auditinga kao mehanizma odbrane od SQL injektovanih napada, zaključno se pričom o enkripciji i njenom uticaju na bezbednost. Svaki navedeni element, opisan je i ispraćen primerima koji ga dodatno objašnjavaju, pokazujući koliko su važni i pojedinačno, ali i kada se kombinuju, što dodatno čini PostgreSQL dobrim izborom za organizacije koje rade sa velikim količinama poverljivih informacija.

PostgreSQL pored opisanih mehanizama održavanja sigurnosti, u praksi vrlo često koristi i fizičko uređenje i ograničenje pristupa podacima, ali i razne restriktivne mere, pogotovu one koje se odnose na regulisanje public klauzole koja postoji kod ove baze podataka, a vrlo često može biti jedan od problema koji narušavaju sigurnost. Pomenute tehnike, opisane su u okviru poslednjeg poglavlja seminarskog rada, zajedno sa procesom sprečavanja eksternih veza ka bazi i načinima šifriranja podataka, koji se takođe vrlo često koriste.

Iz svega priloženog, može se zaključiti da sigurnost baza podataka predstavlja aspekt kome je neophodno posvetiti dosta pažnje, da se i metode koje se koriste za njenu garanciju, užurbano razvijaju i ažuriraju prateći novine u razvoju samog PostgreSQL sistema, pa je neophodno konstantno ostati u toku sa novim načelima sigurnosti i efikasno ih primeniti za svoje potrebe.

6. SPISAK KORIŠĆENE LITERATURE

- [1] Database Security, Imperva
Dostupno: <https://www.imperva.com/learn/data-security/database-security/>
- [2] M. Rouse, "Database Security", 2022.
Dostupno: <https://www.techopedia.com/definition/29841/database-security>
- [3] "What is database security", IBM
Dostupno: <https://www.ibm.com/topics/database-security>
- [4] J. Williams, "How to Secure PostgreSQL Database – Tips and Tricks", 2023.
Dostupno: <https://blog.devart.com/how-to-secure-postgresql-database.html>
- [5] Postgre Security
Dostupno: <https://satoricyber.com/postgres-security/3-pillars-of-postgresql-security/>
- [6] D. Page, "How to Secure PostgreSQL: Security Hardening Best Practices & Tips"
Dostupno: <https://www.enterprisedb.com/blog/how-to-secure-postgresql-security-hardening-best-practices-checklist-tips-encryption-authentication-vulnerabilities#encrypt>
- [7] PostgreSQL Documentation- Security
Dostupno: <https://www.postgresql.org/docs/7.0/security17840.htm>
- [8] R. Tkachenko, "Securing Your PostgreSQL Database", 2022.
Dostupno: <https://goteleport.com/blog/securing-postgres-postgresql/>
- [9] Client Authentication, PostgreSQL Documentation
Dostupno: <https://www.postgresql.org/docs/current/client-authentication.html>
- [10] Authentication Methods, PostgreSQL Documentation
Dostupno: <https://www.postgresql.org/docs/>
- [11] PostgreSQL Documentation- Privileges
Dostupno: <https://www.postgresql.org/docs/13/ddl-priv.html>
- [12] Row Level Security (RLS)
Dostupno: <https://satoricyber.com/postgres-security/postgres-row-level-security/>
- [13] PostgreSQL Documentation-Row Security Policies
Dostupno: <https://www.postgresql.org/docs/15/ddl-rowsecurity.html>
- [14] C. Strong, "Preventing SQL Injection Attacks in Postgres", 2020.
Dostupno: <https://www.crunchydata.com/blog/preventing-sql-injection-attacks-in-postgresql>
- [15] 3 Postgres Audit Methods: How to Choose
Dostupno: <https://satoricyber.com/postgres-security/postgres-audit/>
- [16] PostgreSQL Documentation- Encryption Options
Dostupno: <https://www.postgresql.org/docs/current/encryption-options.html>
- [17] F. Schildorfer, "A Guide to Transparent Data Encryption in PostgreSQL", 2021
Dostupno: <https://arctype.com/blog/transparent-data-encryption/>
- [18] 3 Pillars of PostgreSQL Security
Dostupno: <https://satoricyber.com/postgres-security/3-pillars-of-postgresql-security/>
- [19] H.J. Schonig, "POSTGRES SQL SECURITY: THINGS TO AVOID IN REAL LIFE", Cybertec, 2021.
Dostupno: <https://www.cybertec-postgresql.com/en/postgresql-security-things-to-avoid-in-real-life/>