UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

SIGURNOST KOD POSTGRESQL BAZE PODATAKA

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Predmet: Sistemi za upravljanje bazama podataka

Student:

Ivana Petković, br. ind. 1787

Mentor:

Doc. Dr Aleksandar Stanimirović

Niš, maj 2024. godina

Sadržaj

1. Uvod

Sigurnost baze podataka odnosi se na tri ključna svojstva baze podataka – poverljivost(tajnost), integritet i dostupnost. Poverljivost znači da korisnik ne može da pristupa onome što nije namenjeno njemu. Integritet da korisnik ne može da modifikuje ono što mu nije dozvoljeno, a dostupnost da korisnik može da vidi i modifikuje samo ono što mu je dozvoljeno. Ova svojstva moraju biti ispunjena kako bi ceo sistem koji uključuje tu bazu nesmetano radio. Za najbolja rešenja nije dovoljno samo izolovano posmatrati sigurnost baze podataka, već je neophodno posmatrati sigurnost sistema u celini. Međutim, osiguravanje baze podataka od ključnog je značaja za sigurnost celog sistema. Postoji više aspekata koji utiču na sigurnost baze podataka i više načina za njihovo ostvarenje, o čemu će biti reči u okviru ovog rada.

Na početku ćemo se baviti sigurnošću baze podataka u globalu kako bismo uvideli njen značaj. Nakon toga prelazimo na konkretnu bazu – PostgreSQL, gde ćemo sigurnost proučiti specifično vezano za pomenuti DBMS. Na kraju će biti izložen kratak zaključak nakon koga sledi pregled literature koja je korišćena u radu.

2. Sigurnost baza podataka

Sigurnost baze podataka odnosi se na niz alata, kontrola i mera dizajniranih za uspostavljanje i očuvanje poverljivosti, integriteta i dostupnosti baze podataka. Od ključnog je značaja za sigurnost čitave aplikacije. U najvećem broju povreda podataka poverljivost je element koji je ugrožen.

Pod bezbednošću baze podataka podrazumeva se zaštita podataka u bazi podataka, sistema za upravljanje bazom podataka(DBMS), svih povezanih aplikacija, fizičkog servera baze podataka ili servera virtuelne baze podataka sa hardverom koji je u njenoj osnovi, računarske ili mrežne infrastrukture koja se koristi za pristup bazi podataka.

Sigurnost baze podataka je složen i izazovan poduhvat koji uključuje sve aspekte tehnologija i praksi informacione bezbednosti. Pored toga, sigurnost je prirodno u suprotnosti sa upotrebljivošću baze podataka. Što je baza podataka pristupačnija i upotrebljivija, to je ranjivija na bezbednosne pretnje; što je baza podataka neranjivija na pretnje, teže joj je pristupiti i koristiti je. Ovaj paradoks se naziva Andersonovo pravilo.

2.1. Zbog čega je od tolikog značaja?

Baza podataka je ključna komponenta svakog softverskog sistema i njena uloga je u čuvanju podataka koje aplikacija koristi. Stoga povreda ovih podataka ima značajan negativan efekat na sistem. Prema definiciji, povreda podataka je neuspeh da se održi poverljivost podataka u bazi podataka. U zavisnosti od same povrede podataka šteta može biti manja ili veća ali je sigurno da predstavlja značajan problem.

Povreda podataka može prouzrokovati:

- **Kompromitovanje intelektualne svojine**

Intelektualna svojina – poslovne tajne, izumi i slično – mogu biti od kritičnog značaja za održavanje konkurentske prednosti na tržištu. Ako je ta intelektualna svojina ukradena ili razotkrivena, prednost na tržištu može biti teško ili nemoguće povratiti.

- **Narušavanje reputacije brenda**

Jednom kada je reputacija brenda ili kompanije narušena jako teško ju je povratiti. Ukoliko je poverenje klijenata iznevereno i osećaju da vam ne mogu verovati da ćete zaštititi svoje podatke moguće je da više neće biti voljni da kupe vaše proizvode ili usluge ili posluju sa vašom kompanijom.

- **Kazne**

Važno je naglasiti da kompanija koja pretrpi povredu podataka može dobiti i novčanu kaznu zbog nepoštovanja globalnih propisa.

- **Prekid rada**

Povreda podataka može dovesti do prekida rada ukoliko sistem nije u stanju da nastavi sa radom. To dalje može dovesti do velikih gubitaka.

- **Troškove otklanjanja povreda i obaveštavanja klijenata**

Povreda podataka može izazvati velike troškove. Pored samih troškova popravke nastale povrede podataka i štete, popravku povređenih sistema, potrebno je i obaveštavanje klijenata o nastalom stanju, može biti potrebno i platiti forenzičke i istražne aktivnosti, upravljanje kriznim situacijama i još mnogo toga. Generalno povreda podataka može prouzrokovati jako velike troškove.

2.2. Najčešći faktori narušavanja sigurnosti

Mnogo je razloga zbog kojih dolazi do narušavanja sigurnosti baze podataka. Najčešće pretnje i izazovi u očuvanju sigurnosti baze podataka su sledeći:

- **Unutrašnje pretnje**

Unutrašnja pretnja je bezbednosna pretnja iz bilo kog od tri izvora sa privilegovanim pristupom bazi podataka:

1. Zlonamerni insajder koji namerava da učini štetu.
2. Nemaran insajder koji pravi greške koje čine bazu podataka podložnom napadima.
3. Infiltrator, autsajder koji na neki način dobija kredencijale putem šeme, kao što je pecanje ili dobijanjem pristupa samoj bazi podataka kredencijala.

Unutrašnje pretnje su među najčešćim uzrocima narušavanja bezbednosti baze podataka i često su rezultat dozvoljavanja prevelikom broju zaposlenih da imaju privilegovane kredencijale za korisnički pristup.

- **Ljudska greška**

Nesrećni slučajevi, slabe lozinke, deljenje lozinki i druga nemarna i slična ponašanja korisnika su uzrok skoro polovine(49%) svih prijavljenih povreda podataka.

- **Eksploatacija ranjivosti baze podataka**

Hakeri žive od toga što ciljaju i pronalaze ranjivosti u svim vrstama softvera, uključujući softver za upravljanje bazama podataka. Svi glavni dobavljači softvera za komercijalne baze podataka i platforme za upravljanje bazama podataka otvorenog koda izdaju redovna bezbednosna rešenja da bi se pozabavile ovim ranjivostima, ali ako se ne primene na vreme to povećava izloženost napadima.

- **SQL ili NoSQL injection napadi**

Ovo je pretnja koja je specifična za bazu podataka. Ona uključuje umetanje proizvoljnih SQL ili NoSQL stringova u upite baze podataka koje opslužuju veb aplikacije ili HTTP zaglavlja. Organizacije koje ne prate bezbedne prakse kodiranja i ne vrše redovno testiranje ranjivosti otvorene su za ove napade.

- **Eksploatacija prekoračenja bafera**

Prekoračenje bafera se dešava kada proces pokuša da upiše više podataka nego što je dozvoljeno u blok memorije fiksne dužine. Napadači mogu koristiti višak podataka, koji se čuvaju na susednim memorijskim adresama, kao osnovu za početak napada.

- **Zlonamerni programi**

Zlonamerni program(Malware) je softver koji je posebno napisan s namerom da iskoristi ranjivosti ili na drugi način ošteti bazu podataka. Malver može da stigne preko bilo kog krajnjeg uređaja koji se povezuje na mrežu baze podataka.

- **Napadi na rezervne kopije**

U slučajevima kada rezervne kopije nisu zaštićene u onoj meri u kojoj je to obezbeđeno primarnoj bazi podataka, rezervne kopije postaju podložnije napadima. Ove pretnje pogoršavaju sledeći faktori: rastući obim podataka, širenje infrastrukture, sve stroži regulatorni zahtevi i nedostatak veština u sajber bezednosti.

- **DoS i DDoS napadi**

U napadima uskraćivanja usluga(Denial-of-Service – DoS), napadač zasipa ciljni server, u ovom slučaju server baze podataka, sa toliko zahteva da server više ne može da ispunjava legitimne zahteve stvarnih korisnika i često server postaje nestabilan ili se ruši.

U distribuiranom napadu uskraćivanja usluga(Distributed Denial-of-Service – DDoS), plavljenje dolazi sa više servera, što otežava zaustavljanje napada.

2.3. Mehanizmi zaštite

Baza podataka mora biti zaštićena od neovlašćenih ili zlonamernih radnji. Kao što je već naglašeno, ona je deo celokupne zaštite. Stoga je potrebno voditi računa i o sigurnosti operativnog sistema, mreže itd. kako bi se obezbedila adekvatna zaštita čitavog softvera.

Za očuvanje sigurnosti baze podataka postoje ugrađeni mehanizmi u sistemima za upravljanje bazom podataka.

Za korisnike koji imaju fizički pristup neophodna je identifikacija korisnika. Odnosno autentifikacija korisnika nekom metodom autentifikacije, kako bi se osiguralo da je u pitanju odgovarajući korisnik.

Kontrola pristupa mora postojati kako bi svojstva koja definišu sigurnost baze podataka bila ispunjena. Stoga korisnici prostupaju podacima u bazi podataka u zavisnosti od ovlašćenja i prava pristupa koja imaju.

Politika pristupa specifira ko je autorizovan da vrši određene akcije. Mehanizam sigurnosti dozvoljava sprovođenje izabrane pollitike sigurnosti. Osnovni mehanizmi zaštite na nivou DBMS-a su: **diskreciona kontrola pristupa(Discretionary access control – DAC)**, **obavezna kontrola pristupa (Mandatory access control – MAC**) i **kontrola pristupa zasnovana na ulogama (Role-Based access control - RBAC).**

Diskreciona kontrola pristupa se zasniva na konceptu prava pristupa ili privilrgija za objekte(tabele i poglede) i mehanizmima dodele/oduzimanja prava korisnicima. Onaj korisnik koji je kreator objekta automatski dobija sve privilegije nad kreiranim objektom. DBMS čuva informacije o tome kad dobija i gubi privilegije i obezbeđuje da se prihvate zahtevi samo od korisnika koji ima odgovarajuće privilegije za posmatrani objekat (u trenutku slanja zahteva).

Obavezna kontrola pristupa zasniva se na politikama sigurnosti na nivou celog sistema koji ne može da promeni neki od korisnika. Svaki objekat baze podataka ima dodeljenu tzv. klasu sigurnosti (security class). Svaki subjekat (korisnik ili korisnička aplikacija) ima dodeljenu dozvolu (clearance) za klasu sigurnosti. Pravila koja se zasnivaju na klasama sigurnosti i dozvolama upravljaju ko može da izvrši operacije čitanja/pisanja za objekte. Većina komercijalnih sistema ne podržava MAC, dok verzije nekih DBMS-a ga podržavaju i koriste za specijalizovane primene.

Kontrola pristupa zasnovana na ulogama daje privilegije pristupa na osnovu posla koji obavljaju pojedinačni korisnici. Ograničava pristup tako da korisnik može da pristupi samo onim

resursima koji su mu potrebni za obavljanje posla. Implementacija ovog mehanizma zahteva definisanje različitih uloga unutar organizacije i određivanje da li i u kom stepenu te uloge treba da imaju pristup svakom resursu.

2.4. Najbolje prakse

S obzirom da su baze podataka dostupne preko mreže, svaka bezbednosna pretnja unutar mreže ili dela mrežne infrastrukture je takođe pretnja bazi podataka, a svaki napad koji utiče na uređaj ili radnu stanicu korisnika može da ugrozi bazu podataka. Dakle, bezbednost baze podataka se proteže daleko izvan granica same baze podataka.

U cilju očuvanja bezbednosti baze podataka važno je razmotriti sledeće oblasti:

- **Fizička sigurnost**

  Bez obzira da li se server baze podataka nalazi lokalno ili je u oblaku, on mora biti lociran u bezbednom okruženju sa kontrolisanim klimatskim uslovima.

- **Administrativne i mrežne kontrole pristupa**

  Pristup bazi podataka treba da ima minimalni broj korisnika, a njihove dozvole treba da budu ograničene na minimalne nivoe neophodne da bi obavljali svoj posao. Takođe, pristup mreži treba da bude ograničen na minimalni nivo neophodnih dozvola.

- **Bezbednost korisničkih naloga i uređaja**

  Potrebno je biti svestan ko pristupa bazi podataka i kada i kako se podaci koriste. Rešenja za praćenje podataka mogu vas upozoriti ako su aktivnosti sa podacima neobične ili izgledaju rizične. Svi korisnički uređaji koji se povezuju na mrežu u kojoj se nalazi baza podataka treba da budu fizički bezbedni podvrgnuti bezbednosnim kontrolama u svakom trenutku.

- **Enkripcija**

  Svi podaci, uključujući podatke koji su u bazi podataka i podatke o kredencijalima korisnika treba da budu zaštićeni enkripcijom – i dok su u mirovanju i dok su u tranzitu.

- **Sigurnost softvera baze podataka**

  Poželjno je koristiti najnoviju verziju softvera za upravljanje bazom podataka i ažurirati sve bezbednosne popravke čim postanu dostupne.

- **Sigurnost aplikacija i veb servera**

  Svaka aplikacija ili veb server koji stupa u interakciju sa bazom podataka može biti kanal za napad i trebalo bi da bude podvrgnut stalnom testiranju.

- **Sigurnost rezervnih kopija**

  Sve rezervne kopije, kopije ili slike baze podataka moraju biti podvrgnute istim ili jednako strogim bezbednosnim kontrolama kao i sama baza podataka.

- **Revizija**

  Zabeležiti sve prijave na server baze podataka i operativni sistem i evidentirajte sve operacije koje se izvode na osetljivim podacima. Takođe, potrebno je obavljati revizije standarda bezbednosti baze podataka.

3. Sigurnost kod PostgreSQL baze podataka

U okviru ovog poglavlja sagledaćemo aspekte sigurnosti konkretno kod PostgreSQL baze podataka. Verzija PostgreSQL-a koja je razmatrana i na koju se odnose primeri je 16. Poglavlje je podeljeno na delove koji predstavljaju ključne aspekte sigurnosti: Autentifikacija klijenta, Korisničke uloge i Kontrola pristupa. Međutim, granice između ovih poglavlja nisu stroge već postoje značajne interakcije i preklapanja među njima. Autentifikacija je usko povezana sa korisničkim ulogama jer jednom kada je identitet korisnika potvrđen uloge se koriste za obezbeđivanje njihovih prava i privilegija u sistemu. Korisničke uloge su povezane sa kontrolom pristupa jer se kroz uloge određuje ko može pristupiti kojim resursima i na koji način. Kontrola pristupa se nadovezuje na korisničke uloge jer se prava pristupa definišu na nivou uloga.

Osim toga značajni za sigurnost kod svih pa i kod PostgreSQL baze podataka jesu revizija i enkripcija. Oni pružaju dodatne mere sigurnosti i svaka od njih ponaosob može predstavljati zasebnu celinu za istraživanje. Revizija se odnosi na praćenje i bečeženje aktivnosti unutar baze podataka radi sigurnosti i usklađenosti sa propisima. Revizija pristupa se koristi za praćenje i beleženje pokušaja pristupa različitim resursima. Revizija se oslanja na informacije o autentifikaciji i korisničkim ulogama kako bi se tačno beležile i analizirale aktivnosti specifičnih korisnika. Kontrola pristupa utiče na to koje aktivnosti će biti evidentirane. Enkripcija obezbeđuje zaštitu podataka tokom prenosa i u stanju mirovanja. Enkripcija je povezana sa autentifikacijom jer često koristi sertifikate za sigurnu komunikaciju. Enkriptovani podaci utiču na kontrolu pristupa jer enkripcija pruža dodatni sloj zaštite.

3.1. Autentifikacija klijenta

Autentifikacija je proces kojim server baze podataka uspostavlja identitet klijenta i određuje da li je klijentskoj apliakciji (ili korisniku koji pokreće klijentsku aplikaciju) dozvoljeno da se poveže sa traženim korisničkim imenom baze podataka.
PostgreSQL pruža više različitih metoda za autentifikaciju klijenata:

- Trust autentifikacija, koja jednostavno veruje da su korisnici oni za koje se predstavljaju.
- Password autentifikacija, odnosno autentifikacija lozinkom, koja zahteva od korisnika da unesu lozinku.
- GSSAPI autentifikacija, koja se oslanja na GSSAPI-kompatabilnu bezbednosnu biblioteku. Obično se koristi za pristup serveru za autentifikaciju kao što je Kerberos ili Microsoft Active Directory server.
- SSPI autentifikacija, koja koristi Windows-specifičan protokol koji je sličan GSSAPI-ju.
- Provera identiteta, koja se oslanja na uslugu „Identification Protocol“ (RFC 1413) na klijentskoj mašini. (Na lokalnim Unix-soket vezama, ovo se tretira kao peer autentifikacija)
- Peer autentifikacija, se oslanja na mogućnosti operativnog sistema za identifikaciju procesa na drugom kraju lokalne veze. Ovo nije podržano za udaljene veze.
- LDAP autentifikacija, koja se oslanja na LDAP server za autentifikaciju.
- RADIUS autentifikacija, koja se oslanja na RADIUS server za autentifikaciju.
- Autentifikacija sertifikatom, koja zahteva SSL vezu i vrši autentifikaciju korisnika proverom SSL sertifikata koje šalju.
- PAM autentifikacija, koja se oslanja na PAM(Pluggable Authentication Modules) biblioteku.
- BSD autentifikacija, se oslanja na BSD autentifikacioni okvir(trenutno dostupna samo na

OpenBSD-u).

Koji metod autentifikacije ćete izbrati zavisi konkretno od karakteristika i potreba vašeg sistema.
Peer autentifikacija se obično preporučuje za lokalne veze, iako bi trust autentifikacija mogla biti dovoljna u nekim okolnostima. Autentifikacija lozinkom je najlakši izbor za udaljene veze. Sve ostale opcije zahtevaju neku vrstu spoljne bezbednosne infrastrukture(obično server za autentifikaciju ili autoritet za sertifikaciju za izdavanje SSL sertifikata) ili su specifične za neku platformu.
U okviru ovog poglavlja nećemo detaljno prolaziti kroz svaku od metoda autentifikacija, već ćemo samo videti kako izgleda LDAP i SSL autentifikacija. Više o svakoj od ostalih može se naći u PostgreSQL dokumentaciji.
Kada se koristi neki spoljašnji sistem za autentifiakciju potrebno je usaglasiti imena korisnika na tom sistemu sa imenima u bazi podataka, što se ostvaruje pomoću mapa korisničkih imena. Korisnička imena PostgreSQL baze podataka su logički odvojena od korisničkih imena operativnog sistema u kojem server radi. Ako svi korisnici određenog servera takođe imaju naloge na mašini servera, ima smisla dodeliti korisnička imena baze podataka koja odgovaraju njihovim korisničkim imenima unutar operativnog sistema. Međutim, server koji prihvata udaljene veze može imati mnogo korisnika baze podataka koji nemaju nalog lokalnog operativnog sistema i u takvim slučajevima ne mora postojati veza između korisničkih imena baze podataka i korisničkih imena operativnog sistema.
Pre svega potrebno je upoznati se sa pg_hba.conf fajlom, koji predstavlja osnovni fajl za autentifikaciju klijenata.
Autentifikacija je od izuzetnog značaja za sigurnost baze podataka jer je izuzetno važno ograničiti koji korisnici baze podataka mogu da se povežu.

3.1.1. pg_hba.conf fajl

Autentifikaciju klijenta kontroliše konfiguracioni fajl pod nazivom **pg_hba.conf** i smešten je u direktorijumu podataka klastera baze podataka. **HBA** predstavlja skraćenicu za autentifikaciju zasnovanu na hostu(**host-based authentication**). Podrazumevana datoteka se instalira kada initdb inicijalizuje direktorijum podataka.

Opšti format datoteke pg_hba.conf je skup zapisa, po jedan po redu. Prazne linije se ignorišu, kao i svaki tekst posle znaka komentara #. Zapis se može nastaviti na sledeći rede tako što se red završava obrnutom kosom crtom. Zapis se sastoji od više polja koja su odvojena razmacima i/ili tabulatorima. Polja mogu da sadrže razmak ako je vrednost polja u dvostrukim navodnicima.

Svaki zapis za autentifikaciju specificira tip veze, opseg IP adresa klijenta(ako je relevantan za tip veze), ime baze podataka, korisničko ime i autentifikacioni metod koji će se koristiti. Prvi zapis sa odgovarajućim tipom veze, adresom klijenta, traženom bazom podataka i korisničkim imenom se koristi. Ako autentifikacija ne uspe sa izabranim zapisom, naredni zapisi se ne uzimaju u obzir. Ako se nijedan zapis ne podudara, pristup biva odbijen.

Svaki zapis može biti direktiva uključivanja(include directive) ili zapis za potvrdu autentičnosti. Include direktive definišu datoteke koje se mogu uključiti, koje sadrže dodatne zapise. Ti zapisi će biti umetnuti na mestu ovih direktiva.

Zapis može imati nekoliko formata, kao što je prikazano na slici 1.

```
local             database  user  auth-method [auth-options]
host              database  user  address     auth-method  [auth-options]
hostssl           database  user  address     auth-method  [auth-options]
hostnossl         database  user  address     auth-method  [auth-options]
hostgssenc        database  user  address     auth-method  [auth-options]
hostnogssenc      database  user  address     auth-method  [auth-options]
host              database  user  IP-address  IP-mask      auth-method  [auth-options]
hostssl           database  user  IP-address  IP-mask      auth-method  [auth-options]
hostnossl         database  user  IP-address  IP-mask      auth-method  [auth-options]
hostgssenc        database  user  IP-address  IP-mask      auth-method  [auth-options]
hostnogssenc      database  user  IP-address  IP-mask      auth-method  [auth-options]
include           file
include_if_exists file
include_dir       directory
```

Slika 1. Formati zapisa u pg_hba.conf fajlu

Značenja polja:

1) local – ovaj zapis odgovara pokušajima povezivanja pomoću Unix-domen soketa.
2) host – ovaj zapis odgovara pokušajima povezivanja putem TCP/IP protokola. Host zapisi odgovaraju SSL i ne-SSL pokušajima povezivanja, kao i GSSAPI enkriptovanim i neenkriptovanim pokušajima povezivanja.
3) hostssl - ovaj zapis odgovara pokušajima povezivanja putem TCP/IP protokola ali samo kada je veza napravljena sa SSL šifrovanjem.
4) hostnossl – ovaj tip zapisa ima suprotno ponašanje od hostssl-a. Odgovara samo pokušajima povezivanja preko TCP/IP protokola koji ne koriste SSL.
5) hostgssenc - ovaj zapis odgovara pokušajima povezivanja putem TCP/IP protokola ali samo kada je veza napravljena pomoću GSSAPI enkripcije.
6) hostnogssenc - ovaj tip zapisa ima suprotno ponašanje od hostgssenc -a. Odgovara samo pokušajima povezivanja preko TCP/IP protokola koji ne koriste GSSAPI.
7) include – ova linija će biti zamenjena sadržajem date datoteke.
8) include_if_exists – ova linija će biti zamenjena sadržajem datoteke, ako datoteka postoji. U suprotnom se evidentira porzka koja označava da je datoteka preskočena.
9) include_dir – ova linija će biti zamenjena sadržajem svih datoteka koje se nalaze u direktorijumu

database – određuje sa kojim imenom/ima baze podataka se ovaj zapis poklapa. Vrednost *all* označava da odgovara svim bazama podataka. Vrednost *sameuser* navodi da se zapis podudara ako zahtevana baza podataka ima isto ime kao traženi korisnik. Vrednost *samerole* navodi da traženi korisnik mora biti član uloge sa istim imenom kao zahtevana baza podataka. (*samegroup* je zastarela ali još uvek prihvaćena oznaka za *samerole*)
user – određuje kojim korisničkim imenima baze podataka ovaj zapis odgovara. Vrednost *all* označava da odgovara svim korisnicima.
address – određuje adresu/adrese klijentske mašine sa kojom se ovaj zapis podudara. Ovo polje može sadržati ili ime hosta, ili opseg IP adresa ili jednu od posebnih ključnih reči.
IP-address, **IP-mask** – ova dva polja se mogu koristiti kao alternativa zapisu IP adresa/dužina maske. Ova polja se ne primenjuju na local zapise.
auth-method – određuje autentifikacioni metod koji će se koristiti kada se veza podudara sa ovim zapisom. Mogući izbori su: trust, reject, scram-sha-256, md5, password, gss, sspi, ident, peer, ldap, radius, cert, pam, bsd. Sve opcije su case-sensitive i validne su samo napisane malim slovima.
auth-options – nakon auth-method polja, mogu postojati polja oblika name=value koja specificiraju opcije za metod autentifikacije. Za različite metode autentifikacije dostupne su različite opcije.

3.1.2. Mapiranje korisničkih imena

U nekim slučajevima, korisničko ime za autentifikaciju se razlikuje od PostgreSQL korisničkog imena. Na primer, ovo se može desiti kada se koristi eksterni sistem za autentifikaciju, kao što je provera autentičnosti sertifikata ili bilo koji drugi eksterni sistem. U ovim slučajevima možda ćete morati da omogućite eksterno autentifikovanom korisniku da se poveže kao više korisnika baze podataka. U takvim slučajevima možete odredti pravila za mapiranje spoljašnjeg korisničkog imena u odgovarajućeg korisnika(ulogu) baze podataka.

Pripremite listu korisničkih imena iz spoljnog sistema za autentifikaciju i odlučite sa kojim korisnicima baze podataka im je dozvoljeno da se povežu, odnosno koji spoljni korisnici se mapiraju na koje korisnike baze podataka.

Da biste koristili mapiranje korisničkog imena, navedite map=map-name u polju opcija u pg_hba.conf datoteci. Ova opcija je podržana za sve metode autentifikacije koja primaju spoljna korisnička imena. Pošto za različite veze mogu biti potrebna različita mapiranja, ime mape koja će se koristiti je navedeno u parametru map-name u pg_hba.conf fajlu da bi se naznačilo koju mapu treba koristiti za svaku pojedinačnu vezu.

Mape korisničkih imena su definisane u datoteci koja se podrazumevano zove pg_ident.conf(jer je prvobitno korišćena za metod provere identiteta) i čuva se u direktorijumu podataka klastera. Moguće je promeniti ime ove datoteke preko konfiguracionog parametra ident_file u postgresql.conf.

Linije u datoteci mapa imaju format:

```
map-name system-username database-username
```

Ovo treba tumačiti na sledeći način „sistemsko korisničko ime“ je dozvoljeno da se poveže kao korisničko ime baze podataka“, a ne „svaki put kada se sistemsko korisničko ime poveže, moraće da koristi korisničko ime baze podataka“.
Ovde je map-name vrednost mapirane opcije iz odgovarajuće linije u pg_hba.conf fajlu, system-username je korisničko ime pod kojim je spoljni sistem potvrdio autentičnost veze, a database-username je korisnik baze podataka sa kojim je ovom korisniku sistema dozvoljeno da se poveže. Istom korisniku sistema može biti dozvoljeno da se poveže kao više korisnika baze podataka, tako da ovo nije mapiranje 1:1, već lista dozvoljenih korisnika baze podataka za svakog korisnika sistema.
Ako system-username počinje kosom crtom (/), onda se ostatak tretira kao regularni izraz(Regular Expression – regex), a ne kao string koji se direktno podudara i moguće je koristiti string \1 u database-username za upućivanje na deo obuhvaćen u zagradama u regularnom izrazu.

Primer 1: Razmotrimo sledeće:

```
salesmap /^(.*)@sales\.comp\.com$ \1
salesmap /^(.*)@sales\.comp\.com$ sales
salesmap manager@sales.comp.com auditor
```

Ovo će omogućiti svakom korisniku autentifikovanom sa `@sales.comp.com` imejl adresom da se poveže i kao korisnik baze podataka koji je jednak imenu pre @ prijavi za njihovu adresu e-pošte i kao korisnik prodaje. Oni će dodatno omogućiti `manager@sales.comp.com` da se poveže kao revizor.

Zatim, modifikujte liniju pg_hba.conf kako biste specificirali opciju map=salesmap, kako bi mapiranje bilo moguće.

Nakon autentifikacije veze pomoću spoljnog sistema za autentifikaciju, PostgreSQL će obično proveriti da li se korisničko ime sa spoljnom autentifikacijom poklapa sa korisničkim imenom baze podataka sa kojim korisnik želi da se poveže, i odbaciće vezu ako se ova dva ne podudaraju. Ako postoji parametar map= specificiran za trenutnu liniju u pg_hba.conf, onda će sistem skenirati mapu liniju po liniju i pustiti klijenta da nastavi da se povezuje ako se pronađe podudaranje.

3.1.3. Lightweight Directory Access Protocol(LDAP) metod autentifikacije

Ovaj metod autentifikacije funkcioniše slično kao i autentifikacija lozinkom, osim što koristi LDAP server kao metod provere lozinke. LDAP se koristi samo za validaciju parova korisničkog imena/lozinke. Stoga korisnik mora već postojati u bazi podataka pre nego što se LDAP može koristiti za autentifikaciju.

LDAP autentifikacija može raditi u dva režima. U prvom režimu, koji se zove jednostavni režim povezivanja, server će se vezati za prepoznatljivo ime u formatu prefiks korisničko ime sufiks. Obično, prefiks se koristi da se specificira cn= ili DOMAIN\ u okruženju Active Directory.

U drugom režimu koji se zove režim pretrage i vezivanja(search+bind), server se prvo povezuje na LDAP direktorijum sa fiksnim korisničkim imenom i lozinkom, navedenim sa ldapibindnn i ldapbindpasswd, i vrši pretragu korisnika koji pokušava da se prijavi na bazu podataka. Ako korisničko ime i lozinka nisu konfigurisani, pokušaće se anonimno povezivanje na direktorijum. Pretraga će biti izvršena preko podstabla na ldapbasedn i pokušaće da napravi podudaranje sa atributom navedenim u ldapsearchattribute. Kada je korisnik pronađen u ovoj pretrazi, server se diskonektuje i ponovo povezuje na direktorijum kao taj korisnik, koristeći lozinku koju je naveo klijent, kako bi verifikovao da je prijava ispravna. Ova metoda omogućava značajno veću fleksibilnost u tome gde su korisnički objekti locirani u direktorijumu, ali će uzrokovati uspostavljanje dve odvojene veze sa LDAP serverom.

Sledeće konfiguracione opcije se koriste u oba moda:
1) ldapserver – Imena ili IP adrese LDAP servera za povezivanje. Može se navesti više odvojenih servera.
2) ldapport – Broj porta na LDAP serveru za povezivanje. Ako port nije naveden, koristiće se podrazumevana postavka porta LDAP biblioteke.
3) ldapscheme – Podesiti na ldaps da biste koristili LDAPS. To je nestandardni način korišćenja LDAP-a preko SSL-a i podržan je od strane nekih implementacija LDAP servera. Za alternativu pogledati ldaptls opciju.
4) ldaptls – postaviti na 1 da veza između PostgreSQL-a i LDAP servera koristila TLS šifrovanje.

Treba imati na umu da ldapscheme ili ldaptls šifruje samo saobraćaj između PostgreSQL servera i LDAP servera. Veza između PostgreSQL servera i PostgreSQL klijenta će i dalje biti nešifrovana osim ako se i tamo ne koristi SSL.

Dok se ove koriste samo u jednostavnom modu rada:
1)ldapprefix – string koji se stavlja ispred korisničkog imena prilikom formiranja DN(Distinguished Name)-a za povezivanje, kao kada se radi autentifikacija kod jednostavnog vezivanja.
2) ldapsuffix – string koji se stavlja iza korisničkog imena prilikom formiranja DN-a za povezivanje takođe kao kod autentifikacije jednostavnog vezivanja.

A sledeće samo u režimu pretrage i vezivanja:
1) ldapbasedn – root DN da bi se započela pretraga korisnika.
2) ldapbinddn – DN korisnika za povezivanje sa direktorijumom za obavljanje pretrage.
3) ldapbindpasswd – lozinka za korisnika da se poveže sa direktorijumom.
4) ldapsearchattribute – atribut koji se podudara sa korisničkim imenom u pretrazi. Ako nijedan atribut nije naveden, koristi se uid atribut.

5) ldapsearchfilter – filter za pretragu koji se koristi kada se vrši autentifikacija u režimu pretrage i vezivanja.
6) ldapurl – RFC 4516 LDAP URL.

Primer 2: Videćemo kako podesiti PostgreSQL sistem tako da koristi LDAP za autentifikaciju.

Uverite se da se korisnička imena u bazi podataka i na vašem LDAP serveru podudaraju, jer ovaj metod funkcioniše za provere autentifikacije korisnika za korisnike koji su već definisani u bazi podataka.

1) Konfiguracija pg_hba.conf fajla i konfiuracija LDAP servera

U pg_hba.conf PostgreSQL datoteci za autentifikaciju definišemo neke opsege adresa da bismo koristili LDAP kao metod autentifikacije i potrebno je konfigurisati LDAP server za taj opseg adresa:

```
host all all 10.10.0.1/16 ldap \
ldapserver=ldap.our.net ldapprefix="cn=" ldapsuffix=",
dc=our,dc=net"
```

Potrebno je konfigurisati LDAP server tako da podržava autentifikaciju za adrese koje su navedene u pg_hba.conf fajlu. U ovom primeru ldapserver opcija specificira adresu LDAP servera, dok ldapprefix i ldapsuffix definišu prefiks i sufiks koji se koriste za formiranje DN(Distinguished Name) korisnika u LDAP stablu.

Ovo podešavanje omogućava da PostgreSQL server proverava lozinke sa konfigurisanog LDAP servera.

Korisnička prava se ne proveravaju sa LDAP servera, već se moraju definisati unutar baze podataka, koristeći komande ALTER USER, GRANT i REVOKE.

Zamena za funkciju mapiranja korisničkog imena

Iako ne možemo da koristimo funkciju mapiranja korisničkog imena sa LDAP-om, možemo postići sličan efekat na strani LDAP-a. Koristite ldapsearchattribute i režim pretrage i povezivanja da biste preuzeli ime PostgreSQL uloge sa LDAP servera.

3.1.4. SSL metod autentifikacije

Ovaj metod podrazumeva korišćenje SSL klijentskih sertifikata za izvršavanje autentifikacije. Stoga je dostupan samo za SSL konekcije. Korišćenjem ovog metoda autentifikacije server će zahtevati da klijent obezbedi važeći, pouzdan sertifikat. Klijentu se neće tražiti lozinka. CN(Common Name) atribut sertifikata će se porediti sa traženim korisničkim imenom baze podataka, i ako se podudaraju prijava će biti dozvoljena. Mapiranje korisničkih imena može se koristiti jer bi CN mogao biti različit od korisničkog imena baze podataka.

Konfiguracija SSL-a se obavlja u okviru postgresql.conf fajla.

Sledeće opcije konfiguracije su podržane za autentifikaciju SSL sertifikatom (konfiguracije dakle u okviru pg_hba.conf datoteke za određivanje pravila autentifikacije; za auth-method je izabrana opcija cert):

map – omogućava mapiranje između sistemskih i korisničkih imena baze podataka.

Nema potrebe koristiti opciju clientcert sa cert autentifikacijom jer cert autentifikacija već podrazumeva clientcert=verify-full, što znači da je sertifikat klijenta u potpunosti verifikovan.

SSL autentifikacija se može koristiti kao dodatni bezbednosni sloj, koristeći dvostruku autentifikaciju, gde klijent mora da ima važeći sertifikat da bi podesio SSL vezu i da zna lozinku

korisnika baze podataka. Takođe se može koristiti kao jedini metod autentifikacije, gde će PostgreSQL server prvo verifikovati klijentsku vezu koristeći sertifikat koji je klijent prikazao, a zatim će preuzeti korisničko ime iz istog sertifikata.

Primer 3: Videćemo kako je moguće podesiti PostgreSQL sistem tako da zahteva od klijenata da pokažu važeći X.509[1] sertifikat kao i koriste SCRAM-SHA-256 autentifikaciju lozinkom pre nego što im dozvoli povezivanje.

1)Generisanje sertifikata:

Nabavite ili generišite osnovni sertifikat i sertifikat klijenta koji će koristiti klijent koji se povezuje. Za potrebe testiranja ili za podešavanje jednog pouzdanog korisnika, možete koristiti samopotpisani sertifikat:

```
openssl genrsa 2048 > client.key
openssl req -new -x509 -key server.key -out client.crt
```

Ovim koracima se generišu privatni ključ(client.key) i samopotpisani sertifikat(client.crt) za klijenta.

2) Konfiguracija servera:

Na serveru podesiti liniju sa hostssl metodom u datoteci pg_hba.conf, a opciju clientcert postaviti na 1:

```
hostssl all all 0.0.0.0/0 scram-sha-256 clientcert=1
```

Stavite root[2] sertifikat klijenta u datoteku root.crt u direktorijumu servera podataka($PGDATA/root.crt). Ova datoteka može da sadrži više pouzdanih root sertifikata. Ako koristite centralni CA[3] verovatno imate i listu opoziva sertifikata(Certificate Revocation List – CRL[4]), koju treba staviti u root.crl datoteku i redovno ažurirati.

3)Konfiguracija klijenta:

Na klijentu, stavite klijentov privatni ključ i sertifikat u ~/.postgresql/postgresql.key i ~/.postgresql/postgresql.crt. Uverite se da datoteka sa privatnim ključem nije čitljiva svim korisnicima ili grupna pokretanjem sledeće komande:

```
chmod 0600 ~/.postgresql/postgresql.key
```

Na Windows klijentu, odgovarajuće datoteke su %APPDATA%\postgresql\postgresql.key i %APPDATA%\postgresql\postgresql.crt. Ne vrši se provera dozvole, jer se lokacija smatra sigurnom.
Ako klijentski sertifikat nije potpisan od strane glavnog CA(root autoriteta za sertifikate) već od posredničkog CA(posrednički/intermedijarni autoritet za sertifikate), onda svi posredni CA sertifikati do korenskog sertifikata takođe moraju biti smešteni u postgresql.crt datoteku.

Ako je opcija clientcert=1 postavljena za red hostssl u pg_hba.conf fajlu, onda PostgreSQL prihvata samo zahteve za povezivanje praćene validnim sertifikatom.

---

[1] X.509 – standard za digitalne sertifikate
[2] Root sertifikat je digitalni sertifikat izdat od strane CA. Obično je samopotpisan.
[3] CA – Certificate Authority, autoritet za sertifikate. Entitet koji izdaje digitalne sertifikate.
[4] Lista sertifikata koji su opozvani pre nego što im je istekao rok važenja.

Validnost sertifikata se proverava u odnosu na sertifikate koji se nalaze u root.crt datoteci u direktorijumu servera.
Ako postoji root.crl datoteka, onda se predočeni sertifikat traži u njoj i ako se pronađe, biva odbijen.
Nakon što je sertifikat potvrđen i SSL veza uspostavljena, server nastavlja sa validacijom stvarnog korisnika koji se povezuje, koristeći bilo koji metod autentifikacije koji je naveden u odgovarajućoj hostssl liniji.

Primer 4: U sledećem primeru, klijenti sa posebne adrese mogu da se povežu kao bilo koji korisnik kada koriste SSL sertifikat i moraju da navedu SCRAM-SHA-256 lozinku za veze koje nisu SSL. Klijenti sa svih ostalih adresa moraju pokazati sertifikat i koristiti SCRAM-SHA-256 autentifikaciju lozinkom:

```
host all all 10.10.10.10/32 scram-sha-256
hostssl all all 10.10.10.10/32 trust clientcert=1
hostssl all all all scram-sha-256 clientcert=1
```

3.2. Korisničke uloge

PostgreSQL upravlja dozvolama za pristup bazi podataka koristeći koncept uloga. Korisničke uloge definišu dozvole i privilegije korisnika. Uloga se može smatrati ili korisnikom baze podataka ili grupom korisnika, u zavisnosti od toga kako je postavljena. Uloge mogu posedovati objekte baze podataka (npr. tabele i funkcije) i mogu dodeliti privilegije nad tim objektima drugim ulogama. Štaviše, može se dodeliti članstvo u ulozi drugoj ulozi, čime se dozvoljava toj ulozi da koristi privilegije dodeljene drugoj ulozi. Koncept uloge obuhvata koncepte korisnika i grupa.

Uloge baze podataka su konceptualno potpuno odvojene od korisnika operativnog sistema. Uloge baze podataka su globalne u celoj instalaciji klastera baze podataka(a ne po pojedinačnoj bazi podataka).

Uloga se može kreirati na sledeći način, korišćenjem komande:

```
CREATE ROLE name;
```

Za uklanjanje postojeće uloge koristi se analogna komanda DROP ROLE:

```
DROP ROLE name;
```

Da bi se odresio skup postojećih uloga, može se ispitati sistemski katalog na sledeći način:

```
SELECT rolname FROM pg_roles;
```

A takođe i da se vide samo one koje mogu da se prijave:

```
SELECT rolname FROM pg_roles WHERE rolcanlogin;
```

Za izlistavanje postojećih uloga takođe je korisna i meta komanda psql-a \du.
Da bi se pokrenuo sistem baze podataka, sveže inicijlizovan sistem uvek sadrži jednu unapred definisanu ulogu koja može da se prijavi. Ova uloga je uvek superkorisnik i imaće isto ime kao korisnik operativnog sistema koji je inicijalizovao klaster baze podataka pomoću initdb osim ako nije navedeno drugačije ime. Ova uloga se obično naziva postgres. Da biste kreirali više uloga, prvo morate da se povežete kao ova početna uloga.
Uloga baze podataka može imati brojne atribute koji definišu njene privilegije i interakciju sa sistemom autentifikacije klijenta. Neki od atributa su: LOGIN, SUPERUSER, CREATEDB, CREATEROLE, REPLICATION, NOINHERIT, BYPASSRLS.
PostgreSQL obezbeđuje skup unapred definisanih uloga. Administratori (uključujući i uloge koje

imaju privilegiju CREATEROLE) mogu dodati(GRANT) ove uloge korisnicima i/ili drugim ulogama u njihovom okruženju, obezbeđujući tim korisnicima pristup određenim mogućnostima i informacijama. Na slici 2 su objašnjene sve predefinisane uloge koje postoje u PostgreSQL-u.

Role	Allowed Access
pg_read_all_data	Read all data (tables, views, sequences), as if having SELECT rights on those objects, and USAGE rights on all schemas, even without having it explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is being used, an administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.
pg_write_all_data	Write all data (tables, views, sequences), as if having INSERT, UPDATE, and DELETE rights on those objects, and USAGE rights on all schemas, even without having it explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is being used, an administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.
pg_read_all_settings	Read all configuration variables, even those normally visible only to superusers.
pg_read_all_stats	Read all pg_stat_* views and use various statistics related extensions, even those normally visible only to superusers.
pg_stat_scan_tables	Execute monitoring functions that may take ACCESS SHARE locks on tables, potentially for a long time.
pg_monitor	Read/execute various monitoring views and functions. This role is a member of pg_read_all_settings, pg_read_all_stats and pg_stat_scan_tables.
pg_database_owner	None. Membership consists, implicitly, of the current database owner.
pg_signal_backend	Signal another backend to cancel a query or terminate its session.
pg_read_server_files	Allow reading files from any location the database can access on the server with COPY and other file-access functions.
pg_write_server_files	Allow writing to files in any location the database can access on the server with COPY and other file-access functions.
pg_execute_server_program	Allow executing programs on the database server as the user the database runs as with COPY and other functions which allow executing a server-side program.
pg_checkpoint	Allow executing the CHECKPOINT command.
pg_use_reserved_connections	Allow use of connection slots reserved via reserved_connections.
pg_create_subscription	Allow users with CREATE permission on the database to issue CREATE SUBSCRIPTION.

Slika 2. Predefinisane uloge

Uloge pg_monitor, pg_read_all_settings, pg_read_all_stats i pg_stat_scan_tables imaju za cilj da omoguće administratorima da lako konfigurišu ulogu u svrhu nadgledanja servera baze podataka.

Uloga pg_database_owner ima jednog implicitnog člana koji zavisi od situacije, naime vlasnika trenutne baze podataka. Kao i svaka uloga, može posedovati objekte i primati privilegije pristupa. Prema tome, kada pg_database_owner ima prava u okviru baze podataka, svaki vlasnik baze podataka instanciran iz tog šablona će koristiti ta prava. On ne može biti član nijedne uloge i ne može imati neimplicitne članove. U početku, ova uloga poseduje javnu šemu, tako da svaki vlasnik baze podataka upravlja lokalnom upotrebom šeme.

Uloga pg_signal_backend je namenjena da omogući administratorima da dozvole pouzdanim ulogama, ali onima koji nisu super korisnici, da šalju signale drugim pozadinskim sistemima.

Uloge pg_read_server_files, pg_write_server_files i pg_execute_server_program imaju za cilj da dozvole administratorima da imaju uloge od poverenja koje nisu super korisnici, koji mogu da pristupaju datotekama i pokreću programe na serveru baze podataka kao korisnik pod kojim se baza podataka pokreće. Pošto ove uloge mogu da pristupe bilo kojoj datoteci na sistemu datoteka servera, one zaobilaze sve provere dozvola na nivou baze podataka kada direktno pristupaju datotekama i mogu se koristiti za dobijanje pristupa na nivou superkorisnika, zbog toga treba biti veoma oprezan kada se te uloge dodeljuju.

Administratori mogu da dodele pristup ovim ulogama od strane korisnika, jednostavno, uz pomoć GRANT komande:

```
GRANT pg_signal_backend TO admin_user;
```

3.2.1. Superuser(Superkorisnik)

PostgreSQL super korisnik je korisnik koji zaobilazi sve provere dozvola, osim prava da se prijavi. To je opasna privilegija i ne bi trebalo da se koristi nesmotreno. Mnoge baze podataka u oblaku uopšte ne dozvoljavaju ovaj nivo privilegija. Normalno je postaviti stroge kontrole na korisnike ovog tipa.

Prava na neke operacije u PostgreSQL-u nisu podrazumevano dostupna i moraju se dodeliti korisnicima. Mora ih izvršiti poseban korisnik koji ima postavljen ovaj specifičan atribut.

Na sledeći način moguće je dodati ili ukloniti privilegije super korisnika bilo kom korisniku:

- Korisnik postaje super korisnik kada je kreiran sa SUPERUSER skupom atributa:

```
CREATE USER username SUPERUSER;
```

- Korisnik može biti lišen statusa super korisnika uklanjanjem atributa SUPERUSER pomoću ove komande:

```
ALTER USER username NOSUPERUSER;
```

- Korisnik se kasnije može vratiti na status super korisnika pomoću komande:

```
ALTER USER username SUPERUSER;
```

- Kada u komandi CREATE USER nisu dati ni SUPERUSER ni NOSUPERUSER, podrazumevano ponašanje je da se kreira korisnik koji nije super korisnik (običan korisnik baze podataka).

PostgreSQL sistem dolazi sa najmanje jednim super korisnikom. Najčešće se ovaj super korisnik naziva postgres, ali povremeno, usvaja isto ime kao korisnik sistema koji poseduje direktorijum baze podataka i sa čijim pravima PostgreSQL server radi.

Pored SUPERUSER atributa, postoje dva manja atributa – CREATEDB i CREATEUSER – koja korisniku daju samo deo moći rezervisanih za super korisnike, dakle, kreiranje novih baza podataka i korisnika. Više o tome možemo videti u poglavlju – Davanje ograničenih ovlašćenja superkorisnika određenim korisnicima.

3.2.2. Davanje ograničenih ovlašćenja superkorisnika određenim korisnicima

Uloga super korisnika ima neke privilegije koje se mogu zasebno dodeliti i ulogama koje nisu superkorisnici.

Sledećom komandom moguće je dati ulozi bob mogućnost da kreira nove baze podataka:

```
ALTER ROLE BOB WITH CREATEDB;
```

Dok, sledećom komandom je moguće dozvoliti ulozi bob da kreira nove korisnike:

```
ALTER ROLE BOB WITH CREATEROLE;
```

Međutim, PostgreSQL dokumentacija upozorava da se pažljivo koristi CREATEROLE privilegija. Razlog tome je što ne postoji koncept nasleđivanja za privilegije uloge CREATEROLE. To znači da čak iako uloga nema određenu privilegiju, ali joj je dozvoljeno da kreira druge uloge, ona lako može kreirati drugu ulogu sa privilegijama drugačijim od svojih (osim za kreiranje uloga sa privilegijama superkorisnika). Na primer, ako uloga „user“ ima privilegiju CREATEROLE, ali ne i privilegiju CREATEDB, ipak može da kreira novu ulogu sa privilegijom CREATEDB. Prema tome, smatrajte uloge koje imaju CREATEROLE privilegiju gotovo ulogama superkorisnika.
Takođe, moguće je dati običnim korisnicima detaljniji i kontrolisaniji pristup radnji rezervisanoj za

superkorisnike, koristeći funkcije definisanja bezbednosti. Isti trik se može koristiti za prenošenje delimičnih privilegija između različitih korisnika.

Pre svega, potreban je pristup bazi podataka kao superkorisnik da bismo delegirali ovlašćenja. Pretpostavićemo da koristimo podrazumevanog superkorisnika po imenu postgres.
Sada ćemo videti dva načina kako da običnom korisniku omogućimo neke funkcionalnosti koje su dostupne samo super korisnicima.

Običan korisnik ne može reći PostgreSQL-u da kopira podatke iz fajla u tabelu baze podataka. Samo superkorisnik ili korisnik sa privilegijom pg_read_server_files može to da uradi:

```
pguser@hvost:~$ psql -U postgres
test2
...
test2=# create table lines(line text);
CREATE TABLE
test2=# copy lines from '/home/bob/names.txt';
COPY 37
test2=# GRANT ALL ON TABLE lines TO bob;
test2=# SET ROLE to bob;
SET
test2=> copy lines from '/home/bob/names.txt';
ERROR: must be superuser or have privileges of the pg_read_server_files
role to COPY from a file
HINT: Anyone can COPY to stdout or from stdin. psql's \copy command also
works for anyone.
```

Da bi dozvolio da bob kopira direktno iz datoteke, superkorisnik može dodeliti pg_read_server_files privilegije bobu na sledeći način:

```
test2=# GRANT pg_read_server_files TO bob;
```

Prilikom davanja prethodne dozvole, možda ćete želeti da proverite da li bob uvozi datoteke samo iz svog matičnog direktorijuma.

Dodeljivanje privilegija za pravljenje rezervnih kopija korisniku

Iz bezbednosnih razloga korisno je imati namensku ulogu ili korisnika namenjenog posebno za pravljenje rezervnih kopija baze podataka. PostgreSQL 16 nudi unapred definisane uloge i za logičke i za fizičke rezervne kopije, koje omogućavaju korisnicima baze podataka da obavljaju zadatke rezervnih kopija bez da budu superkorisnik.

Sledećom komandom moguće je napraviti „rezervnog“ korisnika:

```
CREATE USER backup_user;
```

Za logičke rezervne kopije dodeliti mu ulogu pg_read_all_data:

```
GRANT pg_read_all_data TO backup_user;
```

I atribut REPLICATION za pravljenje fizičkih rezervnih kopija:

```
ALTER USER backup_user WITH REPLICATION;
```

Uloga pg_read_all_data omogućava korisniku backup_user da pristupi svim podacima i definicijama tabela u bazi podataka. Takođe osigurava da ovaj korisnik može da čita sve nove tabele i podatke. REPLICATION atribut omogućava korisniku da pravi fizičke rezervne kopije koristeći komandu pg_basebackup.

Ako želite da dozvolite korisniku da verifikuje ili ponovo učita sva PostgreSQL podešavanja, možete dodeliti pg_read_all_settings i pg_signal_backend uloge.
Ako želite da dozvolite običnom korisniku da pokrene CHECKPOINT u PostgreSQL bazi podataka, treba dodeliti pg_checkpoint ulogu.
Za nadgledanje i ako je potrebno ukidanje povezanih korisnika u PostgreSQL-u, željenom korisniku mogu se dodeliti privilegije povezane sa pg_monitor i pg_signal_backend ulogama. To dozvoljava komande poput pg_terminate_backend.

3.2.3. Kreiranje novog korisnika

Postoje dva načina za kreiranje novog korisnika, jedan korišćenjem komandne linije i drugi korišćenjem SQL upita.

Da biste kreirali nove korisnike, morate ili biti super korisnik ili imati privilegiju CREATEROLE.

Iz komandne linije se može primeniti komanda createuser:

```
pguser@hvost:~$ createuser bob
```

Dodavanjem opcije --interactive može se aktivirati interaktivni režim. Interaktivni režim uključuje postavljanje pitanja kao što su sledeća:

```
pguser@hvost:~$ createuser --interactive alice
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
```

Bez postavljanja ovog moda, prethodna pitanja dobijaju ne kao podrazumevani odgovor; to je moguće promeniti pomoću opcija komandne linije: -s(--superuser), -d(--createdb) i -r(--createrole).
U interaktivnom režimu, pitanja će biti postavljena samo ako imaju smisla. Na primer, ako je korisnik super korisnik tada se ne postavljaju druga pitanja jer super korisnik ne podleže proverama privilegija. Drugi primer je kada se jedna od prethodnih opcija koristi za određivajne postavke koja nije podrazumevana, odgovarajuće pitanje neće biti postavljeno.

Iste korisnike možemo kreirati preko SQL upita na sledeći način:

```
CREATE USER bob;
CREATE USER alice CREATEDB;
```

U psql-u možemo proveriti atribute nekog korisnika na sledeći način:

```
pguser=# \du alice
```

Što daje izlaz:

```
List of roles
```

```
Role name | Attributes | Member of
-----------+------------+-----------
alice | Create DB | {}
```

Komande CREATE USER i CREATE GROUP su zapravo varijacije komande CREATE ROLE. CREATE USER username izraz je ekvivalentan CREATE ROLE username LOGIN i CREATE GROUP groupname izraz je ekvivalentan CREATE ROLE groupname NOLOGIN.

3.2.4. Uklanjanje korisnika bez brisanja njegovih podataka

Kada pokušavamo da uklonimo korisnika koji poseduje neke tabele ili druge objekte baze podataka, dobijamo sledeću grešku koja onemogućava da korisnik bude izbačen:

```
testdb=# drop user bob;
ERROR: role "bob" cannot be dropped because some objects depend on it
DETAIL: owner of table bobstable
owner of sequence bobstable_id_seq
```

Videćemo dva rešenja ovog problema.

Da biste izmenili korisnike, morate da budete super korisnik ili da imate privilegiju CREATEROLE.

Najlakše rešenje ovog problema je uzdržati se od izbacivanja korisnika i iskoristiti trik koji je opisan u odeljku Privremeno sprečavanje korisnika da se poveže, koji sprečava korisnika da se poveže:

```
pguser=# alter user bob nologin;
ALTER ROLE
```

Dodatna prednost je što u ovom slučaju, prvobitni vlasnik tabele može biti dostupan kasnije ako je potrebno, za potrebe revizije ili otklanjanja grešaka.

Zatim je moguće dodeliti prava izbrisanog korisnika novom korisniku, na sledeći način:

```
pguser=# GRANT bob TO bobs_replacement;
GRANT
```

Korisnik je implementiran kao uloga sa postavljenim atributom za prijavu. U ovom rešenju mi taj atribut uklanjamo korisniku i on se zatim čuva samo kao uloga.

Ukoliko je zaista potrebno da se korisnik ukloni, prvo moramo da dodelimo svo vlasništvo drugom korisniku. To je moguće sledećim utpitom, koji predstavlja PostgreSQL ekstenziju za standardni SQL:

```
REASSIGN OWNED BY bob TO bobs_replacement;
```

Radi tačno ono što kaže – dodeljuje vlasništvo nad svim objektima baze podataka koje trenutno poseduje uloga bob ulozi bobs_replacement.

Međutim, morate da imate privilegije i na staroj i na novoj ulozi da biste to uradili, i to morate da uradite u svim bazama podataka u kojima bob poseduje bilo koje objekte, pošto komanda REASSIGN OWNED radi samo na trenutnoj bazi podataka. Nakon ovoga možete izbrisati

korisnika bob.

3.3. Kontrola pristupa

Kontrola pristupa bazi podataka definiše ko i na koji način može pristupati podacima unutar baze podataka i od izuzetnog je značaja za sigurnost same baze podataka. Ovo uključuje različite dozvole i ograničenja korisnika i zavisi od privilegija koje imaju sami korisnici.

Kada se objekat kreira, dodeljuje mu se vlasnik. Vlasnik je obično uloga koja je izvršila naredbu kreiranja. Za većinu vrsta objekata, početno stanje je da samo vlasmik (ili superkorisnik) može da uradi bilo šta sa objektom. Da biste dozvolili drugim ulogama da ga koriste privilegije moraju biti dodeljene. Postoje različite vrste privilegija: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY EXECUTE, USAGE, SET and ALTER SYSTEM.

Objekat se može dodeliti novom vlasniku komandom ALTER odgovarajuće vrste za objekat, na primer:

```
ALTER TABLE table_name OWNER TO new_owner;
```

Dok superkorisnici to uvek mogu da urade. Obične uloge ovo mogu da urade samo ako su i trenutni vlasnik objekta(ili naslede privilegije vlasničke uloge) i ako imaju mogućnost da postave ulogu (SET ROLE) novoj ulozi vlasnika.

Za dodeljivanje privilegija koristi sekomanda GRANT. Na primer, ako je Joe postojeća uloga, a accounts postojeća tabela, privilegija za ažuriranje tabele može se dodeliti sa:

```
GRANT UPDATE ON accounts TO joe;
```

Specijalna uloga PUBLIC se može koristiti za dodelu privilegija svakoj ulozi u sistemu. Takođe „grupne“ uloge se mogu podesiti radi lakšeg upravljanja privilegijama kada postoji mnogo korisnika baze podataka.

Da bismo opozvali prethodno datu privilegiju, potrebna je odgovarajuća REVOKE naredba:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

U nastavku ćemo videti nekoliko zanimljivih primera kontrole prava pristupa korisnika: Dozvola pristupa nekog korisnika tabeli, Dozvola pristupa korisnika određenim kolonama, Dozvola pristupa korisnika određenim redovima i Privremeno sprečavanje korisnika da se poveže.

3.3.1. Dozvola pristupa nekog korisnika tabeli

Da bi mogao da vrši bilo koje akcije nad tabelom u bazi podataka, korisnik prvo treba da ima pristup toj tabeli.

Uverite se da imate definisane odgovarajuće uloge i da su privilegije PUBLIC uloge opozvane:

```
CREATE GROUP webreaders;
CREATE USER tim;
CREATE USER bob;
REVOKE ALL ON SCHEMA someschema FROM PUBLIC;
```

Morali smo da odobrimo pristup šemi da bismo dozvolili pristup tabeli. Pristup datoj šemi se može

koristiti kao brz i efikasan način za sprečavanje bilo kakvog pristupa bilo kom objektu u toj šemi. U suprotnom, ako želite da dozvolite određeni pristup, morate da u skladu sa tim koristite komande GRANT i REVOKE, po potrebi:

```
GRANT USAGE ON SCHEMA someschema TO webreaders;
```

Često je poželjno dati grupi sličnih korisnika dozvolu za pristup grupi objekata baze podataka. Da biste to uradili prvo ćete dodeliti sve dozvole proxy ulozi(poznatoj i kao grupa dozvola), a zatim dodeliti grupu izabranim korisnicima, na sledeći način:

```
GRANT SELECT ON someschema.pages TO webreaders;
GRANT INSERT ON someschema.viewlog TO webreaders;
GRANT webreaders TO tim, bob;
```

Sada i tim i bob imaju privilegiju SELECT nad tabelom pages i INSERT nad tabelom viewlog. Takođe možete dodati privilegije grupnoj ulozi nakon što je dodelite korisnicima.
Razmotrite sledeću komandu:

```
GRANT INSERT, UPDATE, DELETE ON someschema.comments TO webreaders;
```

Nakon pokretanja ove komande, i bob i ti imaju sve gore pomenute privilegije u tabeli comments.
Na osnovu toga pretpostavljamo da su i bob i tim uloge kreirane sa podrazumevanom postavkom INHERIT. U suprotnom, oni ne nasleđuju automatski prava uloga, već moraju eksplicitno da postave svoju ulogu dodeljenom korisniku, koristeći privilegije dodeljene toj ulozi.

Možemo dodeliti privilegije svim objektima određene vrste u specifičnoj šemi, na sledeći način:

```
GRANT SELECT ON ALL TABLES IN SCHEMA someschema TO bob;
```

I dalje morate da dodelite privilegije na samoj šemi u posebnoj GRANT dozvoli.

Prethodni niz komandi, prvo daje pristup šemi za grupnu ulogu, zatim daje odgovarajuća prava pregledea (SELECT) i modifikacije (INSERT) na određenim tabelama ulozi, i konačno, dodeljuje članstvo u toj ulozi za dva korisnika baze podataka.

PostgreSQL ne zahteva da imate neke privilegije da biste imali druge. To znači da možda imate tabele samo za pisanje u koje vam je dozvoljena modifikacija ali ne izbor(selekcija). Ovo se može koristiti za implementaciju funkcionalnosti slične redu pošte, gde nekoliko korisnika postavlja poruke jednom korisniku, ali ne može da vidi šta su drugi korisnici objavili.

Alternativno, možete postaviti situaciju u kojoj možete napisati zapis, ali ga ne možete promeniti ili izbrisati. Ovo je korisno za reviziju tabela tipa dnevnika, gde se sve promene beleže, ali se ne mogu menjati.

3.3.2. Dozvola pristupa korisnika određenim kolonama

Korisniku se može dati pristup samo nekim kolonama tabele.
Pretpostavljamo da već postoji šema koja se zove someschema i uloga koja se zove somerole, sa privilegijama USAGE na njoj. Kreiramo novu tabelu na kojoj ćemo dodeliti privilegije na nivou kolone:

```
CREATE TABLE someschema.sometable2(col1 int, col2 text);
```

Želimo da somerole korisniku dodelimo mogućnost da pregleda postojeće podatke i ubaci nove

podatke. Takođe želimo da obezbedimo mogućnost izmene postojećih podataka, ograničenih samo na kolonu col2.
Koristimo sledeće komande:

```
GRANT SELECT, INSERT ON someschema.sometable2
TO somerole;
GRANT UPDATE (col2) ON someschema.sometable2
TO somerole;
```

Zatim, možemo na sledeći način da testiramo da li je ovo uspešno funkcionisalo:

1. Pretpostavimo identitet uloge somerole i testiramo ove pivilegije pomoću sledećih komandi:

   ```
   SET ROLE TO somerole;
   INSERT INTO someschema.sometable2 VALUES (1, 'One');
   SELECT * FROM someschema.sometable2 WHERE col1 = 1;
   ```

2. Kao što je očekivano, možemo da ubacimo novi red i pogledamo njegov sadržaj. Hajde sada da proverimo sposobnost da ažuriramo nove kolone. Počinjemo sa drugom kolonom, koju smo ovlastili:

   ```
   UPDATE someschema.sometable2 SET col2 = 'The number one';
   ```

   Što vraća izlaz:

   ```
   UPDATE 1
   ```

3. To znači da smo uspešno ažurorali kolonu. Sada pokušajmo da ažuriramo prvu kolonu:

   ```
   UPDATE someschema.sometable2 SET col1 = 2;
   ```

   Što će vratiti:

   ```
   ERROR: permission denied for relation sometable2
   ```

   Ovo potvrđuje da smo uspeli da odobrimo ažuriranje samo druge kolone.

Komanda GRANT je proširena s namerom da omogući specifikaciju liste kolona, što znači da se privilegija daje toj listi kolona, a ne celoj tabeli.

Razmotrimo sada tabelu t, sa kolonama c1, c2 i c3. Postoje dva različita načina da se korisnik (u) ovlasti da izvrši sledeći upit:

```
SELECT * FROM t;
```

Prvi način je dozvoljavanjem privilegije na nivou tabele:

```
GRANT SELECT ON TABLE t TO u;
```

Alternativni način je dozvoljavanjem privilegije na nivou kolone:

```
GRANT SELECT (c1,c2,c3) ON TABLE t TO u;
```

Uprkos tome što ove dve tabele imaju efekte preklapanja, privilegije na nivou tabele se razlikuju od privilegija na nivou kolone, što je tačno pošto je značenje svake od njih drugačije.
Dodeljivanje privilegija nad tabelom znači davanje privilegija svim sadašnjim i budućim kolonama, dok privilegije na nivou kolone zahtevaju eksplicitnu naznaku kolona i stoga se ne proširuju automatski na nove kolone.
Način na koji privilegije funkcionišu u PostgreSQL-u znači da će datoj ulozi biti dozvoljeno da izvrši datu radnju ako odgovara jednoj od njenih privilegija. Ovo stvara izvesnu nejasnoću u oblastima koje se preklapaju. Razmotrimo sledeću sekvencu komandi:

```
GRANT SELECT ON someschema.sometable2 TO somerole;
REVOKE SELECT (col1) ON someschema.sometable2 FROM
somerole;
```

Ishod će nekako biti da je somerole-u dozvoljeno da vidi sve kolone te tabele, koristeći privilegiju na nivou tabele koju daje prva komanda. Druga komanda je bila neefikasna jer je pokušala da opozove privilegiju na nivou kolone (SELECT on col1) to nikada nije bilo odobreno.

3.3.3. Dozvola pristupa korisnika određenim redovima

PostgreSQL podržava davanje privilegija podskupu redova u tabeli pomoću RLS[5]-a.
Pretpostavićemo da već postoji šema koja se zove someschema i uloga koja se zove somerole sa privilegijama USAGE na njoj.

Kreiramo novu tabelu za eksperimentisanje sa privilegijama na nivou reda:

```
CREATE TABLE someschema.sometable3(col1 int, col2 text);
```

RLS takođe mora biti omogućen na toj tabeli:

```
ALTER TABLE someschema.sometable3 ENABLE ROW LEVEL SECURITY;
```

Prvo, dajemo privilegiju somerole-u da pregleda sadržaj tabele:

```
GRANT SELECT ON someschema.sometable3 TO somerole;
```

Pretpostavimo da je sadržaj tabele prikazan sledećom komandom:

```
SELECT * FROM someschema.sometable3;
col1 | col2
------+-----------
1 | One
-1 | Minus one
(2 rows)
```

Da bismo dali mogućnost pristupa samo nekim redovima, kreiramo politiku koja definiše šta je dozvoljeno i na kojim redovima. Na primer, na ovaj način možemo da primenimo uslov da je somerole-u dozvoljeno da bira samo redove sa pozitivnim vrednostima col1:

```
CREATE POLICY example1 ON someschema.sometable3
FOR SELECT
TO somerole
```

[5] RSL – Row Level Security

```
USING (col1 > 0);
```

Efekat ove komande je da se redovi koji ne zadovoljavaju politiku tiho preskaču, kao što je prikazano u sledećem nizu komandi:

```
SELECT * FROM someschema.sometable3;
col1 | col2
------+-----------
1 | One
(1 row)
```

Šta ukoliko želimo da uvedemo politiku nad klauzulom INSERT? Videli smo da USING klauzula određuje koji redovi su pogođeni. Takođe, postoji klauzula WITH CHECK koja se može koristiti da odredi koji umetnuti delovi su prihvaćeni. Uopštenije, klauzula USING se primenjuje na prethodno postojeće redove, dok se WITH CHECK primenjuje na redove koji su generisani naredbom koja se analizira. Dakle, prva radi sa SELECT, UPDATE i DELETE, druga sa INSERT i UPDATE.

Vraćajući se na naš primer, želimo da dozvolimo umetanje tamo gde je col1 pozitivan:

```
CREATE POLICY example2 ON someschema.sometable3
FOR INSERT
TO somerole
WITH CHECK (col1 > 0);
```

Takođe, ne smemo zaboraviti da dozvolimoo INSERT komande na tabeli, kao što smo ranije radili za SELECT:

```
GRANT INSERT ON someschema.sometable3 TO somerole;
SELECT * FROM someschema.sometable3;
col1 | col2
------+-----------
1 | One
(1 row)
```

Sada možemo da ubacimo novi red i zatim ga vidimo:

```
INSERT INTO someschema.sometable3 VALUES (2, 'Two');
SELECT * FROM someschema.sometable3;
col1 | col2
------+-----------
1 | One
2 | Two
(2 rows)
```

RLS politike se kreiraju i prmenjuju na datu tabelu pomoću CREATE POLICY sintakse. Sama RLS politika takođe mora biti eksplicitno omogućena na datoj tabeli jer je podrazumevano onemogućena.
U prethodnom primeru, pored kreiranja RLS politike morali smo da dodelimo privilegije na tabeli ili kolonama. To je zbog toga što se RLS ne posmatra kao još jedna privilegija koja se dodaje drugoj, već kao dodatna provera. U tom smislu bolje je da je podrazumevano onemogućena, pošto moramo da kreiramo politike samo za tabele u kojima naša logika pristupa zavisi od sadržaja reda.

3.3.4. Privremeno sprečavanje korisnika da se poveže

Ponekad je potrebno privremeno opozvati korisnikova prava na povezivanje bez brisanja korisnika ili promene njegove lozinke. Sada ćemo videti kako je moguće to ostvariti.

Da biste modifikovali druge korisnike, morate ili biti super korisnik ili imati privilegiju CREATEROLE (u ovom slučaju samo uloge koje nisu super korisnici se mogu menjati).

Ovim koracima je moguće privremeno sprečiti i ponovo izdati mogućnost prijavljivanja korisnika:

1. Ovom komandom možete privremeno sprečiti korisnika da se prijavi:

```
pguser=# alter user bob nologin;
ALTER ROLE
```

2. A ovom možete dozvoliti korisniku da se ponovo prijavi:

```
pguser=# alter user bob login;
ALTER ROLE
```

Na ovaj način se postavlja fleg u sistemskom katalogu, koji govori PostgreSQL-u da ne dozvoli korisniku da se prijavi. Ne izbacuje korisnike koji su već povezani.

Ograničavanje broja konkurentnih konekcija korisnika

Isti rezultat se može postići postavljanjem ograničenja veze za tog korisnika na 0:

```
pguser=# alter user bob connection limit 0;
ALTER ROLE
```

Da biste omogućili 10 istovremenih veza za korisnika bob, potrebna je ova komanda:

```
pguser=# alter user bob connection limit 10;
ALTER ROLE
```

Da biste dozvolili neograničen broj veza za ovog korisnika, pokrenite sledeću komandu:

```
pguser=# alter user bob connection limit -1;
ALTER ROLE
```

Omogućavanje neograničenih konekcija može dozvoliti DoS napad iscrpljivanjem resursa veze. Takođe, sistem može pasti ili se degradirati zbog preopterećenja legitimnih korisnika. Da biste izbegli ove rizike, poželjno je ograničiti broj istovremenih sesija po korisniku.

Opozivanje pristupa korisnika bazi podataka

Korisnikova veza sa bazom podataka može se ograničiti opozivanjem privilegije baze podataka iz te uloge. Međutim, pre nego što opozovete privilegiju CONNECT pojedinačnog korisnika, prvo se uverite da je ista privilegija opozvana iz uloge PUBLIC. Podrazumevano uloga PUBLIC poseduje privilegiju CONNECT za sve baze podataka. Uklanjanje ove privilegije iz PUBLIC će je posledično opozvati svim korisnicima, potencijalno ih sprečiti da se povežu osim ako im je

eksplicitno dodeljena privilegija CONNECT. Preporučljivo je opozvati sve privilegije sa PUBLIC nakon kreiranja baze podataka.

Koraci:

1. Proverite da li baza podataka daje privilegiju CONNECT za PUBLIC:

```
SELECT has_database_privilege('public', pguser, 'CONNECT');
```

2. Ako prethodna komanda vrati tačno, opozovite privilegiju CONNECT od PUBLIC:

```
REVOKE ALL ON DATABASE pguser FROM PUBLIC;
```

3. Sada dajte privilegiju CONNECT određenim ulogama kojima želite da dozvolite vezu sa bazom podataka:

```
GRANT CONNECT ON DATABASE pguser TO alice;
```

4. Opozovite privilegiju CONNECT korisnicima koje privremeno želite da blokirate iz baze podataka:

```
REVOKE CONNECT ON DATABASE pguser FROM bob;
```

Prisiljavanje NOLOGIN korisnika da prekine vezu

Da biste bili sigurni da su svi korisnici čije su privilegije za prijavu opozvane odmah diskonektovani, pokrenite sledeću SQL naredbu kao super korisnik:

```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity a
JOIN pg_roles r ON a.usename = r.rolname AND NOT rolcanlogin;
```

Ova naredba prekida vezu svim korisnicima kojima više nije dozvoljeno da se povežu ukidanjem bekenda koje su ovi korisnici otvorili.

4. Zaključak

U zaključku ovog rada možemo istaći sigurnost baza podataka kao nezaobilazan deo koji se tiče svake baze podataka i bez koje ne može postojati jedan funkcionalan sistem. Različite baze podataka koriste različite mehanizme zaštite za ostvarenje sigurnosti. Kod PostgreSQL-a smo kroz tri ključna poglavlja: Autentifikacija, Korisničke uloge i Kontrola pristupa videli teorijske osnove i primere mehanizama koji se u njemu koriste. Svi ovi delovi se prepliću i međusobno dopunjuju kako bi obezbedili sveobuhvatnu sigurnost u PostgreSQL bazi podataka. Autentifikacija je prvi korak ka kontroli pristupa, a korisničke uloge definišu prava koja se zatim primenjuju kroz kontrolu pristupa.

5. Literatura

[1] Ciolli G. , Mejias B. , Angelakos J. , Kumar V. , Riggs S. (2023). *PostgreSQL 16 Administration Cookbook: Solve real-world Database Administration challenges with 180+ practical recipes and best practices.* Packt Publishing Ltd.

[2] *Database Security: An Essential Guide*, dostupno na: https://www.ibm.com/topics/database-security (pristupljeno 25. aprila 2024.)

[3] *Zvanična Postgres dokumentacija – PostgreSQL 16.3 Documentation*, dostupno na https://www.postgresql.org/docs/current/ (pristupljeno 30. aprila 2024.)

[4] Stoimenov L. , *Siurnost kod baza podataka SUBP 2015/16*, Katedra za računarstvo, EFN

[5] *Chapter 21. Client Authentication*, dostupno na https://www.postgresql.org/docs/current/client-authentication.html (pristupljeno 9. maja 2024.)

[6] *Chapter 22. Database Roles*, dostupno na https://www.postgresql.org/docs/current/user-manag.html (pristupljeno 13. maja 2024.)

[7] *5.7. Privileges*, dostupno na https://www.postgresql.org/docs/current/ddl-priv.html (pristupljeno 23. maja 2024.)