Univerzitet u Nišu
Elektronski fakultet
Katedra za računarstvo

Seminarski rad

Sigurnost kod Mongo baze podataka

Mentor:
Doc. dr. Aleksandar Stanimirović

Student:
Emilija Bićanin 1474

Niš, 2023

Sadržaj

1. Uvod

Sigurnost je od najveće važnosti u bazama podataka. Baze podataka sadrže osetljive i dragocene informacije, kao što su lični podaci, finansijski zapisi, intelektualna svojina i poverljive poslovne informacije. Zaštita ovih podataka od neovlašćenog pristupa, modifikacije ili otkrivanja je ključna za osiguranje privatnosti, održavanje poverenja, poštovanje propisa i ublažavanje potencijalnih rizika. Evo nekoliko ključnih razloga zašto je bezbednost u bazama podataka važna:

- **Poverljivost podataka**: Baze podataka često čuvaju osetljive informacije, kao što su lične informacije, finansijski detalji, poslovne tajne ili poverljivi podaci. Adekvatne mere bezbednosti, kao što su kontrola pristupa i šifrovanje, sprečavaju neovlašćene osobe da pristupe i pregledaju ove podatke.
- **Integritet podataka:** Održavanje integriteta podataka je od vitalnog značaja da bi se osigurala tačnost i pouzdanost informacija uskladištenih u bazama podataka. Bezbednosni mehanizmi, kao što su pravila provere valjanosti podataka, kontrolne sume i kontrole pristupa, pomažu u sprečavanju neovlašćenih modifikacija, brisanja ili oštećenja podataka.
- **Usklađenost sa propisima**: Mnoge industrije imaju posebne propise i zahteve saglanosti u pogledu bezbednosti i privatnosti podataka. Organizacije treba da primene snažne mere bezbednosti da bi se pridržavale ovih propisa (npr. Opšta uredba o zaštiti podataka [GDPR-General Data Protection Regulation], Zakon o prenosivosti i odgovornosti zdravstvenog osiguranja [HIPAA- Health Insurance Portability and Accountability Act]). Nepoštovanje ovih mera može dovesti do pravnih posledica, finansijskih kazni i štete po ugled.
- **Sprečavanje neovlašćenog pristupa**: Neovlašćeni pristup bazama podataka može dovesti do curenja podataka, krađe identiteta, prevare ili drugih zlonamernih aktivnosti. Primena jakih mehanizama autentifikacije, kontrole pristupa i metoda kriptovanja značajno smanjuje rizik od neovlašćenog pristupa bazama podataka.
- **Kontinuitet poslovanja**: Baze podataka su kritične za poslovanje, a svaki bezbednosni incident ili kršenje podataka može poremetiti usluge, prouzrokovati finansijske gubitke i oštetiti reputaciju organizacije. Primena bezbednosnih mera, kao što su redovne rezervne kopije, planovi oporavka od katastrofe i kontrole pristupa, pomaže da se obezbedi kontinuitet poslovanja i minimizira uticaj potencijalnih bezbednosnih incidenata.
- **Poverenje i reputacija**: Kupci, partneri i zainteresovane strane polažu poverenje u organizacije da bi zaštitile svoje informacije. Demonstriranje posvećenosti bezbednosti baze podataka pomaže u izgradnji i održavanju poverenja, štiti reputaciju organizacije i može dati prednost u konkurenciji na tržištu.
- **Unutrašnje pretnje**: Bezbednost baze podataka se ne odnosi samo na spoljašnje pretnje; unutrašnje pretnje takođe mogu predstavljati značajne rizike. Organizacije treba da implementiraju odgovarajuće kontrole pristupa, korisničke privilegije i sisteme za praćenje kako bi otkrile i ublažile potencijalne zlonamerne aktivnosti ovlašćenih korisnika.
- **Dostupnost podataka**: Mere bezbednosti baze podataka takođe uključuju obezbeđivanje dostupnosti podataka kada je to potrebno. Zaštita od kvarova sistema, prirodnih katastrofa ili sajber napada pomaže u sprečavanju gubitka podataka i osigurava da kritične informacije budu dostupne ovlašćenim korisnicima.

Ukratko, bezbednost u bazama podataka je ključna za zaštitu osetljivih podataka, održavanje usklađenosti sa propisima, sprečavanje neovlašćenog pristupa i modifikacija, obezbeđivanje kontinuiteta poslovanja, izgradnju poverenja i očuvanje reputacije organizacije. Implementacija

robusnih bezbednosnih mera je od suštinskog značaja u današnjem digitalnom okruženju kako bi se ublažili potencijalni rizici i osigurala poverljivost, integritet i dostupnost podataka.

MongoDB nudi veliki broj bezbedonosnih mera poput: utvrđivanja identiteta korisnika (autentifikacija), kontrole pristupa (autorizacija), različitih mehanizama za kriptovanje podataka, praćenja aktivnosti baze i događaja u bazi,...Ovaj seminarski rad se bavi svim merama ponaosob. U prvom delu seminarskog biće opisani svi ovi mehanizmi koje MongoDB nudi i način na koji su implementirani, dok će se drugi deo baviti demonstracijom ovih mehanizama na praktičnim primerima.

2. Mere sigurnosti kod MongoDB baze podataka

2.1 Bezbednosna arhitektura kod MongoDB-ja

Holistička bezbednosna arhitektura mora da pokriva sledeće:

- **Upravljanje pristupom korisnika** radi ograničavanja pristupa osetljivim podacima, implementiran kroz kontrole autentifikacije i autorizacije,
- **Reviziju akcija baze podataka** za forenzičku analizu,
- **Zaštitu podataka putem enkripcije** podataka u pokretu preko mreže, u upotrebi u bazi podataka i u stanju mirovanja u trajnom skladištu,
- **Zaštitu okruženja** kako bi se obezbedio siguran rad domaćina sistema, mreža i drugih infrastruktura baza podataka.

Slika 1. Mere zaštite kod MongoDB-ja

U nastavku će biti dato objašnjenje svake od ovih mera.

Autentifikacija je proces koji podrazumeva utvrđivanje identiteta entiteta koji pristupa podacima. Kod baza podataka ovi entiteti mogu biti:

- Korisnici koji pristupaju bazi kao deo svoje svakodnevne poslovne rutine
- Administratori i developer-i
- Softverski sistemi uključujući aplikacione servere, alate za izveštavanje, sisteme za upravljanje bazom i sistemi za oporavak baze
- Fizički i logički čvorovi koji omogućavaju rad baze podataka. Baze podataka mogu biti distribuirane nad većim brojem čvorova radi skaliranja baze kao i da bi se obezbedio rad baze i u slučaju otkazivanje nekog dela sistema.

Najbolje prakse za autentifikaciju su sledeće:

- Kreiranje bezbedonosnih akreditiva-Treba kreirati zasebne akreditive (korisničko ime i lozinku) za svakog korisnika baze i treba izbegavati zajedničke akreditive pomoću kojih bazi pristupa veći broj korisnika, jer je tako nemoguće utvrditi ko je izvršio koju akciju nad bazom.
- Uspostavljanje zahteva koje lozinka mora da poštuje - Lozinke treba da budu u skladu sa zahtevima minimalne složenosti koje je ustanovila organizacija i trebalo bi da se povremeno resetuju. Lozinke sa niskom entropijom mogu se lako razbiti, čak i ako su enkriptovane. Lozinke visoke entropije mogu biti ugrožene nakon određenog vremena.
- Centralizovano upravljanje korisničkim pristupom
- Interna autentifikacija-Podrazume autentifikaciju čvorova koji pripadaju setovima replika ili *sharding* klasterima. Ovim se sprečava da se maliciozne instance pridruže klasteru baze podataka i onemogućava se nedozvoljeno kopiranje podataka i premeštanje podataka na nesigurne lokacije.

Autorizacija-Nakon procesa autentifikacije, sledi proces autorizacije kojim se utvrđuje koje su akcije dozvoljene određenom entitetu u bazi podataka. Kod procesa autorizacije definišu se **uloge**-uloga definiše skup privilegija nad bazom podataka. Korisnicima se u skladu sa njihovim potrebama dodeljuju različite uloge. U nastavku su date smernice koje MongoDB preporučuje prillikom konfiguracije procesa autorizacije:

- Omogućavanje minimalnog pristupa entitetima-Entitetima treba omogućiti samo pristup onim podacima koji su im neophodni da bi obavili svoju funkciju i ništa više. Ovim se sprečavaju kako maliciozni, tako i slučajni pristupi i izmene podataka.
- Grupisanje čestih skupova privilegija u uloge-Enitete treba grupisati po ulogama-npr. “Developer”, “Sysadmin”, “App server”... Korišćenje uloga pomaže da se pojednostavi upravljanje kontrolom pristupa definisanjem jedinstvenog skupa pravila koja se primenjuju na određene klase entiteta, umesto da se moraju definisati pojedinačna pravila za svakog korisnika.
- Kontrolisanje radnji koje entitet može da izvrši: Prilikom odobravanja pristupa bazi podataka, treba uzeti u obzir za koje specifične akcije ili komande svaki entitet trebaima dozvolu da pokrene. Na primer, aplikaciji će možda biti potrebne dozvole za čitanje/pisanje u bazi podataka, dok alatka za izveštavanje može biti ograničena samo na dozvolu za čitanje. Nekim korisnicima mogu biti dodeljene privilegije koje im omogućavaju da ubace nove podatke u bazu podataka, ali ne i da ažuriraju ili brišu postojeće podatke. Treba voditi računa da se obezbedi samo minimalni skup privilegija. Akreditivi najprivilegovanijih naloga bi mogli da ugroze celu bazu podataka ako budu hakovani interno ili od strane spoljnog uljeza.
- Kontrolisanje pristupa osetljivim podacima: Da bi se sprečilo pojavljivanje silosa podataka, trebalo bi da bude moguće ograničiti dozvole pristupa na pojedinačna polja, na osnovu bezbednosnih privilegija. Na primer, neka polja zapisa mogu biti dostupna svim korisnicima baze podataka, dok druga koja sadrže osetljive informacije, kao što je PII (P*ersonal Identifiable Information*), treba da budu ograničena na korisnike sa posebnom sigurnosnom proverom.

Revizioniranje-Revizioniranjem podataka je moguće pratiti promene u bazi podataka i entitete koji su izvršili te promene, čime se mogu otkriti i pokušaji pristupa neautorizovanim podacima.

Revizija treba da prati promene konfiguracije baze podataka kao i promene samih podataka u bazi. Kod procesa revizije, prilikom svake promene kreira se "log" koji treba da sadrži: akciju promene, entitet koji je načinio promenu i vremensku markicu (*timestamp*) kada je promena načinjena.

Prilikom praćenja promena kod samih podataka, treba pratiti operacije upita kao i operacije upisivanja podataka u bazu. Međutim, ukoliko se u bazi upisuje velik broj podataka u jedinici vremena, kreiranje logova će degradirati performanse baze. Zato treba naći kompromis između detaljnog praćenja promena i performansi baze. Dobro rešenje može predstavljati uvođenje filtera, tako da se logovi kreiraju samo onda kada entitet sa nekom specifičnog IP adresom izvrši promenu ili samo za neku specifičnu operaciju promene.

Enkripcija predstavlja proces šifriranja osetljivih podataka. Ovim se podaci štite od neautorizovane upotrebe. Podaci mogu biti šifrirani na mestu gde se čuvaju (at rest), prilikom njihovog transporta ili prilikom njihovog korišćenja. Tako se podaci štite od malicioznih napada na serveru, mreži, fajl sistemu ili bazi. U nastavku su date smernice za enkripciju od MongoDB-ja:

- **Enkripcija konekcija**-Pristupi svih korisnika i aplikacija bazi podataka treba da se odvijaju preko enkriptovanih kanala. Interna komunikacija između čvorova baze takođe treba biti kriptovana.
- **Enkripcija podataka u mirovanju**-Jedna od najčešćih pretnji bezbednosti baze podataka jesu napadi koji zaobilaze samu bazu i napadaju operativni sistem i fizičko skladište servera kako bi pristupili sirovim podacima. Za smanjivanje ovakvih rizika potrebno je izvršiti enkripciju podataka na samom disku gde se čuvaju.
- **Enkripcija podataka u korišćenju-** Kada se zahteva najviši nivo zaštite podataka, treba iskoristiti šifrovanje na nivou polja (FLE-*Field Level Encryption*) na strani klijenta. FLE vrši šifrovanje i dešifrovanje na klijentu, obezbeđujući da podaci nikada ne budu izloženi dok su na serveru. Sa FLE, čak ni kompromitovani administratorski nalog sa najvećim privilegijama ne može da dešifruje podatke.
- **Potpisivanje i rotacija ključeva za enkripciju**-Ključeve za enkripciju mreže i diska treba periodično rotirati, odnosno menjati.
- **Nametanje jake enkripcije-** Baza podataka treba da podržava FIPS (Federal Information Processing Standard) 140-2 kako bi se osigurala implementacija najbezbednijih algoritama šifrovanja.

Kontrola procesa i okruženja-Okruženje u kome radi baza podataka i njena osnovna infrastruktura treba da bude zaštićeno fizičkim i logičkim kontrolama. One se primenjuju u "deployment" okruženju, a ne u samoj bazi podataka, i uključuju:

° Instalaciju zaštitnih zidova (firewalls)

° Mrežne konfiguracije

° Definisanje dozvola kod fajl sistema (file system permissions)

° Kreiranje fizičkih kontrola pristupa IT okruženju

Uz sveobuhvatne kontrole za upravljanje pravima korisnika, reviziju i šifrovanje, zajedno sa najboljim praksama u zaštiti životne sredine, MongoDB može da ispuni zahteve najbolje prakse opisane ranije.

2.2 MongoDB Autentifikacija

Autentifikacija je proces potvrđivanja identiteta entiteta koji se povezuje sa MongoDB bazom. MongoDB podržava brojne mehanizme autentifikacije uključujući:

° **SCRAM** (podrazumevano)

° **k.509** proveru autentičnosti sertifikata

° **LDAP proxy** autentikaciju

° **Kerberos** autentifikaciju

Ovi mehanizmi omogućavaju MongoDB da se integriše u postojeći sistem autentifikacije i ispuni zahteve različitih okruženja.

2.2.1 SCRAM autentifikacija

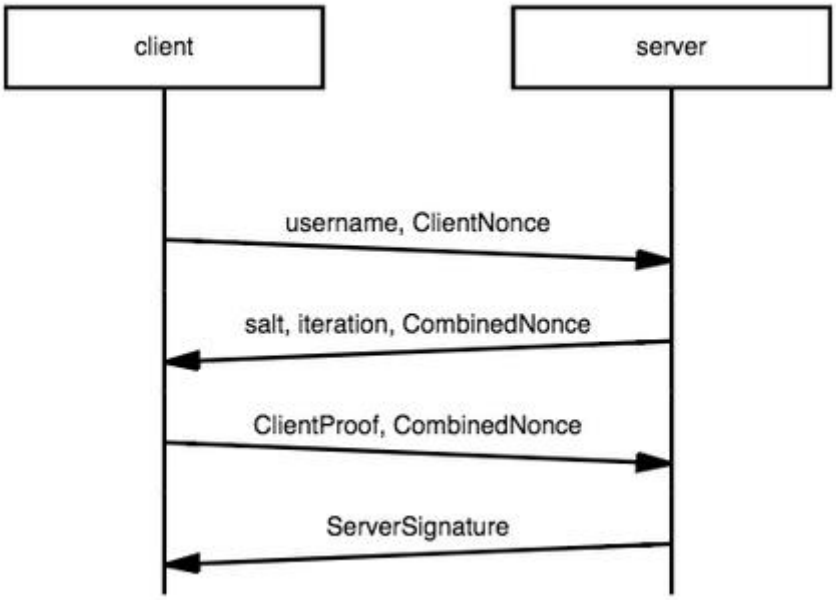
SCRAM-*Salted Challenge Response Authentication Mechanism*-predstavlja podrazumevani mehanizam autentifikacije kod MongoDB-ja zasnovan na lozinkama. Koristeći tehnike kriptografskog heširanja, klijent može da dokaže serveru da korisnik zna tajnu izvedenu iz korisničke lozinke bez slanja same lozinke. Server može da dokaže klijentu da zna tajnu izvedenu iz korisničke lozinke takođe bez potrebe da pošalje stvarnu lozinku. Takođe, na samom serveru se ne čuva lozinka direktno, već rezultat višestruke primene hash funkcije na lozinku i "salt"-slučajni random broj koji server generiše za svaku lozinku. SCRAM mehanizam štiti bazu od sledećih napada:

1. **Prisluškivanje** – U ovom slučaju napadač može da pročita sav saobraćaj koji se razmenjuje između klijenta i servera. Da bi se zaštitio od prisluškivanja, SCRAM klijent nikada ne šalje svoju lozinku kao otvoreni tekst preko mreže.
2. **Replay** – Napadač može ponovo poslati validne odgovore klijenta serveru. Međutim, svaka sesija SCRAM autentifikacije koristi nasumične jednokratne brojeve, tako da su poruke protokola važeće samo za jednu sesiju.
3. **Kompromitacija baze podataka** – U ovom slučaju napadač može da vidi sadržaj trajne memorije servera. Kompromitovanje baze podataka je ublaženo "začinjavanjem" i iterativnim heširanjem lozinki pre njihovog skladištenja.
4. **Zlonamerni server** – Kod ove vrste napada, napadač može da se predstavlja kao server klijentu. Međutim, kod SCRAM autentifikacije, napadač nije u mogućnosti da se predstavlja kao server bez znanja o klijentovim SCRAM akreditivima.

Prilikom kreiranja novog korisnika, server dobija od korisnika korisničko ime i lozinku. Server zatim generiše "salt"-neku random vrednost koju kombinuje sa lozinkom za računanje hash funkcije. Takođe generiše i broj iteracija *i* od kojeg zavisi koliko će se puta izvršiti heširanje lozinke. Što je broj iteracija *i* veći, to je razbijanje lozinke teže. Server kod sebe čuva heširanu lozinku (*Stored Key*), *salt*

vrednost i vrednost *i*. Server takođe čuva i kriptovanu verziju originalne lozinke (*Server Key*) kako bi mogao da portvrdi klijentu da je jednom znao njegovu lozinku u izvornom obliku i tako potvrdi svoj identitet klijentu.

Na slici 2 prikazan je proces razmene informacija prilikom sesije autentifikacije kod verzije SCRAM protokola koju koristi MongoDB. Najpre klijent šalje serveru zahtev za autentifikacijom koji sadrži korisničko ime i random broj (*Client Nonce*) koji se koristi za sprečavanje *replay* napada. Server odgovara porukom koja sadrži *salt* vrednost za tog korisnika, broj iteracija *i* i *Combine Nonce* koji predstavlja *Client Nonce* konkateniran sa random brojem koji generiše server-*Server Nonce (Combine Nonce= Client Nonce+ Server Nonce).* Zatim korisnik na osnovu podataka koje primio od servera računa *Client Proof* primenjujući iste korake koje je sproveo server prilikom računanja *Stored Key.* Takođe, klijent računa i *Client Signature* na osnovu *Combine Nonce* i izračunatog *Stored Key.* Zatim klijent šalje *Client Proof* i *Client Signature* serveru. Server na osnovu ovih vrednosti može utvrditi da li je klijent onaj za koga se predstavlja. Nakon što je klijent dokazao svoj identitet, i server mora dokazati svoj. Server šalje svoj *Server Signature* koji računa na osnovu *Server Key* i prethodno razmenjenih random brojeva. Nakon toga i klijent može da utvrdi da li je server validan.



Slika 2. Razmena informacija kod SCRAM-a

2.2.2 X.509 autentifikacija sertifikata

Proces autentifikacije može koristiti i certifikate umesto lozinki kao kod SCRAM-a. MongoDB za autentifikaciju sertifikatima koristi x.609 sertifikate. X.509 sertifikati obezbeđuju standardizovan i široko prihvaćen okvir za uspostavljanje poverenja i obezbeđivanje bezbedne komunikacije u različitim umreženim okruženjima.

U nastavku je dato nekoliko ključnih činjenica o x.509 sertifikatima:

1. Autoritet za izdavanje sertifikata (CA-Certificate Authority): x.509 sertifikate izdaje poverljivi entitet trećeg lica koji se zove *Autoritet za izdavanje sertifikata*. CA su odgovorni za proveru

identiteta pojedinaca ili organizacija koji traže sertifikate i garantuju za autentičnost informacija sadržanih u sertifikatu.

2. Digitalni potpisi: x.509 sertifikati koriste digitalne potpise kako bi osigurali integritet i autentičnost sadržaja sertifikata. CA potpisuje sertifikat pomoću svog privatnog ključa, a potpis se može proveriti korišćenjem javnog ključa CA.
3. Polja sertifikata: x.509 sertifikati sadrže različita polja koja pružaju informacije o sertifikatu i entitetu koji on predstavlja. Neka uobičajena polja uključuju:
   - Subjekat: Identifikuje entitet (kao što je organizacija ili pojedinac) kome se izdaje sertifikat.
   - Izdavač: identifikuje CA koji je izdao sertifikat.
   - Javni ključ: Sadrži javni ključ koji odgovara privatnom ključu koji drži entitet.
   - Validnost: Određuje period tokom kojeg se sertifikat smatra važećim.
   - Ekstenzije: Dodatne informacije ili atributi povezani sa sertifikatom, kao što su upotreba ključa, alternativna imena subjekta i informacije o opozivu sertifikata.
4. Lanci sertifikata: x.509 sertifikati mogu formirati lance poverenja. Lanac sertifikata se sastoji od više sertifikata gde je svaki sledeći sertifikat u lancu izdavanja CA verifikovan u prethodnom sertifikatu. Lanac se obično završava sa *root* sertifikatom, koji je samopotpisan i predstavlja krajnju tačku poverenja.
5. Scenariji upotrebe: x.509 sertifikati imaju široku primenu u bezbednim komunikacionim protokolima kao što su bezbednost transportnog sloja (TLS/SSL) za obezbeđenje web saobraćaja, bezbedne/višenamenske ekstenzije za internet poštu (S/MIME) za šifrovanje i potpisivanje e-pošte i virtuelne privatne mreže (VPN-ovi) za siguran daljinski pristup.
6. Opoziv sertifikata: x.509 sertifikati se mogu opozvati pre isteka ako su ugroženi ili više nisu važeći. Liste opoziva sertifikata (CRL) ili Online Certificate Status Protocol (OCSP) mogu se koristiti za proveru statusa opoziva sertifikata.

MongoDB podržava x.509 autentifikaciju sertifikata za autentifikaciju klijenta i internu autentifikaciju članova skupova replika i deljenih klastera.

2.2.3 LDAP Proxy autentifikacija

LDAP (engl. *Lightveight Directori Access Protocol*) proxy autentifikacija je mehanizam koji omogućava LDAP serveru da delegira zahteve za autentifikaciju drugom LDAP serveru, poznatom kao proksi server. Ovo podešavanje je korisno u scenarijima gde postoji više LDAP servera i poželjan je centralizovani mehanizam autentifikacije.

Kada klijent inicira zahtev za autentifikaciju proksi serveru, proksi server prosleđuje zahtev odgovarajućem LDAP serveru za autentifikaciju. Proksi server deluje kao posrednik između klijenta i LDAP servera, rukujući zahtevima za autentifikacijom.

Evo opšteg pregleda kako funkcioniše LDAP proxy autentikacija:

1. Klijent pokreće zahtev za autentifikaciju: Klijent (kao što je aplikacija ili korisnik) šalje zahtev za autentifikaciju proksi serveru.
2. Proksi server prima zahtev: Proksi server prima zahtev za autentifikaciju od klijenta. Proksi server pronalazi odgovarajući LDAP server: Na osnovu konfiguracije ili drugih kriterijuma, proksi server određuje koji LDAP server treba da obradi zahtev za autentifikaciju.

3. Proksi server prosleđuje zahtev: Proksi server prosleđuje zahtev za autentifikaciju određenom LDAP serveru.
4. LDAP server vrši autentifikaciju: LDAP server prima zahtev od proksi servera i obavlja proces autentifikacije, uključujući proveru akreditiva u bazi podataka servera direktorijuma.
5. Odgovor na autentifikaciju: LDAP server šalje odgovor za potvrdu autentičnosti nazad proksi serveru.
6. Proksi server prosleđuje odgovor: Proksi server prima odgovor od LDAP servera i šalje ga nazad klijentu.
7. Klijent prima rezultat autentifikacije: Klijent prima rezultat autentifikacije od proksi servera, koji pokazuje da li je autentifikacija bila uspešna ili ne.

LDAP proksi autentifikacija pruža nekoliko prednosti, uključujući:

- Centralizovanu autentifikaciju: Sa LDAP proksi autentifikacijom, zahtevi za autentifikaciju od klijenata mogu se centralno upravljati i delegirati na više LDAP servera po potrebi.
- Balansiranje opterećenja i prevazilaženje greške: Proksi server može da distribuira zahteve za autentifikaciju na više LDAP servera, balansirajući opterećenje i obezbeđujući redundantnost u slučaju kvarova na serveru.
- Bezbednost i kontrola pristupa: Proksi server može da primeni bezbednosne politike i kontrole pristupa, obezbeđujući da samo ovlašćenim klijentima bude dozvoljeno da se autentifikuju.
- Pojednostavljena konfiguracija klijenta: Klijenti samo treba da budu konfigurisani za povezivanje sa proksi serverom, pojednostavljujući proces konfigurisanja i smanjujući potrebu za modifikacijama na strani klijenta.

LDAP proksi autentifikacija se obično koristi u velikim sistemina gde je potreban centralizovani mehanizam autentifikacije, kao što su okruženja sa više servera direktorijuma ili složene mrežne arhitekture.

2.2.4 Kerberos autentifikacija

Kerberos predstavlja standardizovan protokol autentifikacije namenjen velikim, distibuiranim klijent/server sistemima koji omogućava i klijentu i serveru da međusobno potvrde svoje identitete. Kerberos koristi kriptografiju simetričnog ključa i zahteva ovlašćenje od pouzdanog trećeg lica za verifikaciju identiteta korisnika.

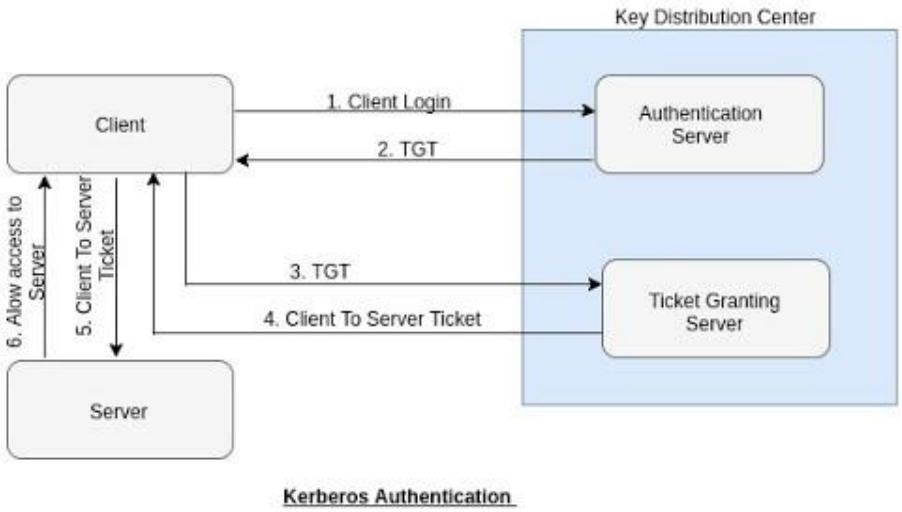
Učesnici u Kerberos protokolu jesu: klijent, server i KDC (engl. Key distribution center) koji predstavlja treće lice od poverenja i pruža usluge autentifikacije i izdavanja “tiketa”. Tiketi sadrže informacije koje omogućavaju verifikaciju identiteta između čvorova.

Svaki korak Kerberos autentifikacije koristi kriptografiju da zaštiti pakete od promene ili čitanja i obezbedi međusobnu autentifikaciju. Klijent traži kartu za korisnika od KDC-a, koristeći korisničku lozinku za šifrovanje zahteva. Ako KDC može da dešifruje zahtev sa korisničkom lozinkom koju je sačuvao, zna da je klijent dao ispravnu lozinku za korisnika. KDC kreira kartu za dodelu tiketa (TGT) za korisnika, šifruje je korisničkom lozinkom i vraća je klijentu. Ako klijent može da dešifruje tu kartu sa korisničkom lozinkom, zna da je KDC legitiman.

Klijent zahteva kartu za uslugu od KDC-a tako što predstavlja svoj TGT i zahtev za uslugu dodeljivanja tiketa (TGS) koji uključuje ime usluge servera za uslugu kojoj želi da pristupi. KDC kreira tiket za uslugu (TGS) koji je šifrovan hešom lozinke usluge (TGS tajni ključ), šifruje tiket i poruku autentifikatora pomoću deljenog ključa sesije za dodelu tiketa i na kraju šalje TGS nazad klijentu.

Klijent zahteva pristup usluzi servera predstavljanjem tiketa usluge koji je dobio od KDC serveru aplikacija, koji dešifruje poruku koristeći sopstveni heš lozinke. Ako uspešno dešifruje TGS, server aplikacija odobrava pristup klijentu.

Na slici 3 prikazan je proces razmene poruka kod Kerberos protokola.



Slika 3. Kerberos autentifikacija

2.3 Autorizacija kod MongoDB-ja

Autorizacija je proces određivanja specifičnih dozvola koje entitet ima u bazi podataka. MongoDB koristi kontrolu pristupa zasnovanu na ulogama (*Role Based Access Control*) za upravljanje pristupom MongoDB sistemu. Korisniku se dodeljuje jedna ili više uloga koje određuju korisnikov pristup resursima i operacijama baze podataka. Izvan dodeljenih uloga, korisnik nema pristup sistemu.

MongoDB obezbeđuje unapred definisane uloge koje podržavaju uobičajene privilegije korisnika i administratora baze podataka kao što su: *dbAdmin, dbOwner, clusterAdmin, readWrite* i mnoge druge.

Ove uloge mogu biti dodatno prilagođene putem korisnički definisanih uloga, omogućavajući administratorima da dodeljuju specifičnije privilegije klijentima, na osnovu njihovih odgovarajućih potreba za pristupom podacima i obradom. Da bi se pojednostavilo obezbeđivanje i održavanje naloga, uloge se mogu delegirati između timova, obezbeđujući primenu doslednih politika u određenim funkcijama obrade podataka unutar organizacije. MongoDB pruža mogućnost specificiranja korisničkih privilegija sa granularnošću na nivou baze podataka i kolekcije.

2.3.1 Uloge

Uloga pruža korisniku privilegije da obavi određeni niz operacija nad resursima. Privilegija pripada ulozi A ukoliko se eksplicitno navede prilikom kreiranja uloge A ili ukoliko pripada ulozi B iz koje je izvedena uloga A. U nastavku je dato nekoliko ključnih osobina uloga kod MongoDB-ja:

1. Kombinovanje-Jedan korisnik može posedovati više uloga. Uloge ne ograničavaju privilegije. Ukoliko korisnik ima na primer dve uloge, uloga sa manje privilegija neće ograničiti mogućnosti uloge sa više privilegija.
2. Restrikcije autentifikacije-Uloge mogu nametnuti neke restrikcije korisnicima, kao na primer logovanje samo sa specifičnih opsega IP adresa.
3. Privilegije-Uloga se definiše privilegijama, a one predstavljaju skup dozvoljenih akcija nad resursom. Resurs može biti baza, kolekcija, skup kolekcija ili klaster.
4. Nasleđivanje privilegija-Uloga se može definisati pomoću već postojećih uloga i tada se kaže da ta uloga nasleđuje postojeće uloge. Samim tim uloga nasleđuje i sve njihove privilegije.
5. Uloge i korisnici-Korisnicima se mogu dodeliti uloge prilikom kreiranja korisnika, ali se one mogu ažurirati kasnije, bilo dodavanjem novih uloga ili uklanjenjem postojećih.
6. Ugrađene i korisnički definisane uloge-MongoDB pruža skup predefinisanih uloga koje se često koriste u bazama podataka. Međutim, ukoliko nijedna od ovih uloga ne sadrži željeni skup privilegija, mogu se definisati custom uloge.
7. Definisanje ograničenja-za svaku ulogu se može ograničiti nad kojim kolekcijama važe privilegije kojim je ona definisana.

MongoDB koristi kombinaciju imena baze podataka i imena uloge da jedinstveno definiše ulogu. Svaka uloga je ograničena na bazu podataka u kojoj se kreira, ali MongoDB čuva sve informacije o ulogama u kolekciji admin.system.roles u admin bazi podataka.

2.32 Ugrađene uloge

MongoDB nudi veliki broj predefinisanih uloga koje su česte u bazama podataka. U nastavku su navedene najznačajnije.

Korisničke uloge-Svaka baza podataka uključuje sledeće klijentske uloge:

- read-Pruža mogućnost čitanja podataka nad svim nesistemskim kolekcijama,
- readWrite-Uključuje sve privilegije read uloge, i dodatno omogućava modifikaciju podatka, kreiranje indeksa, kolekcija, brisanje podataka, kolona, kolekcija,…

Administratorske uloge

U ugrađene administratorske uloge spadaju:

- dbAdmin-omogućava obavljanje administrativnih funkcija nad bazom poput: upravljanja šemom podataka, indeksiranje, prikupljanje statističkih podataka,
- userAdmin-uključuje mogućnost kreiranja korisnika i uloga, kao i njihovu modifikaciju,
- dbOwner-predstavlja vlasnika baze podataka i uključuje sve privilegije readWrite, dbAdmin i userAdmin uloga.

Administratorske uloge na klasteru

U ugrađene administratorske uloge na klasteru spadaju:

- clusterManager-obuhvata privilegije koje omogućavaju upravljanje i nadzor klastera,
- clusterMonitor-obuhvata samo privilegije čitanja kako bi se omogućio nadzor klastera,
- hostManager-omogućava upravljanje serverima koji čine klaster,
- clusterAdmin-obuhvata sve privilegije tri prethodno navedene uloge.

Uloge za oporavljanje baze

- backup-omogućava minimalne privilegije za kreiranje backup-a podataka,
- restore-privilegije koje omogućavaju oporavak baze.

Za korisnike kreirane u *admin* bazi postoje posebne uloge, koje im omogućavaju privilegije nad svim nad svim bazama, sa izuzetkom baza *config* i l*ocal:*

- readAnyDatabase-omogućava privilegije čitanja nad svim bazama, sem local i config baza
- readWriteAnyDatabase-omogućava iste privilegije kao readWrite, ali nad svim bazama sem local i config
- userAdminAnyDatabase- omogućava iste privilegije kao userAdmin, ali nad svim bazama sem local i config
- dbAdminAnyDatabase- omogućava iste privilegije kao dbAdmin, ali nad svim bazama sem local i config

2.4 MongoDB Revizioniranje podataka (*Auditing*)

MongoDB Enterprise nudi mogućnost praćenja aktivnosti sistema uz pomoć auditing framework-a.

Kada se omogući, sistem za reviziju detektuje događaje poput: DDL operacija, operacija replikacije i klasterizacije, autentifikacije, autorizacije i CRUD operacije. Moguće je postaviti filtere tako da se ne prate svi mogući događaji, već samo oni od većeg interesa. Filteri se mogu odnositi na korisnike, operacije ili uloge. Na primer, moguće je pratiti samo akcije korisnika koji su pristupili određenom dokumentu, korisnika sa određenom IP adresom, sa određenom ulogom ili samo akcije određenih operacija-npr. Mogu se pratiti samo operacije upisivanja. Pošto revizija podataka usporava bazu, potrebno je razmisliti o filterima kako bi se to usporavanje smanjilo.

Logovi o događijima se mogu upisivati na više destinacija (npr. konzola, fajl,..) i u više formata (JSON, BSON).

Sistem za reviziju upisuje svaki događaj koji se prati u poseban bafer. Periodično se taj bafer prazni i njegov sadržaj se upisuje na disk. Za sve događaje koji pripadaju jednoj konekciji, MongoDB garantuje njihovo totalno uređenje: kada se upiše neki događaj na disk, sistem garantuje da su svi događaji koji su mu prethodili i pripadaju istoj konekciji već upisani na disk.

2.4 Enkripcija podataka

MongoDB omogućava administratorima da enkriptuju podatke prilikom njihovog transporta ali i u stanju mirovanja. Takođe omogućava enkripciju podataka korisnicima na nivou polja kako bi zaštitili svoje senzitivne podatke od administratora i ostalih korisnika. U nastavku će biti opisane vrste enkripcija koje MongoDB podržava.

2.4.1 Mrežna enkripcija

MongoDB podržava TSL/SSL (Transport Layer Security/Secure Sockets Layer) za enkripciju svih podataka tokom njihovog kretanja u mreži. TLS/SSL vodi računa o tome da podaci budu čitvljivi jedino klijentima.

TSL/SSL protokol podržava samo algoritme jake enkripcije, odnosno algoritme koji koriste veoma dug ključ. Minimalni zahtev za dužinu ključa je 128 bita. TLS/SSL koristi asimetrično kriptovanje koje se sastoji iz javnog i privatnog ključa. Javni ključ se koristi za enkripciju podataka, a privatni za dekripciju. Za utvrđivanje verodostojnosti ključeva koriste se sertifikati.

Za svaku sesiju kreira se **privremeni ključ**. Privremeni ključ je zaštićen tako što je kriptovan serverovim privatnim ključem. Međutim, čak i ako privremeni ključ bude kompromitovan, on kriptuje samo jednu sesiju tako da sve ostale sesije ostaju neugrožene.

MongoDB podržava *Forward Secrecy* šifrator koji koristi *Diffie-Hellman (DHE)* i *Ephemeral Elliptic Curve Diffie-Hellman (ECDHE)* algoritme.

Za upotrebu TSL/SSL protokola sa MongoDB-jem, neophodno je koristiti validne sertifikate u PEM fajl formatu.

Počev od verzije 4.4, MongoDB podržava OCSP (*Online Certificate Status Protocol*) koji služi da utvrdi da li je sertifikat validan ili je istekao. Ovim protokolom se eliminiše potreba periodičnog skidanja liste isteklih sertifikata (CRL-*Certificate Revocation List*) i restartovanja servera onda kada se uoče promene u listi.

2.4.2 Enkripcija podataka u mirovanju

Mrežna enkripcija omogućava bezbedan transport podataka, ali od velikog je značaja zaštiti podatke i onda kada su u stanju mirovanja u skladištu. Zato MongoDB nudi mogućnosti kriptovanja podataka u stanju mirovanja na disku. Ukoliko se enkripcija u mirovanju omogući, podrazumevani algoritam za enkripciju je AES256-CBC (*256-bit Advanced Encryption Standard in Cypher Block Chaining mode*). Na raspolagaju je i AES256 u GCM (*Galois/Counter mode*) režimu. Ovi algoritmi za enkripciju koriste simetrične ključeve, što znači da se isti ključ koristi i za enkripciju i za dekripciju podataka.

Proces enkripcije podataka u mirovanju uključuje:

- Generisanje master ključa
- Generisanje ključa za svaku od baza podataka
- Enkripciju podataka pomoću ključeva baza
- Enkripciju ključeva baza pomoću master ključa

Svi fajlovi sa podacima su potpuno kriptovani iz perspektive fajl sistema, podaci postoje u nekriptovanom stanju samo onda kada napuste disk u pređu u RAM memoriju servera i tokom transporta (ukoliko se ne korisiti mrežna enkripcija).

Ključevi za enkripciju/dekripciju baza su interni za server i čuvaju se u enkriptovanom obliku na disku servera. Master ključ je eksterni u odnosu na server i nikad se ne prebacuje na disk servera.

Master ključ zahteva eksterno upravljanje, van servera. Za upravljanje master ključem podržane su dve opcije:

- Integracija sa sistemom za upravljanje ključevima nekog trećeg lica pomoću *Key Manager Interoperability Protcol-a* (KMIP). Ovaj način za upravljanje master ključem se i preporučuje od strane MongoDB-ja.
- Lokalno upravljanje ključem koji se nalazi u lokalnom fajlu.

Bitno je napomenuti da enkripcija u mirovanju nije sastavni deo replikacije. Naime, svaki čvor u sistemu će imati svoj sopstveni master ključ i ključeve baza, oni se ne repliciraju! Iako je moguće koristiti iste ključeve za više čvorova, MongoDB to ne preporučuje, već je bezbednije da se podaci kriptuju individualno na svakom čvoru.

Vrlo je poželjno makar jednom godišnje vršiti **rotaciju ključeva**, odnosno menjati stare ključeve za nove periodično.

U Atlas klasteru svi podaci su podrazumevano kriptovani u mirovanju.

2.4.3 Enkripcija podataka u upotrebi (In-use Encryption)

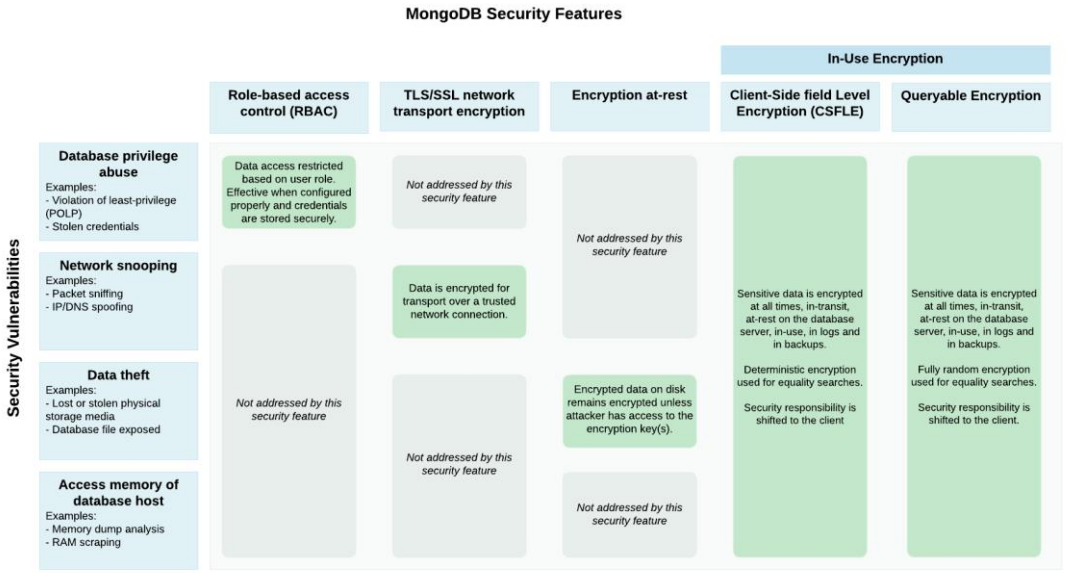
Ova vrsta enkripcije omogućava da senzitivni podaci budu dostupni u izvornom obliku samo krajnjem klijentu, a da se u mreži prilikom transporta i u skladištu nalaze samo kriptovane verzije podataka. Na ovaj način se postiže da senzitivni podaci budu isključivo u vlasništvu klijenta i da im čak ni administratori ne mogu pristupiti. Serveri nisu ni svesni enkripcije i procesom enkripcije upravljaju MongoDB drajveri na klijentskoj strani. Još jedna prednost ovakve enkripcije koja se izvršava na klijentu jesta ta što na serveru nemamo gubitke u performansama uzrokovane procesom enkripcije. Postoje dve vrste ovakve ekripcije koje MongoDB podržava: *Client-side field level encyption-CSFLE* i *Queryable Encryption*.

In-use enkripcija štiti od napada sledećih vrsta:

- Direktni pristup poljima od strane superkorisnika baze (nekog sa visokim ovlašćenjima poput admina baze),
- Čitanja RAM memorije servera,
- Prisluškivanja mreže,
- Čitanja podataka na disku.

Na sledećoj slici prikazane su različite metode sigurnosti i napade od kojih one štite.

Sa slike se vidi da jedino CSFLE enkripcija i *Queryable* enkripcija štite podatke od svih vrsta napada. Takođe, jedino ove vrste enkripcija štite korisničke podatke od zloupotreba administratora baze. Ipak, metode sigurnosti ne isključuju jedna drugu već pojačavaju sveukupnu bezbednost ukoliko se koriste zajedno, pa je poželjno uključiti što veći broj mera bezbednosti u sistem.



Slika 4. Poređenje mera sigurnosti kod MongoDB-ja

2.4.3.1 CSFLE (Client-side field-level encryption)

Ova vrsta enkripcije omogućava enkripciju na nivou polja u kolekciji, odnosno klijenti imaju mogućnost da definišu koja polja u kolekciji će biti enkriptovana i na koji način. Za svako polje ponaosob se može specificirati drugačiji ključ za enkripciju.

Koja polja će u dokumentima biti enkriptovana i na koji način definiše se **šemom enkripcije**. Šema enkripcije predstavlja JSON objekat koji se sastoji iz parova ključ-vrednost pri čemu ključ predstavlja naziv polja koje se enkriptuje a vrednost predstavlja objekat kojim se definišu enkripciona pravila. Jedno enkripciono pravilo treba da sadrži informacije o tome koji se algoritam za enkripciju koristi, koji ključ za enkripciju se koristi i BSON tip polja koje se enkriptuje. Na sledećoj slici prikazano je kako treba da izgleda enkripciono pravilo za određeno polje.

```
"<field-name-to-encrypt>": {
  "encrypt": {
    "algorithm": "<encryption algorithm to use>",
    "bsonType": "<bson type of field>",
    "keyId": [UUID("<_id of your Data Encryption Key>" )]
  }
}
```

Slika 5. Enkripciono pravilo

Pri navođenju algoritma za enkripciju treba specificirati da li je on deterministički ili randomiziran. Ukoliko je deterministički to će omogućiti izvršenje upita po enkriptovanom polju (jer deterministčka enkripcija uvek za isti ulaz daje isti izlaz). Ukoliko se ne planira izvršenje upita po enkriptovanom polju, algoritam može biti randomiziran. Randomzиran algoritam daje različite izlaze i za iste ulaze. Deterministička enkripcija je ranjiva ukoliko polje ima malu kardinalnost, odnosno mali broj mogućih vrednosti, pa je moguće lako dešifrirati vrednosti statističkom analizom.

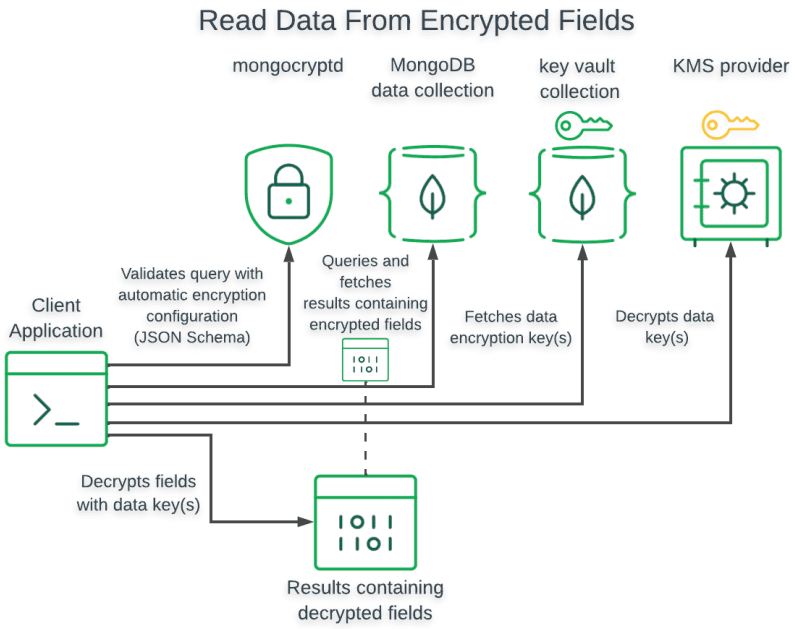
Ključevi koji se koriste pri CSFLE enkripciji su:

- *CMK (Customer Master Key)*-koristi se za enkripciju DEK ključeva. Ovo je najosetljiviji podatak u celom CSFLE sistemu za enkripciju jer se njegovim otkrićem mogu ugroziti svi podaci.
- *DEK (Data Encryption Key)* -koristi se za enkripciju polja u MongoDB dokumentima. DEK ključevi su enkriptovani pomoću CMK ključa i tako se čuvaju u *Key Vault* kolekciji mongo baze.

Kao što je pomenuto DEK ključevi se čuvaju u *key vault* kolekciji u mongo bazi. Ova kolekcija mora biti indeksirana po imenima ključa. Ime ključa sadrži informaciju o tome polju koje se enkriptuje. Za čuvanje CMK ključa preporučuju se KMS (Key Management System) provajderi poput: *Amazon Web Services KMS*-a, *Azure Key Vault*-a, *Google Cloud Platform KMS*-a,..

Na sledećoj slici prikazan je prikazuje tok izvršenja upita kod CSFLE onda kada upit uključuje enkriptovana polja. Po prijemu upita, MongoDB drajver koristi pravila šifrovanja po polju specificirana u okviru validacije JSON šeme da bi utvrdio da li su neka šifrovana polja uključena u upit. Zatim pribavlja enkriptovane podatke iz Mongo baze. Zatim se iz key vault kolekcije pribavljaju odgovarajući DEK ključevi za dekripciju polja. Pošto su oni enkriptovani CMK ključem, treba pribaviti ovaj ključ od KMS provajdera. Nakon što se pomoću master ključa dekriptuju DEK ključevi, moguće je njima na klijentskoj strani dekriptovati podatke i vratiti ih klijentu.

Pošto server nema pristup ključevima, neke upite poput pronalaženja opsega i sortiranja nije moguće izvršiti na serveru.



Slika 6. Tok upita kod CSFLE

2.4.3.2 Queryable Encryption

Princip po kojem radi Queryable enripcija je vrlo sličan CSFLE enkripciji. Glavna razlika je u tome što CSFLE enkripcija uglavnom koristi deterministčku enkripciju, dok queryable enkripcija koristi **struktuiranu enkripciju** tako da se omogućavaju kompleksniji upiti nad enkriptovanim poljima.

3. Praktična demonstracija mera sigurnosti kod MongoDB baze

3.1 Autentifikacija

3.1.1 Kreiranje administratora korisnika

Procesi autentifikacije i autorizacije podrazumevaju postojanje korisnika. Svi entiteti koji pristupaju bazi treba da budu predstavljeni korisničkim nalogom.

Da bi kreiranje korisnika bilo omogućeno, treba najpre kreirati poseban nalog sa ulogom korisničkog admina. Naredba kojom se kreira korisnički admin prikazana je na sledećoj slici:

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: passwordPrompt(), // or cleartext password
    roles: [
      { role: "userAdminAnyDatabase", db: "admin" },
      { role: "readWriteAnyDatabase", db: "admin" }
    ]
  }
)
```

Slika 7. Kreiranje admina za upravljanje korisnicima

Korisnički admin je kreiran u *admin* bazi. Admin, kao i ostali korisnici, kreira se *db.createUser* naredbom. Kao parametar ove naredbe prosleđuje se dokument sa sledećim poljima: *user* (korisničko ime korisnika), *pwd* (lozinka korisnika) i *roles* (niz uloga koje se dodeljuju korisniku). Uloge koje su dodeljene korisničkom administratoru su: *userAdminAnyDatabase* kako bi mogao da kreira različite korisnike u različitim bazama i *readWriteAnyDatabase* za pristup podacima u bazama. Prilikom specificiranja uloge, navodi se vrsta uloge (bilo da je ugrađena ili bilo korisnička) i baza podataka u kojoj ta uloga postoji.

3.1.2 Prijavljivanje korisnika

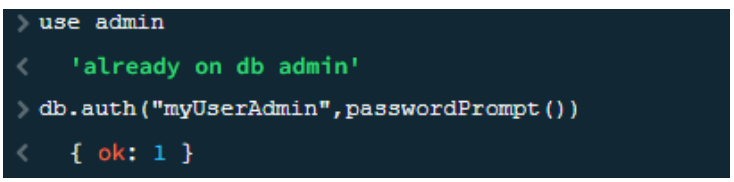
Nakon što je kreiran prvi korisnik, možemo ga iskoristiti za demonstraciju procesa autentifikacije. Podrazumevani algoritam za autentifikaciju kod mongoDB-ja je SCRAM. On koristi SHA-1 i SHA-256 heš funkcije za skladištenje lozinki. Na sledećoj slici prikazano je kako se korisnik kreiran prethodnom naredbom čuva u bazi podataka. On je kreiran u *admin* bazi, a db.createUser naredbom on je smeštem *system.users* kolekciji.

Slika 8. Čuvanje korisničkog naloga u bazi kod SCRAM autentifikacije

Kao što je prikazano na slici iznad, lozinka koju je korisnik uneo prilikom kreiranja naloga se ne čuva u bazi. Ono što se čuva jesu izlazi iz hash funkcija koje su primenjene više puta (*iterationCount*) nad lozinkom i *salt* vrednošću.

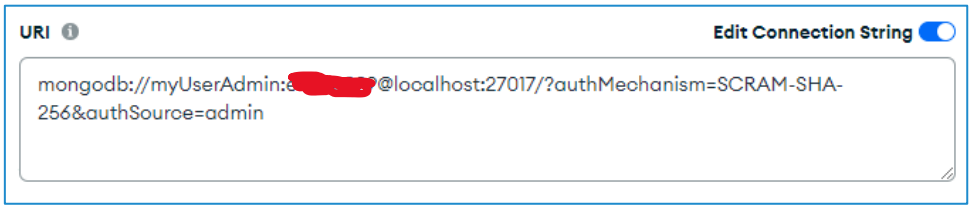
Da bi se prijavili u MongoDB bazu kao autentifikovan korisnik potrebno je uraditi sledeće:

1. Povezati se na MongoDB server (ovde je to lokalni server) korišćenjem URI stringa (npr. mongodb://localhost:27017)
2. Zatim je potrebno pozicionirati se u bazu u kojoj se čuvaju korisnici i izvršiti *db.Auth* naredbu. Ova naredba kao parametre uzima korisničko ime korisnika i njegovu lozinku. Ovde je umesto lozinke navedena naredba passwordPromt() koja sprečava da lozinka bude vidljiva pri unosu.



Slika 9. Autentifikacija nakon konekcije

Umesto naredbe *db.auth*, moguće je odmah izvršiti autentifikaciju prilikom kreiranja konekcije sa bazom korišćenjem sledećeg URI-ja:



Slika 10. Autentifikacija prilikom konekcije

Kao što se vidi sa slike, prvo se navodi korisničko ime, pa lozinka, zatim ime hosta i porta i na kraju se navodi mehanizam koji se koristi za autentifikaciju (u ovom slučaju SCRAM-SHA-256) i baza u kojoj se korisnik. Mana ovog pristupa je što se lozinka direktno navodi u konekcionom stringu.

3.2 Autorizacija

Kada je korisnički admin logovan, on ima privilegiju kreiranja korisnika. U nastavku će biti prikazani primeri kreiranja korisnika sa različitim privilegijama.

3.2.1 Kreiranje read-only korisnika

Sledećom naredbom kreiran je korisnik koji ima samo pristup čitanja i to samo nad bazom *medicalRecords.*

```
> db.createUser({user:"readOnlyuser",pwd:"ronly",roles:[{role:"read",db:"medicalRecords"}]})
<   { ok: 1 }
```

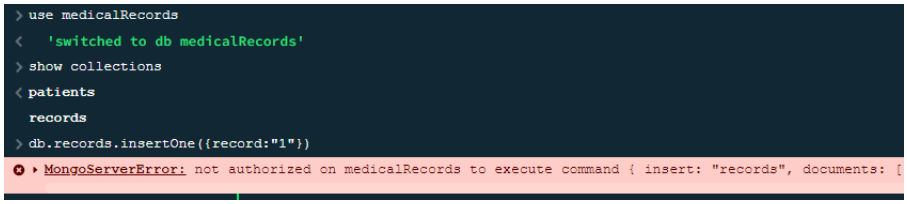
Slika 11. Kreiranje read-only korisnika

Ukoliko se sada prijavimo kao novokreirani korisnik koji ima samo privilegiju čitanja nad *medicalRecords* bazom i izvršimo naredbu *show databases*, jedina baza koja će biti prikazana je *medicalRecords.* Sve ostale baze su sakrivene od ovog korisnika.

```
>_MONGOSH
> show databases
< medicalRecords  80.00 KiB
Enterprise test>
```

Slika 12. Baze dostupne korisniku

Ukoliko ovaj korisnik pokuša da u *medicalRecords* bazi izvrši neku od operacija čitanja (kao npr. Show collections) to će mu biti dozvoljeno, međutim ukoliko na primer pokuša da izvrši operaciju upisa (kao npr. ubacivanje dokumenta u kolekciju), ta operacija će mu biti odbijena. Ovo je demonstrovano na sledećoj slici:



Slika 13. Pokušaj dozvoljene i nedozvoljene operacije

3.2.2 Kreiranje korisnika sa više uloga

Uloge je moguće međusobno kombinovati. Kada se korisnik definiše sa većim brojem uloga, on poseduje uniju privilegija definisanih navedenim ulogama. Na sledećoj slici prikazan je primer kreiranja korisnika sa više uloga.

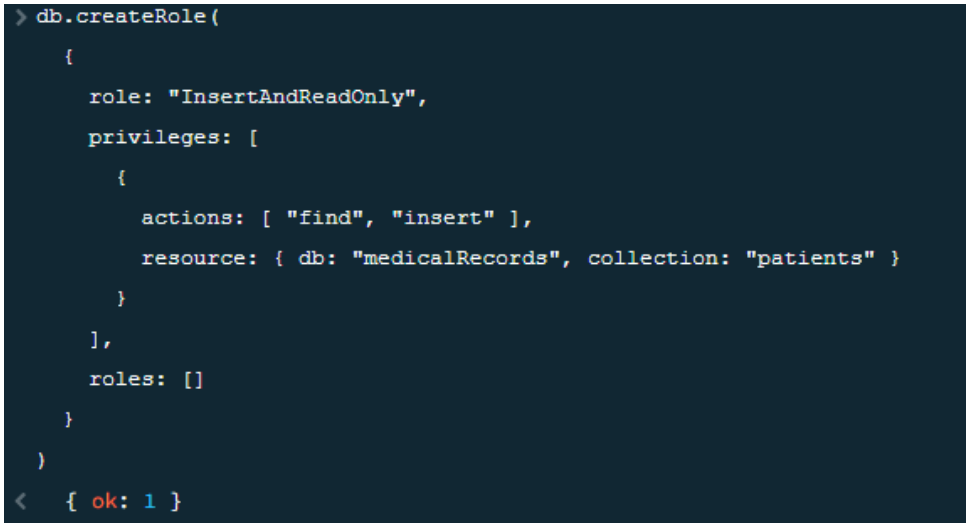
```
Enterprise reporting> db.createUser(
                      {
                        user: "reportsUser",
                        pwd: passwordPrompt(),  // or cleartext password
                        roles: [
                          { role: "read", db: "medicalRecords" },
                          { role: "readWrite", db: "reports" }
                        ]
                      }
                     )
```

Slika 14. Kreiranje korisnika sa većim brojem uloga

Na slici iznad prikazan je *reportsUser* korisnik koji, recimo, kreira izveštaje. Njemu je zato data privilegija čitanja nad bazom *medicalRecords* i privilegija čitanja i upisa nad bazom *reports*, kako bi na osnovu podataka iz baze *medicalRecords* mogao da piše izveštaje u bazi *reports*. Ovaj korisnik nije svestan postojanja nijedne druge baze sem ove dve i nad njima može da obavlja samo navedene operacije.

3.2.3 Kreiranje korisnički definisanih uloga

Svim prethodno kreiranim korisnicima bile su dodeljene predefinisane, ugrađene MongoDB uloge. Međutim, MongoDB ima opciju kreiranja custom korisnički definisanih uloga, koje omogućavaju korisnicima da sami definišu koje dozvoljene operacije podrazume određena uloga.Kreiranje uloge, kao i kreiranje korisnika, može da izvrši nalog korisničkog admina koji poseduje *userAdmin* ulogu, pa je pre kreiranja uloge potrebno prijavljivanje na njegov nalog. Na slici ispod prikazan je primer kreiranja korisnički definisane uloge:



Slika 15. Kreiranje korisnički definisane uloge

Korisnički definisana uloga kreira se naredbom *db.createRole* koja kao parametar uzima dokument sa sledećim poljima:

- *Role*-definiše naziv uloge, u ovom slučaju *InsertAndReadOnly,*
- *Privileges*-predstavlja listu dozvoljenih akcija koje podrazumeva ova uloga i resursa nad kojim su ove akcije dozvoljene,
- *Roles*-predstavlja listu uloga koje novokreirana uloga nasleđuje, mogu biti i ugrađene i korisnički definisane. Nasleđivanje podrazumeva da nova uloga preuzima sve privilegije uloge koju nasleđuje.

Uloga koja je kreirana na slici podrazumeva dozvoljene *find i insert* operacije nad *patients* kolekcijom *medicalRecords* baze.

Da bi uverili da je uloga zaista kreirana, možemo pozvati db.getRoles() metodu:

Slika 16. *getRoles* metoda

Ukoliko želimo da kreiramo korisnika sa ovom novom ulogom, to se radi na isti način kao u prethodnim primerima kada su se koristile ugrađene uloge.

3.3 Client-Side Filed Level Encyption (CSFLE)

U nastavku će biti opisana jednostavna C# aplikacija kreirana za potrebe demonstriranja CSFLE enkripcije kod MongoDB-ja.

3.3.1 Kreiranje master ključa

Najpre je potrebno kreirati 96bitni master ključ kojem pristup ima jedino klijent i koji će se koristiti za enkripciju ključeva koji se koriste za enkripciju polja dokumenata. Uloga svih ovih ključeva detaljno je objašnjena u prethodnom poglavlju. Master ključ se može kreirati u kodu, ali je ovde iskorišćen openssl protokol za kreiranje 96bitnog ključa u base64 formatu. Za kreiranje master ključa korišćena je sledeća naredba:

```
OpenSSL> rand -base64 -out C:/Users/user/Desktop/key.txt 96
```

Slika 17. Kreiranje master ključa

3.3.2 Kreiranje indeksa nad keyVault kolekcijom

Kako bi pronalazak ključeva prilikom enkripcije bio efikasan, treba kreirati indeks nad imenima ključeva korišćenim nad poljima (ime ključa specificirano je *keyAltNames* poljem). U nastavku je dat C# kod kojim se kreira ovakav indeks.

```
var connectionString = credentials["MONGODB_URI"];
var keyVaultNamespace = CollectionNamespace.FromFullName("encryption.__keyVault");
var keyVaultClient = new MongoClient(connectionString);
var indexOptions = new CreateIndexOptions<BsonDocument>();
indexOptions.Unique = true;
indexOptions.PartialFilterExpression = new BsonDocument { { "keyAltNames", new BsonDocument { { "$exists", new BsonBoolean(true) } } } };
var builder = Builders<BsonDocument>.IndexKeys;
var indexKeysDocument = builder.Ascending("keyAltNames");
var indexModel = new CreateIndexModel<BsonDocument>(indexKeysDocument, indexOptions);
var keyVaultDatabase = keyVaultClient.GetDatabase(keyVaultNamespace.DatabaseNamespace.ToString());
keyVaultDatabase.DropCollection(keyVaultNamespace.CollectionName);
keyVaultClient.GetDatabase("medicalRecords").DropCollection("patients");
var keyVaultCollection = keyVaultDatabase.GetCollection<BsonDocument>(keyVaultNamespace.CollectionName.ToString());
keyVaultCollection.Indexes.CreateOne(indexModel);
```

Slika 18. Kreiranje indeksa nad keyVault kolekcijom

3.3.3 Kreiranje ključeva za enkripciju polja

Sledeći korak predstavlja kreiranje ključeva kojim će osetljiva polja u dokumentu biti enkriptovana. Na slici ispod prikazan je C# kod kojim se jedan ovakav ključ kreira u base64 formatu. Prilikom kreiranja ovog ključa, potrebno je navesti ime kolekcije u kojoj će se čuvati (keyVaults kolekcija) kao i klijenta koji kreira ovaj ključ.

```
var clientEncryptionOptions = new ClientEncryptionOptions(
    keyVaultClient: keyVaultClient,
    keyVaultNamespace: keyVaultNamespace,
    kmsProviders: kmsProviders
    );

var clientEncryption = new ClientEncryption(clientEncryptionOptions);
var dataKeyOptions = new DataKeyOptions();
List<string> keyNames = new List<string>();
keyNames.Add("demo-data-key");
var dataKeyId = clientEncryption.CreateDataKey(provider, dataKeyOptions.With(keyNames), CancellationToken.None);
var dataKeyIdBase64 = Convert.ToBase64String(GuidConverter.ToBytes(dataKeyId, GuidRepresentation.Standard));
```

Slika 19. Kreiranja ključa za enkripciju polja

3.3.4 Kreiranje šeme enkripcije

Nakon što su ključevi uspešno kreirani, sledeći korak u ovom procesu jeste kreiranje šeme enkripcije kojom će biti definisana polja u dokumentu koja će biti kriptovana, kao i način njihove enkripcije i njihov tip. U ovoj aplikaciji ono što se kriptuje jesu osetljivi podaci o pacijentima koji se nalaze unutar *patients* kolekcije *medicalRecords* baze.

Na sledećoj slici prikazano je pravilo enkripcije definisano za bloodType polja. Sva pravila enripcije nalaze se unutar jedinstvenog JSON objekta i predstavljaju šemu enkripcije za pomenutu *medicalRecords* kolekciju.

```
        },
        { "bloodType", new BsonDocument {
            { "encrypt" ,new BsonDocument {
                    { "bsonType", "string" },
                    { "algorithm", "AEAD_AES_256_CBC_HMAC_SHA_512-Random"}
                    }
                }
            }
        },
```

Slika 20. Primer jednog enkripcionog pravila u aplikaciji

3.3.4 Kreiranja dokumenta

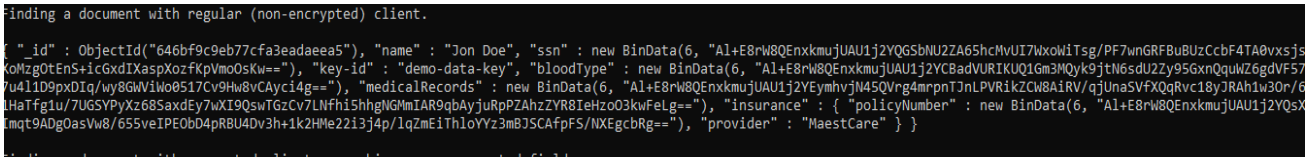
Nakon što su definisani i ključevi i šema enkripcije, može se početi sa dodavanjem dokumenta u kolekciju. Sva polja koja su navedena u šemi enkripcije biće kriptovana ključem za enkripciju na samoj klijentskoj strani i biće poslata na server u kriptovanom obliku. Na sledećoj slici dat je primer kreiranja dokumenta čija će polja biti kriptovana.

```
var sampleDocFields = new BsonDocument
{
    { "name", "Jon Doe" },
    { "ssn", 145014000 },
    { "key-id", "demo-data-key"},
    { "bloodType", "AB-" },
    {
        "medicalRecords", new BsonArray
        {
            new BsonDocument("weight", 180),
            new BsonDocument("bloodPressure", "120/80")
        }
    },
    {
        "insurance", new BsonDocument
        {
            { "policyNumber", 123142 },
            { "provider", "MaestCare" }
        }
    }
};

// Construct an auto-encrypting client
var secureCollection = secureClient.GetDatabase(db).GetCollection<BsonDocument>(coll);

// Insert a document into the collection
secureCollection.InsertOne(sampleDocFields);
```

Slika 21. Dodavanje dokumenta

3.3.5 Izvršenje upita nad dokumentom sa kriptovanim poljima

Kada neki drugi klijent, koji nije kreator ključeva za enkripciju polja, pokuša da pribavi dokument, polja koja su specificirana u enkripcionoj šemi biće mu nečitljiva. Na sledećoj slici prikazan je rezultat takvog upita.



Slika 22. Rezultati upita za klijenta koji ne poseduje ključeve

Sa druge strane, ukoliko je korisnik taj koji je kreirao dokument i ključeve, moći će da pribavi ključeve polja iz keyVaults kolekcije i dekriptuje ih svojim master kljućem. Na sledećoj slici prikazan je rezultat upita takvog korisnika. Njemu su sva polja potpuno čitvljiva.

Slika 23. Rezultati upita za klijenta koji poseduje ključeve

4. Zaključak

Kao što je u seminarskom prikazano, MongoDB nudi veliki broj mera za poboljšanje sigurnosti baze. Treba iskoristiti što veći broj ovih mera, pogotovo ukoliko radimo sa veoma senzitivnim podacima. MongoDB posebno preporučuje korišćenje enkripcije na klijentskoj strani jer ona štiti podatke klijenata čak i od zlonamernih administratora baze.

Iako je radi bezbednosti baze dobro koristiti mere bezbednosti u što većem broju, ove mere mogu usporiti performanse baze. Naime, procesi enkripcije i dekripcije, upisivanje logova, proveravanje identiteta i prava pristupa korisnika su sve procesi koji će usporiti operacije nad bazom. Zato treba pažljivo konfigurisati sve ove procese, tako da se nepotrebne radnje svedu na minimum.

5. Literatura

- MongoDB Encryption at Rest and Data Encryption Types
- Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained - Pt. 1 | MongoDB
- MongoDB Security Architecture | MongoDB
- Security — MongoDB Manual