Hvala doc.dr.sc. Stjepanu Grošu i asistentu Karlu Slovenecu na ukazanoj pomoći i danim smjernicama pri istraživanju i izradi završnog rada.

Sadržaj

1. Uvod

Svakog dana pojavljuju se nove prijetnje bazama podataka. Kritični podaci, pohranjeni na poslužiteljima diljem svijeta, predstavljaju privlačnu metu napadačima raznih profila. Postavlja se pitanje na koji način te podatke zaštititi, a da određeni klijentski zahtjevi ostanu osigurani. Najčešće rješenje je primjena kriptografskih algoritama kojima se napadačima onemogućava pristup podacima na disku. No koliko su šifrirani podaci zapravo sigurni? Koji se novi problemi javljaju uslijed korištenja šifriranja? Glavni motiv rada upravo su ta pitanja. Cilj rada jest proučiti kako funkcionira šifriranje u Postgresu, jednom od najpopularnijih sustava za upravljanje bazama podataka otvorenog koda [1]. Osim toga, cilj je i pokazati koliko informacija o pohranjenim podacima napadači mogu saznati u određenim slučajevima.

Sve češće se baze podataka smještaju u oblak [2]. Ako se kao metoda zaštite podataka koristi njihovo šifriranje, iznimno je važno znati koje su prednosti a koji nedostaci šifriranja podataka. Isto tako, nužno je ispravno definirati model prijetnje kako bi arhitekti baza podataka znali od kojih točno prijetnji se šifriranjem štite, a od kojih ne.

Rad je strukturiran na sljedeći način. Nakon uvodnog poglavlja, u drugom poglavlju uvode se osnovni pojmovi šifriranja, šifriranje se stavlja u kontekst baza podataka i njihove zaštite, predstavljaju se neki od načina na koji se efikasnije izvode upiti nad šifriranim podacima te je dan model prijetnje za šifriranja u bazama podataka. U trećem poglavlju naglasak je na bazi podataka Postgres. Objašnjena je arhitektura baze podataka Postgres, koje su dostupne metode šifriranja te je na primjeru pokazano kako se može implementirati šifriranje. U četvrtom poglavlju predstavljeni su napadači na bazi podataka i na komunikacijskom putu. Eksperimentom je pokazano koliko precizno napadači na komunikacijskom putu mogu odrediti koliko se redaka tablice nalazi u odgovoru na postavljene upite.

2. Općenito o šifriranju

Podaci su izloženi riziku neovisno o tome gdje se nalaze i zahtijevaju pripadnu razinu zaštite. Postoje razni načini kako se oni mogu zaštititi. U ovom radu naglasak će biti na jednoj od najkorištenijih metoda zaštite podataka, šifriranju [3].

Za početak u nastavku poglavlja uvodi se podjela podataka s obzirom na stanje. Zatim se uvode osnovni pojmovi šifriranja, a potom se šifriranje stavlja u kontekst baza podataka te se pokazuju neki od načina na koji se upiti nad šifriranim podacima mogu efikasnije izvoditi. Konačno, na kraju poglavlja objašnjeno je modeliranje prijetnji te je dan model prijetnje za šifriranja u bazama podataka.

2.1. Stanja u kojima se podaci nalaze

Da bi se u potpunosti shvatila motivacija za šifriranjem podataka, nužno je odrediti njihovu klasifikaciju. Jedna od često korištenih podjela podataka je podjela s obzirom na stanje.

Podaci, s obzirom na stanje, dijele se na *podatke u pokretu* (engl. *data in motion*) i *podatke u mirovanju* (engl. *data at rest*). Uz navedenu podjelu u nekoj literaturi [4] može se dodatno pronaći i treće stanje podataka, a to su *podaci koji se koriste* (engl. *data in use*).

Podaci u pokretu su podaci koji se kreću s jednog mjesta na drugo putem interneta ili lokalne mreže. Primjerice, podaci se mogu slati s oblaka u bazu podataka ili s lokalnog uređaja za pohranu na oblak. Učinkovite mjere za zaštitu takvih podataka od iznimne su važnosti jer se smatra da su podaci općenito ugroženiji dok su u pokretu [4].

Izraz podaci u mirovanju koristi se za podatke trenutno pohranjene na bilo kakvom uređaju ili mediju za pohranu. Napadačima su češće zanimljiviji ovi podaci iako su oni općenito sigurniji od onih u pokretu [4].

Kad se govori o podacima koji se koriste najčešće se misli na podatke koje trenutno obrađuje neka aplikacija. To mogu biti podaci koji su u postupku izrade, ažuriranja ili brisanja. Ovaj naziv uključuje i podatke koje korisnici trenutno pregledavaju.

Šifriranje je od velike važnosti kod zaštite svih prethodno navedenih stanja podataka. Kod zaštite podataka u pokretu, kompanije se obično odluče na šifriranje osjetljivih podataka prije nego se podaci pošalju na odredište i koriste šifrirane konekcije, primjerice TLS (engl. *Transport Layer Security*). Kod zaštite podataka u mirovanju, podaci se obično šifriraju prije pohrane u bazu podataka.

2.2. Šifriranje

U području kriptografije, šifriranje je postupak kodiranja poruke ili informacije na način da samo ovlaštene osobe imaju pristup informaciji. Trenutno, šifriranje je jedno od najpopularnijih metoda zaštite podataka [3][5].

U području računarstva, za nešifrirane podatke koristi se naziv *čisti tekst* (engl. *plain text*), a za šifrirane podatke naziv *šifrat* (engl. *cyphertext*). Formule koje se koriste za šifriranje i dešifriranje poruka nazivaju se *algoritmi za šifriranje*, ili skraćeno *šifre* (engl. *cypher*). Da bi bio učinkovit, algoritam za šifriranje za postupak šifriranja koristi varijablu, koja se naziva ključ. Ključ je informacija koja čini šifrat jedinstvenim.

Šifriranje je od velike važnosti za niz područja informacijskih tehnologija jer osigurava niz sigurnosnih zahtjeva. Osigurava povjerljivost, cjelovitost, autentičnost i neporecivost [6].

Najčešća podjela algoritama za šifriranje je na simetrične i asimetrične.

Simetrični algoritmi koriste isti ključ za šifriranje i dešifriranje. Simetrično šifriranje starija je i poznatija tehnika od asimetričnog. Snaga simetričnog algoritma najčešće se mjeri duljinom ključa. Za simetrični algoritam nužno je da se na temelju čistog teksta i šifrata ne može odrediti ključ [7].

Simetrične algoritme može se podijeliti u dva osnovna tipa. Prvi tip su blok orijentirani algoritmi (engl. *block cyphers*) i koriste ključ koji je fiksne duljine, najčešće 128 ili 256 bita. Drugi tip su bit/bajt orijentirani algoritmi (engl. *stream cyphers*), a za njih je karakteristično da je ključ iste veličine kao i poruka. Trenutno najkorišteniji simetrični algoritam za šifriranje, koji pripada blok orijentiranim algoritmima, je AES (engl. *Advanced Encryption Standard*).

Unatoč širokoj primjeni, simetrični algoritmi imaju određene probleme. Vjerojatno najveći problem je kako sigurno prenijeti ključ drugoj strani. Uz to, postoji i problem odabira ključa koji bi trebao biti slučajan kako ga napadač ne bi mogao pogoditi.

Kod asimetričnih algoritama za šifriranje koriste se različiti ključevi za šifriranje i dešifriranje zbog čega se ovi algoritmi i zovu asimetričnima. Svaki sudionik ima dva ključa, javni i privatni. Javni ključ koristi se za šifriranje, a privatni za dešifriranje [8].

Svaki sudionik objavljuje svoj ključ za šifriranje, koji se zbog toga naziva javnim. Pošiljatelji poruka koriste taj ključ kako bi šifrirali poruke. Primatelji poruka zatim koriste svoj privatni, tajni ključ kako dešifrirali poruke koje su šifrirane njihovim javnim ključem.

Najpopularniji asimetrični algoritam za šifriranje je RSA, koji se temelji na faktorizaciji velikih brojeva [9].

Prednosti asimetričnih algoritama su te što rješava problem distribucije ključa drugoj strani te činjenica da omogućava digitalne potpise, čime se može otkriti je li poruka izmijenjena u prijenosu. Najveći nedostatak ovih algoritama je to što su u usporedbi sa simetričnim algoritmima sporiji, pogotovo u slučaju šifriranja većih količina podataka.

U praksi se podaci šifriraju simetričnim algoritmima, a zatim se za razmjenu ključa koriste asimetrični algoritmi [10].

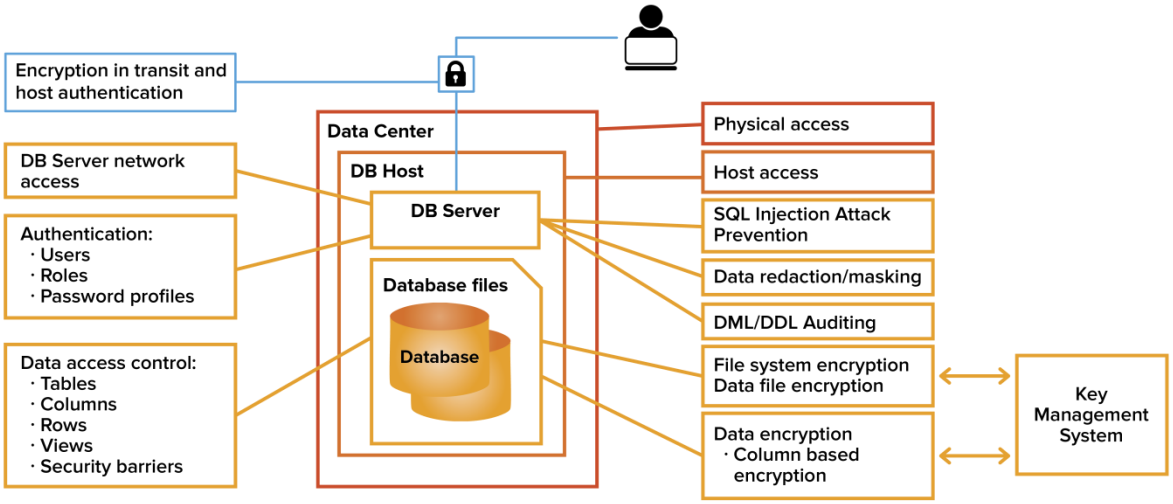
2.3. Šifriranje baza podataka

Izraz *šifriranje baza podataka* koristi se kao skupni naziv niza metoda zaštite baza podataka, koje mogu biti implementirane unutar i izvan samog sustava za upravljanje bazom podataka (engl. *database management system*).

Konceptualno, baza podataka je spremnik u koji se spremaju podaci. Koristeći tu analogiju, može se objasniti raspodjela metoda šifriranja baza podataka. Moguće je zaštititi cijeli spremnik što bi odgovaralo šifriranju na razini datoteka ili cijelog operacijskog sustava. Nadalje, moguće je zaštititi samo sadržaj spremnika što bi odgovaralo šifriranju cijele baze podataka. Konačno, moguće je zaštititi samo podskup sadržaja spremnika što bi odgovaralo šifriranju na razini atributa, tablice ili sheme [11].

Na slici 2.1. može se vidjeti spomenuta arhitektura. Svakom dijelu arhitekture na slici pridružena je komponenta gdje se može vidjeti na koji način se može zaštititi pojedini dio arhitekture.

U nastavku poglavlja opisane su dvije metode šifriranja podataka, transparentno šifriranje i šifriranje podataka na razini pojedinih atributa. Postoji i vrsta šifriranja zvana *potpuno šifriranje diska* (engl. *Full Disk Encryption, FDE*) [12], ali nije specifično za baze podataka već ovisi o pojedinom operacijskom sustavu. Korištenjem FDE-a postiže se da su svi podaci na disku šifrirani te se dešifriraju automatski dok se čitaju s diska. Ova vrsta šifriranje može se koristiti kao zaštita samo u slučaju krađe diska na kojem su pohranjeni podaci.



Slika 2.1. Sigurnosna arhitektura baze podataka [13]

2.3.1 Transparentno šifriranje podataka

Transparentno šifriranje podataka (engl. *transparent data encryption, TDE*) jedno je od popularnijih metoda za zaštitu podataka. Koristi se za zaštitu podataka u mirovanju. Problem zaštite podataka u mirovanju riješen je šifriranjem podataka na tvrdom disku [14].

Podaci se automatski dešifriraju dok ih autorizirani korisnici pregledavaju kroz sustav za upravljanje bazom podataka te se zbog toga naziva transparentnim. Može se zaključiti da podaci koji se koriste nisu zaštićeni od zlonamjernih administratora baza podataka, upravo

zbog toga što ih autorizirani korisnici vide u čistom, nešifriranom obliku. Uz to važno je i napomenuti da podaci u pokretu također nisu zaštićeni korištenjem ove metode.

Korist od transparentnog šifriranja je zaštita podataka u slučaju gubitka fizičkog medija za pohranu na kojem se nalaze podaci. Uz to, podaci su zaštićeni dok su i na disku.

Transparentno šifriranje najčešće je ostvareno šifriranjem svih sadržaja baze podataka na disku koristeći simetričnu kriptografiju. Sustav za upravljanje bazom podataka zatim pri inicijalizaciji baze dohvaća ključ, koji zatim koristi za dešifriranje šifriranih podataka.

Najveći problem koji se javlja kod ovog pristupa šifriranju je pohrana ključa. Jedan od koraka prema sigurnoj pohrani ključa je njegova pohrana na izolirani sustav. Uz sve navedeno, važno je naglasiti da postoji i rizik od gubitka ključa jer ako se ključ izgubi, praktički su i svi podaci u bazi podataka izgubljeni jer je dešifriranje bez ključa gotovo nemoguće.

Primjer baze podataka gdje se koristi ova vrsta šifriranja je MariaDB.

2.3.2 Šifriranje podataka na razini pojedinih atributa

Šifriranje podataka na razini pojedinih atributa (engl. *column level encryption*) vrsta je šifriranja koja omogućava korisniku da šifrira samo dio tablice u bazi podataka. Ova vrsta šifriranja korisna je u slučaju kad je samo dio pohranjenih podataka osjetljiv. Uz navedeno, postoji i mogućnost korištenja različitih ključeva za svaki od atributa relacije, što pridonosi fleksibilnosti uporabe za korisnike [15].

Bitna karakteristika ove vrste šifriranje je što su podaci zaštićeni i dok su u uporabi, za razliku od transparentnog šifriranja.

Korištenje različitih ključeva za različite atribute, uz uvjet da su pohranjeni na siguran i sistematičan način, pridonosi razini sigurnosti baze podataka. To je očevidno u slučaju ako napadač dođe do nekih od ključeva, moći će dešifrirati samo one stupce relacije za koji su korišteni ključevi koje posjeduje.

Primjer baze podataka gdje je dostupna ova metoda šifriranja je Postgres.

2.3.3 Prednosti i nedostaci šifriranja baza podataka

Kad se govori o šifriranju u bazama podataka kao jednom od načina zaštite podataka, vrlo je važno razumjeti koje su prednosti, a koji nedostaci.

Glavna, evidentna prednost šifriranja jest ispunjavanje sigurnosnih zahtjeva klijenata. Šifriranjem podataka se prije svega postiže cjelovitost i povjerljivost. Naravno, vrsta prijetnje određuje koja se vrsta šifriranja koristi.

Naime, sve prednosti dolaze uz određenu cijenu. O kojim nedostacima se radi ponajviše ovisi o tome koja se vrsta šifriranja koristi. Međutim, postoji niz nedostataka koji je zajednički svim vrstama šifriranja. Kao dva glavna nedostatka najčešće se ističu problem pohrane, odnosno upravljanja ključevima za šifriranje te opći vremenski trošak koji se pojavljuje uslijed korištenja šifriranja [16].

Upravljanje ključevima može vrlo brzo prerasti u poprilično zahtjevan problem. Primjerice, to se jasno očituje u slučaju kada su klijentski zahtjevi takvi da je u bazi nužno postaviti različite ključeve za šifriranje nad istim stupcem tablice.

Vremenski trošak potreban za izvođenje upita nad šifriranim podacima jedan je od presudnih faktora zašto se više kompanija ne odlučuje za prijenos svojih podataka na bazu podataka u oblaku [17]. Ako se radi o podacima koji su osjetljivi, nužno je koristiti dobru zaštitu, što s druge strane dovodi do problema filtriranja istih. Poslužitelj treba šifrirati, odnosno dešifrirati podatke nakon svih operacija koje se provode nad podacima, što predstavlja veliki vremenski trošak.

2.4. Efikasnije izvođenje upita nad šifriranim podacima

Problem efikasnog izvođenja upita nad šifriranim podacima jedan je od aktualnih problema današnjice u području šifriranja baza podataka. Sve postojeće predložene sheme imaju određene ranjivosti, najčešće u vidu curenja neke vrste informacije. U nastavku slijedi pregled nekih od predloženih rješenja za izvođenje upita nad šifriranim podacima. Polazni skup podataka zapisan u čistom tekstu prikazan je u tablici 2.1. U prvom stupcu nalaze se vrijednosti identifikatora, a u drugom stupcu cjelobrojne vrijednosti.

Tablica 2.1. Ulazni skup podataka zapisan u čistom tekstu

ID	Vrijednost
1	5
2	11
3	5
4	7
5	3

2.4.1 Deterministička shema šifriranja

Glavna karakteristika determinističke sheme je što za dani čisti tekst i ključ uvijek stvara isti šifrat [18]. Takav način šifriranja danas predstavlja relativno uobičajeno rješenje za potrebe efikasnijeg izvođenja upita nad šifriranim podacima. U tablici 2.2. prikazana je ilustracija kako bi izgledale vrijednosti iz tablice 2.1. da su šifrirane nekim izmišljenim deterministčkim algoritmom. Ključno je primijetiti da podaci u recima gdje su vrijednosti identifikatora 1 i 3 imaju jednake šifrate.

Tablica 2.2. Ilustracija šifriranja deterministčkim algoritmom

ID	Vrijednost
1	0x3b45e2
2	0xa21b3d
3	0x3b45e2
4	0x10a1ce
5	0x5aa4fd

Deterministčke sheme omogućavaju izvršavanje upita nad intervalima pod uvjetom da je praktično napraviti enumeraciju svih vrijednosti unutar intervala. Primjerice, neka klijent želi dohvatiti sve retke gdje je vrijednost između 3 i 5. Da bi poslužitelj ispunio taj zahtjev

jednostavno treba vratiti sve retke u kojima je vrijednost iz skupa {DET(3), DET(4), DET(5)} gdje DET(X) znači da se vrijednost X šifrira deterministčkim algoritmom.

2.4.2 OPE i ORE sheme

Šifriranje s očuvanjem poretka (engl. *Order Preserving Encryption, OPE*) [18][19] vrsta je šifriranja gdje je očuvan poredak podataka. Dakle, ako je x<y, onda je i OPE(x)<OPE(y). U tablici 2.3. prikazana je ilustracija kako bi mogle izgledati vrijednosti iz tablice 2.1. nakon šifriranja koristeći OPE. Ključno je primijetiti da je poredak vrijednosti očuvan. Neki algoritmi OPE šifriranja su deterministički, što je i prikazano u tablici 2.1. gdje se kao i u primjeru za determinističke sheme šifriranja može primijetiti da podaci u recima gdje su vrijednosti identifikatora 1 i 3 imaju jednake vrijednosti šifrata.

Tablica 2.3. Ilustracija šifriranja koristeći OPE

ID	Vrijednost
1	231
2	5433
3	231
4	1572
5	126

Šifriranje s otkrivanjem poretka (engl. *Order Revealing Encryption, ORE*) [17][18] generalizacija je OPE-a. Kod OPE-a moguće je jednostavnom usporedbom odrediti koja je od dvije vrijednosti manja. ORE se razlikuje po tome što je nužno korištenje funkcije nad parom vrijednosti kako bi ih se usporedilo. Vrsta funkcije ovisi o konkretnoj implementaciji.

OPE i ORE sheme su korisne jer omogućavaju upite nad intervalima. Važno je napomenuti da nisu sasvim sigurne. Rekonstrukcijski napad vrsta je napada koji iskorištava određena svojstva ORE i OPE shema. Na temelju pročitanih odgovora na upit moguće je rekonstruirati vrijednosti podataka iz relacije bez da su poznate krajnje vrijednosti pojedinih upita nad intervalima. Na jednostavnom primjeru može se pokazati zašto je to moguće.

Neka relacija sadrži 10 zapisa s identifikatorima od 1 do 10. Ako napadač uoči da se kao odgovor na upite nad intervalima javljaju identifikatori {2,3,5,10} i {2,4,5,8}, može zaključiti da zapisi s identifikatorima 3 i 10 imaju vrijednosti manje od zapisa s identifikatorima 2 i 5, koji imaju vrijednosti manje od zapisa s identifikatorima 4 i 8. Nakon dovoljno velikog broja upita i uz odrađen niz operacija presjeka na skupovima, moguće je grupirati zapise i sortirati ih po vrijednosti.

2.4.3 Slijepi indeksi

Slijepi indeksi (engl. *blind index*) jedna su od strategija/tehnika koje predstavljaju korak prema efikasnijoj pretrazi šifriranih vrijednosti [20].

Ideja ove tehnike je stvaranje sažetka vrijednosti koja se šifrira i spremanje iste u zasebni stupac tablice. Upiti se rade na način da se prije izvršavanja svakog upita stvori sažetak vrijednosti koja se pretražuje, a potom se ta vrijednost pretražuje u bazi. Time se zapravo samo omogućavaju upiti koji koriste točno podudaranje.

Postavljaju se pitanja kada je korištenje slijepih indeksa korisno te koji su potencijalni novi propusti stvoreni njihovim korištenjem.

Za početak važno je napomenuti da se iz sažetaka podataka mogu doznati neke informacije. Ako su dva podatka sadržajno jednaka, primjerice ako se radi o dva jednaka znakovna niza, njihovi sažeci bit će isti. To u konačnici znači da napadač koji nekako dođe do šifriranih podataka i vidi dvije iste vrijednosti u stupcu koji se koristi kao slijepi indeks može zaključiti da se radi o dva atributa s istom vrijednošću. Prema tome, potrebno je biti izrazito oprezan pri odabiru za koje vrijednosti se stvaraju takvi indeksi.

Nadalje, u slučaju da se radi o napadaču koji može utjecati na podatke u bazi, primjerice dodavanjem novih ili ažuriranjem postojećih, te ako može nadzirati pohranjene vrijednosti slijepih indeksa, moguće je da izvrši napad u kojem bi prošao po svim mogućim vrijednostima polja nad kojima je stvoren indeks te napravio poveznicu između tog polja i pripadnog slijepog indeksa. Korištenje slijepih indeksa preporuča se za podatke čije vrijednosti imaju visoku entropiju ili za podatke koji nisu sasvim osjetljivi.

Iako slijepi indeksi omogućavaju samo upite za pretraživanje gdje je točno podudaranje, moguće je i stvoriti indekse na takav način da se mogu postavljati upiti gdje se primjerice traže sva polja čije vrijednost počinje na neko slovo [20].

2.5. Model prijetnje za šifriranja u bazama podataka

Prije svega, potrebno je objasniti što je to model prijetnje i proces modeliranja prijetnji. Model prijetnje je strukturirani prikaz svih prijetnji koje utječu na sigurnost neke aplikacije. Modeliranje prijetnje je proces prikupljanja, organiziranja i analize tih informacija. Riječ prijetnja u kontekstu modeliranja prijetnji odnosi se na zlonamjerno djelovanje osobe, namjerno ili nenamjerno, te na šansu da nešto pođe po zlu, primjerice kvar medija za pohranu na kojem se nalaze podaci neke baze podataka [21][22].

Glavni cilj modeliranja prijetnji je utvrditi koje sve prijetnje djeluju na sustav. Na temelju modela prijetnji poduzimaju se mjere kojima je cilj ukloniti postojeće prijetnje. Modeliranje prijetnji svakako je iterativan proces. Bilo da se radi o postojećem sustavu ili sustavu koji je tek u izradi, korištene tehnologije stalno se mijenjaju. Uvođenje novih tehnologija, pa čak i ažuriranje postojećih, može dovesti do stvaranja novih ranjivosti. Iz toga se može zaključiti da je nužno i da model prijetnje ostane ažuran.

Resurs (engl. *asset*) predstavlja vrstu informacije koja se želi zaštititi, a koja napadačima predstavlja metu.

Modeliranje prijetnji, s obzirom na što je usmjereno, može se podijeliti u tri vrste.

1. Modeliranje prijetnji usmjereno na napadača (engl. *attacker-centric*). U ovoj vrsti modeliranja procjenjuju se ciljevi napadača, način na koji se ti ciljevi mogu ostvariti te sama motivacija napadača.
2. Modeliranje prijetnji usmjereno na softver (engl. *software-centric*). To je pristup koji se temelji na analizi dizajna, odnosno arhitekture sustava te se pokušava otkriti mogući napad na svaki od modula. Primjerice, alat za modeliranje prijetnji sadržan u Microsoftovom SDL-u (*Software Development Tool*) radi na ovaj način.
3. Modeliranje prijetnji usmjereno na resurse (engl. *asset-centric*). Glavna karakeristika ovog pristupa je to što je usredotočen na osjetljive resurse sustava. Resursima se

pridodaje mjera važnosti. Procjenjuje se koliko je koji resurs osjetljiv i koliko je vrijedan napadačima kako bi se odredili prioriteti zaštite.

Pri modeliranju prijetnji postoji pet ključnih koraka:

1. Definiranje sigurnosnih zahtjeva
2. Stvaranje aplikacijskog dijagrama
3. Identifikacija prijetnji
4. Uklanjanje prijetnji
5. Potvrda da su prijetnje uklonjene

Postoji mnogo struktura koje se koriste pri identificiranju prijetnji. Jedna od poznatijih i često korištenih danas je *STRIDE* model. Riječ STRIDE zapravo je akronim. U tablici 2.4. može se vidjeti što svako slovo iz akronima [22][23] STRIDE znači. U desnom stupcu iste tablice predstavljeno je pojašnjenje svakog od navedenih pojmova.

Tablica 2.4. Objašnjenje STRIDE modela za identifikaciju prijetnji

Prijetnja	**Definicija**
Lažiranje (engl. ***S**poofing*)	Napadač se pretvara da je netko/nešto što zapravo nije
Uplitanje (engl. ***T**ampering*)	Napadač pokušava izmijeniti podatke pohranjene u bazi podataka ili podatke koji se kreću između korisnika i aplikacije
Poricanje (engl. ***R**epudiation*)	Napadač poriče akciju u kojoj je sudjelovao
Otkrivanje informacija (engl. ***I**nformation disclosure*)	Napadač dolazi u pristup privatnim informacijama
Uskraćivanje usluge (engl. ***D**enial of service*)	Napadač zlonamjernim djelovanjem uskraćuje korištenje usluge legitimnim korisnicima
Prisvajanje prava (engl. ***E**levation of privilege*)	Napadač koji ima manja prava zlonamjernim djelovanjem dolazi do prava koja ima privilegiraniji korisnik

Također, postoje i gotovi modeli za procjenu rizika. Primjer jednog takvog modela je *DREAD* [22][24]. U tablici 2.5. prikazano je značenje svake od komponenti od koje se DREAD model sastoji.

Svakoj komponenti potrebno je dodijeliti ocjenu od 0 do 10. Primjerice, kad bi se ocjenjivala komponenta mogućnosti reprodukcije za neku prijetnju, ocjena 0 označavala bi da je napad vrlo težak ili nemoguć, čak za administratore aplikacije, dok bi ocjena 7 mogla značiti da je napad lako izvesti za autoriziranog korisnika.

Konačna ocjena rizika prema DREAD modelu dobije se tako da se zbroj svih ocjena podijeli s 5, što znači da će konačna ocjena uvijek biti broj između 0 i 10.

Tablica 2.5. Objašnjenje DREAD modela za procjenu rizika prijetnji

Kategorija	**Objašnjenje**
Mogućnost štete (engl. ***D**amage potential*)	Koliko štete će biti učinjeno ako se iskoristi propust
Mogućnost reprodukcije (engl. ***R**eproducibility*)	Koliko je teško izvesti napad
Iskoristivost (engl. ***E**xploitability*)	Koliko je teško iskoristiti prijetnju
Zahvaćeni korisnici (engl. ***A**ffected users*)	Koliko je korisnika zahvaćeno
Mogućnost otkrivanja (engl. ***D**iscoverability*)	Koliko je teško otkriti prijetnju

U nastavku je prikazan pokušaj generalizacije modela prijetnje za šifriranje u bazama podataka. Korišten je STRIDE model za identificiranje prijetnji. Rezultati su prikazani u tablici 2.6.

Pretpostavka je da se pod naziv šifriranje u bazama podataka ubrajaju sve moguće vrste šifriranja koje imaju veze s bazama podataka. Konkretno, to znači da se razmatra zaštita podataka na komunikacijskom putu uporabom protokola TLS i zaštita podataka u mirovanju/upotrebi korištenjem šifriranja na razini atributa ili transparentnog šifriranja.

Tablica 2.6. Model prijetnje za šifriranja u bazama podataka

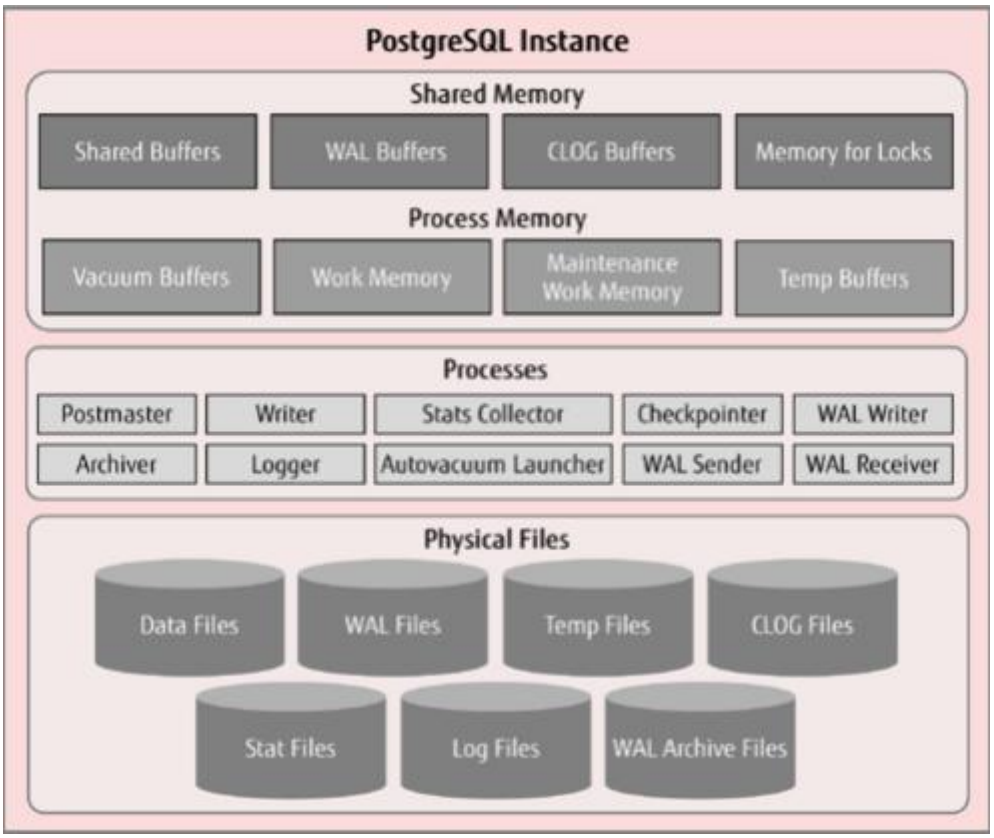
Prijetnja	**Vrsta prijetnje**	**Metoda zaštite**
Krađa diska na kojem su pohranjeni podaci	Otkrivanje informacija	Transparentno šifriranje ili šifriranje na razini pojedinih atributa
Izmjena podataka spremljenih u bazi	Uplitanje	Šifriranje na razini pojedinih atributa
Izmjena podataka na komunikacijskom putu	Uplitanje	Korištenje TLS protokola
Čitanje podataka na komunikacijskom putu	Otkrivanje informacija	Korištenje TLS protokola
Neovlašteni pristup podacima od strane sistemskog administratora	Otkrivanje informacija	Šifriranje na razini pojedinih atributa
Neovlašteni pristup podacima od strane administratora baze podataka	Otkrivanje informacija, Uplitanje	Šifriranje na razini pojedinih atributa

3. Šifriranje u bazi podataka Postgres

3.1. Arhitektura baze podataka Postgres

Što se tiče arhitekture sustava, Postgres koristi model klijent-poslužitelj. U komunikaciji klijenta i poslužitelja sudjeluju dva procesa. Prvi proces, poslužiteljski, služi za upravljanje podacima baze podataka, prihvaća klijentske veze i izvršava klijentske zahtjeve. Drugi proces, klijentski, predstavlja bilo kakvu vrstu klijentske aplikacije i služi kako bi omogućio klijentima komunikaciju i zahtijevanje izvršavanja naredbi od poslužitelja. Postgres poslužitelj može prihvatiti više klijentskih veza.

Fizička arhitektura Postgresa sastoji se od datoteka pohranjenih na disku, dijeljene memorije, niza pozadinskih procesa i memorije koju ti procesi koriste [25]. Na slici 3.1. može se vidjeti ta podjela.



Slika 3.1. Fizička arhitektura Postgresa [25]

3.1.1 Dijeljena memorija, procesi i memorija za procese

Najvažniji dio dijeljene memorije su dijeljeni međuspremnici (engl. *shared buffers*) i WAL međuspremnici (engl. *Write Ahead Log buffers, WAL*). Dijeljeni međuspremnici služe kao privremena memorija za Postgres kako bi se što brže obavljale operacije čitanja i pisanja. WAL međuspremnici služe za zapisivanje metapodataka o promjenama podataka u bazi, što u osnovi služi za rekonstrukciju podataka pri oporavku baze.

Osim spomenutih dijelova, može se primijetiti da u dijeljenom dijelu memorije još postoje i CLOG međuspremnici (engl. *Commit log buffer, CLOG*) i memorija za zaključavanje (engl. *memory for locks*). CLOG međuspremnici služe za zapisivanje metapodataka o transakcijama, odnosno bilježi se je li pojedina transakcija izvršena ili ne. Memorija za zaključavanje se koristi pri zaključavanju određenih tablica kako ne bi došlo do istovremenog pristupa.

Od svih procesa koje Postgres koristi, najvažniji je Postmaster. Postmaster je prvi proces koji se pokreće pri pokretanju Postgresa, roditelj je svim ostalim procesima Postgresa, inicijalizira dijeljenu memoriju te izravno stvara procese koji služe za prihvaćanje klijentskih veza.

Logger proces zapisuje poruke o greškama u log datoteku. *Checkpointer* proces povremeno stvara kontrolne točke tijekom kojih zapisuje izmijenjene podatke na disk. *Writer* proces služi za premještanje izmijenjenih podataka na disk kako bi se što manje podataka trebalo prebaciti na disk kada dođe nova kontrolna točka. Procesi *WAL writer, sender* i *receiver* služe za organizaciju i zapisivanje metapodataka o promjenama u bazi na disk [26].

Stats collector proces služi za skupljanje statističkih informacija o bazi. *Archiver* proces kopira WAL datoteku u definiranu datoteku. Konačno, *Autovacuum launcher* proces služi za aktivaciju automatskog čišćenja. Glavna svrha automatskog čišćenja je brisanje podataka, odnosno redaka tablica koji su označeni za brisanje.

Vacuum Buffers su međuspremnici koje prethodno opisani *autovacuum* proces koristi. Radna memorija (engl. *Work Memory*) je memorija rezervirana za izvršavanje operacija sortiranja, primjerice *order by* ili *distinct* te operacije izračunavanja sažetka, primjerice *hash-join*.

Radna memorija za održavanje (engl. *Maintenance Work Memory*) je memorija rezervirana za izvršavanje operacija poput dodavanja stranog ključa tablici i stvaranja indeksa. Privremeni međuspremnici (engl. *Temp Buffers*) služe kao memorija za privremene tablice koje postoje u bazi podataka.

3.1.2 Lokacija pohranjenih podataka na disku

Sve datoteke vezane za Postgres spremaju se u isti bazni direktorij. Lokacija tog direktorija može se saznati pokretanjem naredbe prikazane u ispisu 3.1. To je lokacija pohranjena u sustavskoj varijabli *PGDATA* [27].

```
SHOW data_directory;
```

Ispis 3.1. Otkrivanje baznog direktorija baze podataka Postgres

Unutar tog direktorija postoje tri poddirektorija gdje se spremaju tablice. Prvi od spomenuta tri poddirektorija je */base*, unutar kojeg se nalaze stvorene baze podataka u *pg_default* tabličnom prostoru (engl. *tablespace*). Drugi poddirektorij, odnosno */global*, služi za pohranu tablica koje ne pripadaju nijednoj bazi podataka, takozvani *pg_global* tablični prostor. Posljednji poddirektorij je */pg_tblspc* koji sadrži simboličke poveznice na tablične prostore izvan okoline *PGDATA*.

Svaka baza podataka ima svoj poddirektorij čije ime je određeno njezinim identifikatorom OID (engl. *Object Identifier*). OID-ovi pojedinih baza mogu se saznati pokretanjem naredbe prikazane u ispisu 3.2.

```
SELECT datname, oid FROM pg_database;
```

Ispis 3.2. Pretraživanje OID vrijednosti pojedinih baza podataka

Također, naredbom *pg_relation_filepath* može se saznati točna lokacija tablice na disku. Rezultat te naredbe je relativna putanja u odnosu na prethodno spomenuti */base* direktorij. Primjer korištenja može se vidjeti u ispisu 3.3. Primjer korištenja na konkretnoj tablici zajedno sa potvrdom da se mogu vidjeti i spremljeni podaci prikazan je u poglavlju o napadaču na bazi podataka.

```
SELECT pg_relation_filepath('table_name');
```

Ispis 3.3. Upit za lokaciju tablice *table_name* na disku

3.2. Dostupne metode šifriranja u modulu pgcrypto

Postgres korisnicima nudi niz mogućnosti za šifriranje podataka [28]. U nastavku slijedi pregled dostupnih metoda.

3.2.1 Metode za generiranje sažetaka

Modul *pgcrypto* omogućava šifriranje pojedinih atributa relacije. To je korisno kada je samo dio podataka iz relacije osjetljiv. U okviru *pgcrypta* dostupan je niz kriptografskih metoda [29].

Dostupne su dvije metode za stvaranje sažetaka. Radi se o metodama *digest* i *hmac*.

Kao što se vidi na ispisu 3.4., metoda *digest* prima dva parametra. Kao prvi parametar predaje se podatak čiji sažetak se stvara. Drugi parametar *type* označava vrstu algoritma koji se koristi. Dostupni algoritmi su md5, sha1, sha224, sha256, sha384 i sha512.

```
digest(data text, type text) returns bytea

digest(data bytea, type text) returns bytea
```

Ispis 3.4. Definicija metode *digest*

Metoda *hmac* prikazana na ispisu 3.5. prima tri parametra. Prvi i treći parametar odgovaraju prvom i drugom parametru metode *digest*. Drugi parametar, *key*, služi kao ključ pri stvaranju sažetka.

```
hmac(data text, key text, type text) returns bytea

hmac(data bytea, key bytea, type text) returns bytea
```

Ispis 3.5. Definicija metode *hmac*

Sažetak koji stvaraju prethodne dvije metode tipa je *bytea*. Tip podataka *bytea* predstavlja binarni znakovni niz, koji se od običnog znakovnog niza razlikuje po tome što dozvoljava

neispisive (engl. *non printable*) znakove. Maksimalna dozvoljena veličina podatka koji je tipa *bytea* jest 1 GB.

Postoje i još dvije metode koje se koriste za stvaranje sažetaka, ali se koriste specifično za stvaranje sažetaka lozinki. Radi se o metodama *crypt* i *gen_salt*.

Metoda *crypt* prikazana u ispisu 3.6. služi za generiranje sažetka lozinke, dok metoda *gen_salt* zapravo služi kao pomoćna metoda za generiranje drugog parametra metode *crypt*.

```
crypt(password text, salt text) returns text
```

Ispis 3.6. Definicija metode *crypt*

Pri prvom pohranjivanju nove lozinke nužno je za drugi parametar, *salt*, koristiti *gen_salt* metodu. To se može vidjeti na primjeru postavljanja nove lozinke prikazanom u ispisu 3.7.

```
UPDATE ... SET pswd_hash = crypt('new password', gen_salt('md5'));
```

Ispis 3.7. Pohrana nove lozinke

Pri provjeri valjanosti lozinke unesene od strane korisnika kao drugi parametar metode crypt koristi se lozinka koju je korisnik unio, kao što je prikazano u ispisu 3.8.

```
SELECT (pswd_hash = crypt('entered password', pswhash)) AS pswd_match FROM ... ;
```

Ispis 3.8. Provjera valjanosti unesene lozinke

Metoda *gen_salt* prikazana je u ispisu 3.9. Ona zapravo generira salt dodatak koji predstavlja nasumično generiran niz koji se koristi kao nadopuna lozinki koju korisnici stvaraju. Parametar *type* određuje koji će se algoritam koristiti pri generiranju salt vrijednosti. Moguće vrijednosti parametra *type* su *des, xdes, md5* i *bf5*. Također, na temelju generirane salt vrijednosti, metoda *crypt* zna koji algoritam se treba koristiti za generiranje samog sažetka. Drugi, opcionalni parametar služi za određivanje koliko će se iteracija izvršiti pri generiranju salt dodatka kod onih algoritama koji prolaze kroz više iteracija.

```
gen_salt(type text [, iter_count integer ]) returns text
```

Ispis 3.9. Definicija metode *gen_salt*

U usporedbi s metodama *digest* i *hmac* koje također generiraju sažetke, metode *crypt* i *gen_salt* znatno su sporije. Razlog tomu je upravo činjenica da se koriste za generiranje sažetaka lozinki, koje su u općem slučaju kratke duljine pa je nužno da njihovi sažeci budu iznimno otporni na brute-force napade. Također, iz ispisa 3.6 i 3.9. može se primijetiti da *crypt* i *gen_salt* vraćaju rezultat koji je tipa *text*.

3.2.2 Metode za simetrično i asimetrično šifriranje

Metode za simetrično i asimetrično šifriranje u Postgresu dio su OpenPGP standarda [30].

Dio metoda opisanih u nastavku sadrži parametar *options* koji služi za specificiranje niza dostupnih opcija. Pregled nekih od važnijih opcija prikazan je nakon opisa svih metoda.

Metoda *pgp_sym_encrypt*, prikazana u ispisu 3.10. služi za šifriranje podatka zadanog prvim parametrom simetričnim ključem specificiranim drugim parametrom *psw*.

```
pgp_sym_encrypt(data text, psw text [, options text ]) returns bytea
pgp_sym_encrypt_bytea(data bytea, psw text [, options text ]) returns bytea
```

Ispis 3.10. Definicija metode *pgp_sym_encrypt*

Metoda *pgp_sym_decrypt* prikazana u ispisu 3.11. služi za dešifriranje podatka zadanog prvim parametrom koristeći simetrični ključ specificiran drugim parametrom.

```
pgp_sym_decrypt(msg bytea, psw text [, options text ]) returns text
pgp_sym_decrypt_bytea(msg bytea, psw text [, options text ]) returns bytea
```

Ispis 3.11. Definicija metode *pgp_sym_decrypt*

Metoda *pgp_pub_encrypt* prikazana u ispisu 3.12. služi za šifriranje podatka javnim PGP ključem.

```
pgp_pub_encrypt(data text, key bytea [, options text ]) returns bytea
```

Ispis 3.12. Definicija metode *pgp_pub_encrypt*

Metoda *pgp_pub_decrypt* prikazana u ispisu 3.13. koristi se za dešifriranje poruke koja je šifrirana javnim ključem. Drugi parametar, odnosno ključ, mora biti tajni ključ koji odgovara javnom ključu koji je bio korišten za šifriranje.

```
pgp_pub_decrypt(msg bytea, key bytea [, psw text [, options text ]]) returns text
```

Ispis 3.13. Definicija metode *pgp_pub_decrypt*

Što se tiče parametra *options*, jedna od zanimljivijih opcija je *cipher-algo*, koji služi za odabir algoritma za šifriranje koji će se koristiti u metodama *pgp_sym_encrypt* i *pgp_pub_encrypt*, s mogućim vrijednostima *bf, aes128, aes192, aes256,* dok je pretpostavljena vrijednost *aes128*.

Osim opcije *cipher-algo* jedna od važnijih opcija je *unicode-mode*, koja određuje hoće li se pri izvođenju metoda *pgp_sym_encrypt* i *pgp_pub_encrypt* tekstualni podaci iz baze prije šifriranja pretvarati u tekstualne podatke s načinom kodiranja UTF-8. Moguće vrijednosti su 0 i 1, a pretpostavljena vrijednost je 0.

Postoji i niz drugih opcija, ali preporuča se da se one ostave na pretpostavljenim vrijednostima.

Ukoliko se nekoj od metoda kao jedan od parametara preda vrijednost *NULL*, sama metoda će uvijek kao rezultat vratiti *NULL*.

Algoritmi za šifriranje koji se mogu koristiti u metodi *pgp_sym_encrypt* nisu deterministički. To znači da kad bi se šifrirao neki podatak više puta istim ključem, šifrat bi svaki put bio različit. Stoga se može zaključiti da nije iz baze moguće direktno pretraživati šifrirani oblik.

Isto tako, moguće je primijetiti da sve metode za simetrično i asimetrično šifriranje dostupne u modulu *pgcrypto* vraćaju rezultat koji je po tipu *bytea*. Također, može se primijetiti da je moguće šifrirati samo podatke koji su po tipu *bytea* i *text*. Ta činjenica predstavlja ograničenje

jer je dostupnim metodama nemoguće šifrirati sve vrste podataka koji se ne mogu operacijom *CAST* pretvoriti u *bytea* ili *text*.

Operator *CAST* koristit će se u nastavku kod primjera šifriranja. Služi za pretvorbu jednog tipa podatka u drugi. Koristi se na način prikazan u ispisu 3.14.

```
CAST ( expression AS target_type );
```

Ispis 3.14. Definicija operatora *CAST*

3.3. Primjer šifriranja

U nastavku će se demonstrirati šifriranje u bazi podataka Postgres korištenjem modula pgcrypto. Koristiti će se simetrično šifriranje pretpostavljenim algoritmom, koji je u ovom slučaju AES-128.

Za početak stvara se baza podataka *encryption_example_database*, što je prikazano u ispisu 3.15. Za vlasnika baze postavit će se postgres, a kao način kodiranja postavlja se UTF-8.

```
CREATE DATABASE "encryption_example_database"

WITH

OWNER = postgres

ENCODING = 'UTF8'
```

Ispis 3.15. Stvaranje baze podataka

Općenito je dobra praksa koristiti sheme jer omogućavaju organizaciju objekata baze podataka u logičke cjeline i organizaciju korisnika na način da ih više može koristiti istu bazu podataka [31]. Sukladno tome, u ovom primjeru unutar stvorene baze podataka stvara se shema *encryption_example_schema* što se može vidjeti u ispisu 3.16. Korištenje te sheme omogućuje se korisniku postgres.

```
CREATE SCHEMA "encryption_example_schema"
AUTHORIZATION postgres;
```

Ispis 3.16. Stvaranje sheme

Unutar stvorene sheme stvara se relacija *company_user_details*. Stvorena relacija koristit će se kod primjera šifriranja i predstavlja detalje o pojedinom zaposleniku neke kompanije. Sastoji se od četiri atributa. Prvi atribut je identifikator koji se automatski generira i predstavlja primarni ključ relacije. Ostala tri atributa određuju detalje pojedinog zaposlenika, konkretno njegovo ime, ulogu i dob. U ovom primjeru pretpostavlja se da su podaci spremljeni u stupcima *name*, *role* i *age* svi osjetljivi i da ih je potrebno šifrirati. Stvaranje tablice prikazano je u ispisu 3.17.

```
CREATE TABLE "encryption_example_schema"."company_user_details"
(
id serial NOT NULL,
 name bytea NOT NULL,
 role bytea NOT NULL,
 age bytea NOT NULL,
 PRIMARY KEY (id)
 )
```

Ispis 3.17. Stvaranje tablice

Prije svih upita u kojima će se koristiti modul *pgcrypto* potrebno je taj modul prvo instalirati. Naredbu iz ispisa 3.18. potrebno je pozvati samo jednom.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Ispis 3.18. Instalacija modula *pgcrypto* unutar baze

U idućem koraku prikazanom u ispisu 3.19. predstavljen je primjer umetanja novih podataka u tablicu. Koristi se simetrično šifriranje gdje je kao drugi parametar, odnosno ključ metodi *pgp_sym_encrypt* predano *„AES_KEY*”.

```
INSERT INTO "encryption_example_schema"."company_user_details"
(name, age, role) VALUES (
PGP_SYM_ENCRYPT('John', 'AES_KEY'),
 PGP_SYM_ENCRYPT('30', 'AES_KEY'),
PGP_SYM_ENCRYPT('CEO' ,'AES_KEY')
 );
```

Ispis 3.19. Unos podataka u tablicu

Provede li se upit iz ispisa 3.20., odnosno pretraživanje svih podataka iz relacije:

```
SELECT * FROM "encryption_example_schema"."company_user_details";
```

Ispis 3.20. Pretraživanje svih podataka iz tablice

dobiva se rezultat vidljiv na slici 3.2., iz kojeg je vidljivo da su osjetljivi podaci spremljeni u šifriranom obliku.

	id [PK] integer	name text	role text	age text
1	1	\303\015\004\007\003\002g\24...	\303\015\004\007\003\002\360\\\21...	\303\015\004\007\003\002\012\351\201...

Slika 3.2. Rezultat pretraživanja svih redaka šifrirane tablice bez dešifriranja

U ispisu 3.21. demonstrira se način na koji bi se određeni podaci pretraživali u tablici. Konkretno u primjeru traže se podaci o svim zaposlenicima čije polje *name* u nekom dijelu sadrži riječ John .

```
SELECT
      PGP_SYM_DECRYPT(CAST(name AS BYTEA), 'AES_KEY') as name,
      PGP_SYM_DECRYPT(CAST(role AS BYTEA), 'AES_KEY') as role,
      PGP_SYM_DECRYPT(CAST(age AS BYTEA), 'AES_KEY') as age
FROM "encryption_example_schema"."company_user_details" WHERE (
      LOWER(PGP_SYM_DECRYPT(CAST(name AS BYTEA), 'AES_KEY'))
      LIKE LOWER ('%John%')
);
```

Ispis 3.21. Pretraživanje određenog podataka u šifriranoj tablici

Rezultat upita prikazan je na slici 3.3., a može se primijetiti da je pretraživanje dalo očekivani rezultat.

	name text	role text	age text
1	John	CEO	30

Slika 3.3. Rezultat pretraživanja svih redaka uz dešifriranje

U primjeru prikazanom u ispisu 3.22. demonstrira se ažuriranje podataka za redak u tablici čiji je *id* jednak 1.

```
UPDATE "encryption_example_schema"."company_user_details" SET
(role, age) = (
       PGP_SYM_ENCRYPT('CTO' , 'AES_KEY'),
      PGP_SYM_ENCRYPT('31', 'AES_KEY')
 ) WHERE id='1';
```

Ispis 3.22. Ažuriranje podataka u tablici

Provede li se sad upit za pretraživanje iz prošlog primjera, dobiva se rezultat prikazan na slici 3.4. Može se primijetiti da su podaci ispravno ažurirani.

	name text	role text	age text
1	John	CTO	31

Slika 3.4. Rezultat pretraživanja svih redaka uz dešifriranje nakon izvršenog ažuriranja

4. Analiza napadača na bazi podataka i na komunikacijskom putu

U ovom poglavlju predstavljene su analize dviju vrsta napadača. U prvom dijelu analizira se napadač koji se nalazi na bazi podataka, a u drugom dijelu analizira se napadač koji se nalazi na komunikacijskom putu. Cilj analize je saznati koliko takvi napadači mogu saznati informacija o pohranjenim podacima. Naglasak je na otkrivanje informacija o podacima koji su pohranjeni u bazi podataka Postgres.

4.1. Analiza napadača na bazi podataka

Jedan od nedostataka Postgresa je to što trenutno nema ugrađenu podršku za transparentno šifriranje podataka. Unatoč tome, kao što je već pokazano, podaci se mogu zaštititi modulom *pgcrypto*. Moguća prijetnja napadača na bazi podataka, primjerice zlonamjernog sistemskog administratora ili administratora baze podataka kompanijama predstavlja najveću prepreku za prijenos baze podataka u oblak. U nastavku slijedi analiza tri vrste napadača koji se nalaze na bazi podataka. Prvo se razmatra napadač koji je korisnik bez administratorskih ovlasti i pristupa bazi na računalu gdje se nalazi baza podataka. Zatim se razmatra napadač koji je administrator na računalu gdje se nalazi baza podataka i napadač koji je administrator na samoj bazi podataka.

4.1.1 Napadač korisnik na računalu bez administratorskih ovlasti i pristupa bazi

Kao što je već spomenuto u poglavlju koje se bavi arhitekturom Postgresa, podaci spremljeni u bazi podataka mogu se pronaći na disku. Ako podaci nisu zapisani u šifriranom obliku, svatko tko ima pristup datotekama na disku gdje se nalaze tablice može pronaći nešifrirane podatke.

Najvažniji korak, te ujedno i korak koji bi se prvi trebao poduzeti protiv potencijalnih napadača koji imaju pristup računalu na kojem se nalazi baza podataka je postavljanje odgovarajućih dozvola za čitanje i pisanje. Konkretno, dozvole bi trebalo postaviti tako da pristup svih direktorijima Postgresa imaju samo ovlašteni korisnici, odnosno oni kojima je to i

namijenjeno. U nastavku slijedi primjer kojem je svrha pokazati kako se može doći do podataka iz baze zapisanih na disku.

Za početak u ispisu 4.1. stvara se tablica *test* i u nju se dodaje testni podatak *'sample text'*.

```
CREATE TABLE test
(
      id serial,
      text_data text
)
INSERT INTO test (text_data) VALUES
(
      'sample text'
);
```

Ispis 4.1. Stvaranje tablice test i dodavanje testnog podatka

Idući korak je saznavanje gdje se stvorena tablica nalazi na disku. Prvi korak je otkrivanje baznog direktorija Postgresa prikazan u ispisu 4.2.

```
SHOW data_directory;
/var/lib/postgresql/12/main
```

Ispis 4.2. Otkrivanje baznog direktorija Postgresa

Zatim se otkriva putanja do stvorene tablice na način prikazan u ispisu 4.3.

```
SELECT pg_relation_filepath('test');
base/16536/16607
```

Ispis 4.3. Otkrivanje putanje do stvorene tablice

Odnosno putanja do direktorija gdje se tablica nalazi na disku izgleda kao što je prikazano u ispisu 4.4.

```
/var/lib/postgresql/12/main/base/16536
```

Ispis 4.4. Putanja do direktorija gdje se nalazi tablica test

Podaci tablice nalaze se unutar datoteke *16607* u prethodno spomenutom direktoriju. Nakon pozicioniranja u direktorij iz ispisa 4.4., preostaje vidjeti kako se može vidjeti testni podatak koji je zapisan u tablicu. Prikaz sadržaja datoteke će se izvesti korištenjem alata *hexdump*, koji služi za prikazivanje podataka u raznim dostupnim formatima. Uz *hexdump*, u ispisu 4.5. može se primijetiti da je korištena i zastavica *-C*, koja određuje da se ispis prikaže u heksadekadskom i ASCII obliku.

Konačno, u ispisu 4.5. vidljiv je rezultat pokretanja naredbe *hexdump*. Vidljivo je da je ulazni testni podatak, *'sample text'*, vidljiv u obliku čistog teksta.

```
hexdump -C 16607
00000000  00 00 00 00 a0 35 b6 01  00 00 00 00 1c 00 d8 1f  |.....5..........|

00000010  00 20 04 20 00 00 00 00  d8 9f 50 00 00 00 00 00  |. . ......P.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001fd0  00 00 00 00 00 00 00 00  60 02 00 00 00 00 00 00  |........`.......|
00001fe0  00 00 00 00 00 00 00 00  01 00 02 00 02 09 18 00  |................|
00001ff0  01 00 00 00 19 73 61 6d  70 6c 65 20 74 65 78 74  |.....sample text|
00002000
```

Ispis 4.5. Ispis tablice s diska

Zaključno s prethodno prikazanim primjerom pokazuje se da je vrlo važno osigurati odgovarajuće dozvole pristupa datotekama kako neautorizirani korisnici ne bi imali pristup podacima iz baze.

Naravno, korištenjem šifriranja modulom *pgcrypto,* u prethodnom primjeru ne bi se vidjeli podaci zapisani u čistom tekstu već bi bili šifrirani. To predstavlja dodatni korak u zaštiti podataka od korisnika na računalu.

4.1.2 Napadač administrator na računalu

Slučaj u kojem je napadač administrator na računalu na kojem se nalazi baza podataka vrlo je sličan prethodno opisanom slučaju kada je napadač korisnik na računalu bez administratorskih ovlasti. Glavna, očita, razlika je to što u ovom slučaju korisnik ima administratorske ovlasti.

Sukladno tome, može se zaključiti da ovakav napadač može, bez većih poteškoća, doći do podataka zapisanih na disku. Uz administratorske ovlasti, i bez pristupa samoj bazi, lako se može otkriti gdje su zapisani podaci Postgres sustava za upravljanja bazama podataka. To se može ostvariti primjerice korištenjem alata *grep*. Također, uz administratorske ovlasti na računalu napadač može jednostavno doći do administratorskih ovlasti na bazi podataka, a zatim se ovaj slučaj pretvara u slučaj napadača administratora na bazi podataka opisan u nastavku.

Konačno, jedini način zaštite od ovakvog napadača jest šifriranje podataka na razini atributa neovisno koristi li se ugrađeni Postgresov modul *pgcrypto* ili klijentsko šifriranje.

4.1.3 Napadač administrator na bazi podataka

U ovom slučaju razmatra se napadač koji je administrator baze podataka. Pretpostavka je da ima pristup svim podacima u bazi i da ih može čitati i modificirati.

U slučaju ovakvog napadača čak i korištenje metoda za šifriranje koje nudi modul *pgcrypto* može biti nedovoljno za zaštitu podataka. Razlog tomu je što klijentska aplikacija pri šifriranju koristeći *pgcrypto* na poslužiteljsku bazu podataka šalje ključ i podatke koji se trebaju šifrirati. Konkretno, to znači da postoji kratki trenutak kad napadač, koji je administrator baze, može podatke presresti. Presretanje, odnosno otkrivanje podataka poslanih od strane klijenta, napadač može izvršiti na razne načine. Najjednostavniji pristup otkrivanju tih podataka bilo bi uključivanje bilježenja svih upita u dnevnik (engl. *log*). Napadač koji je

administrator bi zatim mogao iz zapisa u dnevniku pročitati cjelovite klijentske upite što bi u konačnici omogućilo da pročita ključeve i podatke koji se šifriraju.

Jedini ispravni način zaštite od takve vrste napadača bilo bi klijentsko šifriranje. Klijent bi šifrirao podatke na svom računalu, a tek ih potom slao na poslužiteljsku bazu podataka.

4.2. Analiza napadača na komunikacijskom putu

U ovom poglavlju cilj je analizirati određene aspekte napadača koji se nalazi na komunikacijskom putu između klijentske aplikacije i poslužiteljske baze podataka. Konkretno, cilj je otkriti koliko precizno napadač može iz veličine paketa koji se šalju mrežom otkriti koliko zapisa je vraćeno kao odgovor na pojedini upit.

4.2.1 Postavljanje eksperimenta

Za početak potrebno je podesiti alat za analizu mrežnog prometa. Za potrebe ovog eksperimenta korišten je alat Wireshark. Upiti su slani lokalno, odnosno klijentska aplikacija i poslužiteljska baza podataka nalazile su se na istom računalu. Upravo zbog toga što se sva komunikacija odvijala lokalno, Wireshark je bio podešen tako da prati promet na sučelju *lo0,* odnosno na *loopback* sučelju.

Alat Wireshark nudi opciju filtriranja paketa. Veza između klijenta i poslužitelja šifrirana je protokolom TLS, stoga spomenuti filtar treba postaviti tako da se prate samo *tls* paketi. Nadalje, pretpostavljeni port koji Postgres koristi je TCP port 5432.

Upiti se šalju koristeći biblioteku *psycopg2* za programski jezik Python.

Što se tiče konfiguracije baze podataka poslužitelja, stvaraju se dvije tablice, jedna nešifrirana imena *test_unencrypted* i druga, šifrirana, imena *test_encrypted*. Obje tablice sadrže polja jednakih imena: *id*, *username* i *user_salary*. Jedina razlika je što su polja *username* i *user_salary* tablice *test_unencrypted* po vrsti podataka znakovni niz i cijeli broj, dok su ista polja u tablici *test_encrypted* vrsta podataka *bytea*. Polja *id* u obje tablice vrste su *serial*, što je učinjeno zbog lakšeg postavljanja upita. U obje tablice ubačeno je 6 istih redaka, a jedina

razlika je što su u tablici *test_encrypted* ti podaci šifrirani. Šifriranje je provedeno korištenjem modula *pgcrypto* koristeći algoritam AES-128.

U nastavku, u ispisu 4.6. prikazan je pokazni primjer kako je stvorena veza između klijenta i baze podataka u Pythonu [32]. U primjeru se prvo može vidjeti kako se stvara veza s bazom podataka, a zatim kako se postavlja upit pretraživanja. Sve je implementirano unutar *try/except/finally* bloka gdje je osigurano da se veza s bazom podataka uvijek pravilno zatvori.

```
import psycopg2

try:

     connection = psycopg2.connect(user = "postgres", password = "postgres", host = "localhost", port = "5432", database = "postgres")

     cursor = connection.cursor()

     cursor.execute("SELECT * FROM test_unencrypted where id=1")

     record = cursor.fetchone()

     print(record)

except(Exception, psycopg2.Error) as error:

     print("Greška pri spajanju na Postgres bazu.",error)

     connection = None

finally:

     if(connection != None):

          cursor.close()

          connection.close()

          print("Veza sa Postgres bazom je zatvorena.")
```

Ispis 4.6. Stvaranje veze s bazom podataka i postavljanje upita pretraživanja u Pythonu pomoću biblioteke *psycopg2*

4.2.2 Provedba analize i rezultati

Za početak je ideja usporediti odgovore na upit za pojedine retke šifrirane i nešifrirane tablice te potom pokušati iz toga izvući zaključke.

Na slici 4.1. mogu se vidjeti podaci koji su zapisani u tablici *test_unencrypted*, dok se na slici 4.2. mogu vidjeti podaci zapisani u tablici *test_encrypted.*

```
1  select * from test_unencrypted
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	username character varying (100)	usersalary integer
1	1	John	1000
2	2	Mark	2000
3	3	Juan	1000
4	4	Jonathan	10000
5	5	Anthony_Joshua	150000
6	6	Wolfgang_Amadeus_Mozart	5000

Slika 4.1. Podaci zapisani u prvoj, nešifriranoj tablici

```
1  select * from test_encrypted
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	username bytea	usersalary bytea
1	1	[binary data]	[binary data]
2	2	[binary data]	[binary data]
3	3	[binary data]	[binary data]
4	4	[binary data]	[binary data]
5	5	[binary data]	[binary data]
6	6	[binary data]	[binary data]

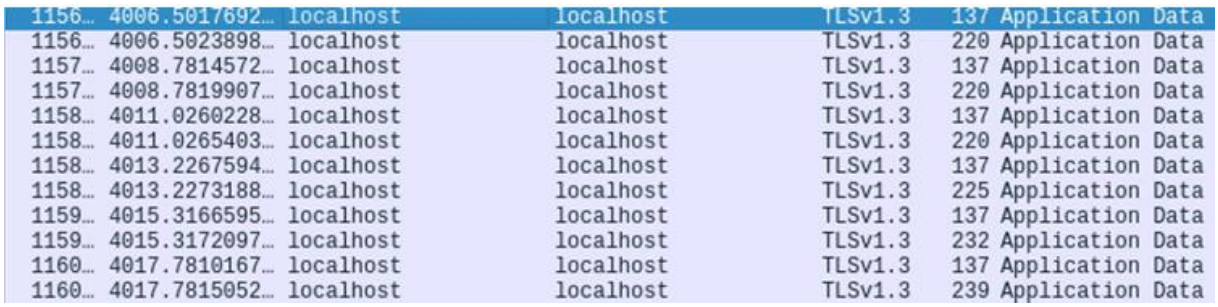
Slika 4.2. Podaci zapisani u drugoj, šifriranoj tablici

U ispisu 4.7. prikazani su upiti bazi iz kojih se pretražuju pojedini reci nešifrirane tablice. Na slici 4.3. mogu se vidjeti paketi koji su rezultat spomenutih upita. Polje koje se promatra na slici 4.3. je predzadnje polje koje predstavlja veličinu paketa. Svi neparni reci na slici 4.3., odnosno svi reci gdje je veličina upita 137, predstavljaju same upite, dok su svi ostali reci odgovori na upit. Za potrebe ovog eksperimenta važno je promatrati samo pakete koji predstavljaju odgovore na upit.

Iz slike 4.3. može se odrediti da odgovori na upite za retke 1, 2 i 3 iz tablice imaju istu veličinu, 220 bajta, jer su vrijednosti polja *username* tih redaka iste veličine, kao i *usersalary* i *id*. Također, odgovori koji sadrže retke 4, 5 i 6 veličine su 225, 232 te 239 bajta, što prvenstveno proizlazi iz toga što imaju veću duljinu polja *username* od prva tri retka.

```
cursor.execute("SELECT * FROM test_unencrypted where id=1")

cursor.execute("SELECT * FROM test_unencrypted where id=2")

cursor.execute("SELECT * FROM test_unencrypted where id=3")

cursor.execute("SELECT * FROM test_unencrypted where id=4")

cursor.execute("SELECT * FROM test_unencrypted where id=5")

cursor.execute("SELECT * FROM test_unencrypted where id=6")
```

Ispis 4.7. Niz upita u kojima se pretražuju pojedini reci nešifrirane tablice



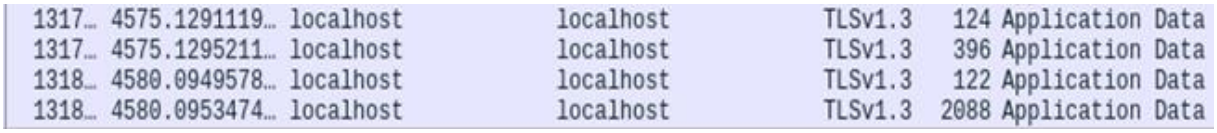
1156…	4006.5017692…	localhost	localhost	TLSv1.3	137	Application Data
1156…	4006.5023898…	localhost	localhost	TLSv1.3	220	Application Data
1157…	4008.7814572…	localhost	localhost	TLSv1.3	137	Application Data
1157…	4008.7819907…	localhost	localhost	TLSv1.3	220	Application Data
1158…	4011.0260228…	localhost	localhost	TLSv1.3	137	Application Data
1158…	4011.0265403…	localhost	localhost	TLSv1.3	220	Application Data
1158…	4013.2267594…	localhost	localhost	TLSv1.3	137	Application Data
1158…	4013.2273188…	localhost	localhost	TLSv1.3	225	Application Data
1159…	4015.3166595…	localhost	localhost	TLSv1.3	137	Application Data
1159…	4015.3172097…	localhost	localhost	TLSv1.3	232	Application Data
1160…	4017.7810167…	localhost	localhost	TLSv1.3	137	Application Data
1160…	4017.7815052…	localhost	localhost	TLSv1.3	239	Application Data

Slika 4.3. Paketi snimljeni u alatu Wireshark kao rezultat upita iz ispisa 4.7.

Iduća stvar je postavljanje upita za pretraživanje svih redaka, prvo iz nešifrirane, a potom iz šifrirane tablice, kao što je prikazano u ispisu 4.8. Rezultat su snimljeni paketi vidljivi na slici 4.4. Odgovor koji sadrži sve retke nešifrirane tablice veličine je 396 bajta, dok je odgovor za šifriranu tablicu veličine 2088 bajta.

```
cursor.execute("SELECT * FROM test_unencrypted")

cursor.execute("SELECT * FROM test_encrypted")
```

Ispis 4.8. Upiti za pretraživanje svih redaka prvo nešifrirane, a potom šifrirane tablice

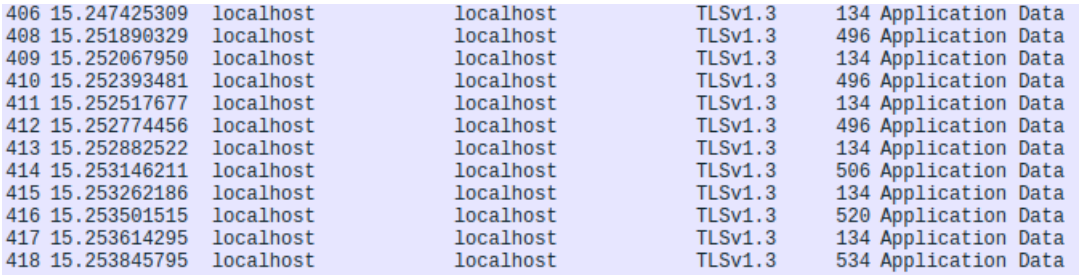


Slika 4.4. Paketi snimljeni u alatu Wireshark kao rezultat upita iz ispisa 4.8.

Zatim se postavlja niz upita vidljiv u ispisu 4.9. kojim se radi pretraživanje pojedinih redaka šifrirane tablice. Rezultat je prikazan na slici 4.5. Kao što je za očekivati, odgovori koji sadrže retke 1, 2 i 3 iz šifrirane tablice su iste veličine, baš kao što su i u nešifriranoj, a ta veličina iznosi 496 bajta. Odgovori koji sadrže retke 4, 5 i 6 šifrirane tablice veličina su 506, 520 i 534 bajta.

```
cursor.execute("SELECT * FROM test_encrypted where id=1")

cursor.execute("SELECT * FROM test_encrypted where id=2")

cursor.execute("SELECT * FROM test_encrypted where id=3")

cursor.execute("SELECT * FROM test_encrypted where id=4")

cursor.execute("SELECT * FROM test_encrypted where id=5")

cursor.execute("SELECT * FROM test_encrypted where id=6")
```

Ispis 4.9. Niz upita kojima se pretražuju pojedini reci šifrirane tablice



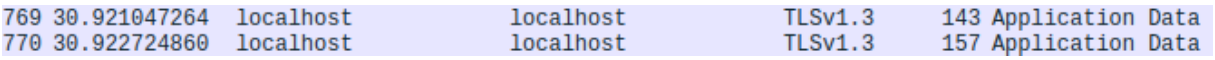
406	15.247425309	localhost	localhost	TLSv1.3	134 Application Data
408	15.251890329	localhost	localhost	TLSv1.3	496 Application Data
409	15.252067950	localhost	localhost	TLSv1.3	134 Application Data
410	15.252393481	localhost	localhost	TLSv1.3	496 Application Data
411	15.252517677	localhost	localhost	TLSv1.3	134 Application Data
412	15.252774456	localhost	localhost	TLSv1.3	496 Application Data
413	15.252882522	localhost	localhost	TLSv1.3	134 Application Data
414	15.253146211	localhost	localhost	TLSv1.3	506 Application Data
415	15.253262186	localhost	localhost	TLSv1.3	134 Application Data
416	15.253501515	localhost	localhost	TLSv1.3	520 Application Data
417	15.253614295	localhost	localhost	TLSv1.3	134 Application Data
418	15.253845795	localhost	localhost	TLSv1.3	534 Application Data

Slika 4.5. Paketi snimljeni u alatu Wireshark kao rezultat upita iz ispisa 4.9.

Iz rezultata prikazanih do sad uočavaju se određene poteškoće pri određivanju neke konkretne formule koja bi pokazala koliko je redaka vraćeno kao odgovor na upit. Radi se o tome da veličina odgovora može poprilično varirati ovisno o tome vraćaju li se kao odgovor cijeli reci ili samo dijelovi redaka. To je ukratko demonstrirano postavljanjem upita iz ispisa 4.10. u kojem se iz tablice dohvaća samo polje *username* i to za prvi redak. Rezultat je prikazan na slici 4.6. gdje se vidi da je odgovor na taj upit veličine 157 bajta. Za usporedbu, odgovor na upit za cijeli taj redak bio je veličine 220 bajta.

```
cursor.execute("SELECT username FROM test_unencrypted where id=1")
```

Ispis 4.10. Upit za pretraživanje polja *username* iz prvog retka nešifrirane tablice



769	30.921047264	localhost	localhost	TLSv1.3	143 Application Data
770	30.922724860	localhost	localhost	TLSv1.3	157 Application Data

Slika 4.6. Paketi snimljeni u alatu Wireshark kao rezultat upita iz ispisa 4.10.

Nadalje, reci mogu međusobno varirati po veličini. To je vidljivo, primjerice u testu kad su traženi svi reci šifrirane baze. Odgovor koji sadrži prvi redak bio je veličine 496 bajta, dok je odgovor za šesti redak bio veličine 534 bajta. Takve varijacije u veličini bi još više došle do izražaja kod većih tablica gdje spremljeni podaci mogu znatno varirati po veličini.

Iz prethodno navedenih razloga, u ostatku eksperimenta bit će naglasak na otkrivanje koliko je redaka vraćeno kao odgovor na upit za nešto specifičniji slučaj. Razmotrit će se slučaj kad se upiti postavljaju nad vrijednostima koje imaju malu entropiju duljine. Pod time se misli na vrijednosti koje se ne razlikuju puno u duljini, odnosno u idealnom slučaju su sve iste duljine.

Važno je napomenuti da se može računati i s prosječnom duljinom. Primjerice, u slučaju kad se za neki upit zna da kao rezultat vraća N redaka, moguće je izračunati kolika je prosječna

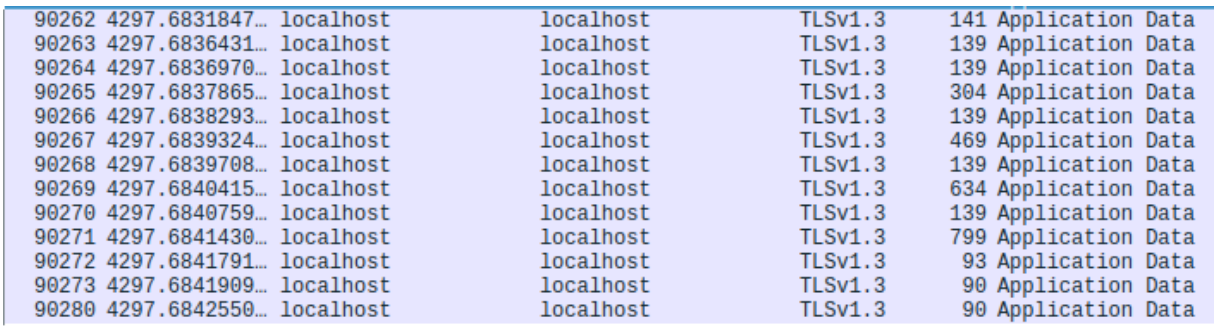
duljina jednog retka na način da se veličina snimljenog paketa podijeli sa N. Pomoću izračunate prosječne duljine retka može se procijeniti koliko je redaka vraćeno u nekom drugom upitu. Rezultati dobiveni tim načinom računanja mogu varirati ovisno o tome kakva je entropija duljine. Ako se vrijednosti spremljene u tablici ne razlikuju previše u duljini dobiveni broj redaka bit će točniji, a ako se vrijednosti spremljene u tablici znatno razlikuju u duljini moguće je da će dobiveni broj redaka biti pogrešan.

U nastavku eksperimenta će se razmatrati nova tablica koja kao i prethodne dvije sadrži polje *id*. Drugo polje tablice predstavlja šifrirani studentski JMBAG (Jedinstveni Matični Broj Akademskog Građanina) spremljen u polju *jmbag*. Ključno svojstvo JMBAG vrijednosti je da je uvijek iste duljine koja iznosi 10 znakova. Kao i u prethodnom eksperimentu, šifriranje je izvršeno algoritmom AES-128. Tablica, imena *test_encrypted2* sadrži 4 zapisa.

Postavlja se niz upita prikazan u ispisu 4.11. kojima se dohvaćaju vrijednosti polja *jmbag*. Rezultat je prikazan na slici 4.7.

```
cursor.execute("SELECT jmbag FROM test_encrypted2 where id=-1")

cursor.execute("SELECT jmbag FROM test_encrypted2 where id=1")

cursor.execute("SELECT jmbag FROM test_encrypted2 where id<3")

cursor.execute("SELECT jmbag FROM test_encrypted2 where id<4")

cursor.execute("SELECT jmbag FROM test_encrypted2 where id<5")
```

Ispis 4.11. Upit za pretraživanje polja *jmbag* u raznim slučajevima iz tablice *test_encrypted2*



Slika 4.7. Paketi snimljeni u alatu Wireshark kao rezultat upisa iz ispisa 4.11.

Veličine odgovora na upit u bajtovima sortirane uzlazno su sljedeće: 139, 304, 469, 634, 799. Može se primijetiti da je razlika između svake dvije susjedne veličine točno 165. Iz te pravilnosti u ovom slučaju može se pročitati volumen odgovora uz određene pretpostavke.

Ako se pretpostavi da je dobivena razlika od 165 bajtova veličina jednog retka koji je vraćen u odgovoru, te ako se pretpostavi da svaki od tih odgovora sadrži isti broj vraćenih polja, može se saznati koliko je redaka vraćeno kao odgovor na upit. Do traženog volumena može se doći tako da se podijeli veličina pojedinog paketa sa spomenutom razlikom.

Dakle, ako se veličina prvog paketa, odnosno 139 bajta podijeli sa 165, dolazi se do brojke 0.8424. Dobiveni broj treba se zaokružiti na najveći cijeli broj, što bi u ovom slučaju bilo 0. To znači da za prvi upit nije vraćen nijedan redak. Nadalje, ako se veličine ostalih paketa, odnosno redom 304, 469, 634 i 799 podijele sa 165 dobivaju se brojke 1.8424, 2.8424, 3.8424 i 4.8424 koje kada se zaokruže na najveći cijeli iznose 1, 2, 3 i 4. To znači da je u odgovoru na drugi upit iz ispisa 4.15. vraćen jedan redak, u odgovoru na treći upit vraćena su dva retka, u odgovoru na četvrti upit vraćena su tri retka i konačno u odgovoru na peti upit vraćena su sva četiri retka iz tablice.

Rezultat dobiven u prethodnom eksperimentu pokazuje da se u specifičnom slučaju kad su šifrirana polja jednake duljine može doći do točnog broja redaka vraćenih kao odgovor na upit. Do približnog broja redaka moglo bi se i doći ako su vrijednosti otprilike približne duljine računajući s prosječnom duljinom.

Za napadače na bazu podataka to bi značilo da moraju znati neke detalje o bazi podataka koju napadaju, primjerice koje vrste su pojedini atributa u tablicama.

5. Zaključak

Stvaranje sigurne baze podataka, uz ispunjavanje klijentskih zahtjeva, složen je posao. Model prijetnji neprestano je potrebno ažurirati. U ovom radu predstavljen je generalizirani model prijetnje za šifriranja u bazama podataka. Na primjeru baze podataka Postgres pokazano je kako šifriranje funkcionira i kako ga je moguće uključiti u postojeću Postgres bazu podataka. No, unatoč mnogim prednostima, treba biti svjestan i novih poteškoća koje se pojavljuju uvođenjem šifriranja. Efikasno izvođenje upita nad šifriranim podacima te pohrana ključeva izazov su za sve arhitekte baza podataka. Iz primjera aktualnih pokušaja rješenja problema efikasnog izvođenja upita nad šifriranim podacima zaključuje se da svi postojeći sustavi imaju manjkavosti.

Nadalje, pokazano je koliko napadači koje se nalaze na bazi podataka i komunikacijskom putu mogu saznati informacija. Iz provedenog eksperimenta u kojem se pokušava doznati koliko je redaka vraćeno na odgovor kao upit može se zaključiti da podaci nisu apsolutno sigurni čak ni kada se prenose sigurnim protokolom kao što je TLS. Ako napadači saznaju detalje o strukturi baze podataka i pripadnim tablicama postoji mogućnost da dovoljno precizno odrede koliko se redaka vraća kao odgovor na upit, pogotovo ako i sami mogu postavljati upite. Konačno, još je potrebno istražiti kako napadači mogu rekonstruirati podatke iz baze koristeći samo volumen odgovora.

6. Literatura

[1] Matt Yonkovit, „The State of the Open Source Database Industry in 2020: Part Three", 22. travnja 2020., Dostupno na: https://www.percona.com/blog/2020/04/22/the-state-of-the-open-source-database-industry-in-2020-part-three/ [Pristupljeno 7. lipnja 2020.]

[2] Matt Asay, „The era of the cloud database has finally begun", 11. svibnja 2018., Dostupno na: https://www.infoworld.com/article/3271128/the-era-of-the-cloud-database-has-finally-begun.html [Pristupljeno 7. lipnja 2020.]

[3] Nate Lord, „What is Data Encryption? Definition, Best Practices & More", 15. srpnja 2019. Dostupno na: https://digitalguardian.com/blog/what-data-encryption [Pristupljeno 24. svibnja 2020.]

[4] Bud Walder, „Best Practices: Securing Data at Rest, in Use, and in Motion", 12. prosinca 2015. Dostupno na: https://www.datamotion.com/best_practices_-securing_data_at_rest_in-use_and_in_motion/ [Pristupljeno 24. svibnja 2020.]

[5] Članak s Wikipedije, „Encryption": https://en.wikipedia.org/wiki/Encryption [Pristupljeno 24. svibnja 2020.]

[6] Terry Chia, „Confidentiality, Integrity, Availability: The three components of the CIA Triad", 20. kolovoza 2012. Dostupno na : https://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/ [Pristupljeno 24. svibnja 2020.]

[7] Članak s Wikipedije, „Symmetric-key algorithm": https://en.wikipedia.org/wiki/Symmetric-key_algorithm [Pristupljeno 24. svibnja 2020.]

[8] Ravi Das, „A Review of Asymmetric Cryptography", Dostupno na: https://resources.infosecinstitute.com/review-asymmetric-cryptography/ [Pristupljeno 24. svibnja 2020.]

[9] Članak s Wikipedije, "RSA (cryptosystem)", Dostupno na: https://en.wikipedia.org/wiki/RSA_(cryptosystem) [Pristupljeno 24. svibnja 2020.]

[10] Članak s Wikipedije, „Hybrid cryptosystem", Dostupno na: https://en.wikipedia.org/wiki/Hybrid_cryptosystem [Pristupljeno 24. svibnja 2020.]

[11] Rick Mogull, Adrian Lane, „Understanding and Selecting a Database Encryption or Tokenization Solution", Dostupno na: https://securosis.com/assets/library/reports/Securosis_Understanding_DBEncryption.V_.1_.pdf [Pristupljeno 24. svibnja 2020.]

[12] Antwanye Ford, Latia Hutchinson, „Full disk encryption: do we need it?“, 16. siječnja 2018., Dostupno na: https://www.csoonline.com/article/3247707/full-disk-encryption-do-we-need-it.html [Pristupljeno 24. svibnja 2020.]

[13] Marc Linster, „Creating a multi-layered security architecture for your databases“, 22. listopada 2019., Dostupno na: https://www.itopstimes.com/itsec/creating-a-multi-layered-security-architecture-for-your-databases/ [Pristupljeno 12. lipnja 2020.]

[14] Članak s Wikipedije, „Transparent data encryption“, Dostupno na: https://en.wikipedia.org/wiki/Transparent_data_encryption [Pristupljeno 24. svibnja 2020.]

[15] Članak s Wikipedije, „Column level encryption“, Dostupno na: https://en.wikipedia.org/wiki/Column_level_encryption [Pristupljeno 24. svibnja 2020.]

[16] Jeremy Stieglitz, „Encryption: Pros and Cons“, 12. rujna 2017., Dostupno na: https://www.imperva.com/blog/encryption-pros-and-cons/ [Pristupljeno 24. svibnja 2020.]

[17] Daniela Oliveira, Anna Squicciarini, Dan Lin, "Cloud Security Baselines from: Cloud Computing Security, Foundations and Challenges", 9. rujna 2016., Dostupno na: https://www.routledgehandbooks.com/doi/10.1201/9781315372112-5 [Pristupljeno 4. lipnja 2020.]

[18] Marie-Sarah Lacharité, „Breaking Encrypted Databases: Generic Attacks on Range Queries“, kolovoz 2019., Dostupno na: https://i.blackhat.com/USA-19/Thursday/us-19-Lacharite-Breaking-Encrypted-Databases-Generic-Attacks-On-Range-Queries-wp.pdf [Pristupljeno 24. svibnja 2020.]

[19] Dmytro Bogatov, George Kollios, Leonid Reyzin, „A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols“, Dostupno na: https://eprint.iacr.org/2018/953.pdf [Pristupljeno 7. lipnja 2020.]

[20] Scott Arciszewski, „Building Searchable Encrypted Databases with PHP and SQL“, 25. svibnja 2017., Dostupno na: https://paragonie.com/blog/2017/05/building-searchable-encrypted-databases-with-php-and-sql [Pristupljeno 24. svibnja 2020.]

[21] Stručni članak, „Application Threat Modelling“, Dostupno na: https://owasp.org/www-community/Application_Threat_Modeling [Pristupljeno 24. svibnja 2020.]

[22] Stručni rad, Centar Informacijske Sigurnosti „Modeliranje sigurnosnih prijetnji (Threat modeling)“, Dostupno na: https://www.cis.hr/files/dokumenti/CIS-DOC-2012-05-049.pdf [Pristupljeno 24. svibnja 2020.]

[23] Članak s Wikipedije, „STRIDE (security)“, Dostupno na: https://en.wikipedia.org/wiki/STRIDE_(security) [Pristupljeno 24. svibnja 2020.]

[24] Stručni članak, Microsoft, „Threat Modeling“, Dostupno na: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff921345(v=pandp.10) [Pristupljeno 24. svibnja 2020.]

[25] Nawaz Ahmed, „Let's get back to basics – PostgreSQL Memory Components“, 30. siječnja 2018., Dostupno na: https://www.postgresql.fastware.com/blog/back-to-basics-with-postgresql-memory-components [Pristupljeno 24. svibnja 2020.]

[26] Kaarel Moppel, „PostgreSQL: Writer and Wal Writer Processes Explained“, 25. studeni 2016., Dostupno na: https://www.cybertec-postgresql.com/en/postgresql-writer-and-wal-writer-processes-explained/ [Pristupljeno 24. svibnja 2020.]

[27] Gary Evans, „How PostgreSQL maps your tables into physical files“, 29. kolovoza 2019., Dostupno na: https://www.postgresql.fastware.com/blog/how-postgresql-maps-your-tables-into-physical-files [Pristupljeno 24. svibnja 2020.]

[28] Postgres dokumentacija, „Encryption Options“, Dostupno na: https://www.postgresql.org/docs/12/encryption-options.html [Pristupljeno 24. svibnja 2020.]

[29] Postgres dokumentacija, „pgcrypto“, Dostupno na: https://www.postgresql.org/docs/12/pgcrypto.html [Pristupljeno 24. svibnja 2020.]

[30] OpenPGP službena stranica, "Standard", Dostupno na: https://www.openpgp.org/about/standard/ [Pristupljeno 24. svibnja 2020.]

[31] Postgres dokumentacija, „Schemas“, Dostupno na: https://www.postgresql.org/docs/12/ddl-schemas.html [Pristupljeno 24. svibnja 2020.]

[32] Online tehnička stranica, „Python PostgreSQL Tutorial Using Psycopg2“, 26. travnja 2020., Dostupno na: https://www.postgresql.org/docs/12/pgcrypto.html [Pristupljeno 24. svibnja 2020.]

Sažetak

Naglasak ovog rada je šifriranje u bazama podataka. Definirani su osnovni pojmovi šifriranja podataka i šifriranja u bazama podataka te je definiran njihov model prijetnje. Nadalje, opisano je šifriranje u bazi podataka Postgres, dan je pregled arhitekture Postgresa, dostupne metode šifriranja i primjerom je pokazano kako implementirati šifriranje u Postgresu. Zadnji dio rada bavi se temom koliko napadači na bazi podataka i na komunikacijskom putu mogu saznati informacija. Konačno, proveden je eksperiment kojem je cilj utvrditi koliko precizno se može otkriti koliko je redaka iz baze vraćeno kao odgovor na upit.