Debreceni Egyetem
Informatikai Kar

ADATBÁZISOK BIZTONSÁGI KÉRDÉSEI

Témavezető:
Kollár Lajos
egyetemi tanársegéd

Készítette:
Kozell József
programtervező informatikus
BSc

Debrecen
2011

Tartalomjegyzék

Táblázatok jegyzéke

1. Bevezetés

Amit három ember tud, az már nem titok.

Az internet csordultig van különböző módszerekkel, melyekkel szinte minden program bizonyos funkcióit meg lehet kerülni, különösen a licencelés és autentikáció témakörében. A legtöbb módszer minimális jogosultságokkal – esetenként távoli eléréssel – képes megkerülni egy-egy ilyen funkcionalitást, melyet időigényes elemzéssel és fondorlatos megoldásokkal hajtanak végre.

Kevés szó esik azonban az olyan helyzetekről, amikor olyasvalaki van adatközelben, akinek pont az (lenne) a dolga, hogy a fondorlatos technikákat más, ugyancsak fondorlatos technikákkal távol tartsa a rendszertől. Mi van akkor, ha az amúgy bizalmi állásban levő rendszergazdánk (`root`), adatbázis-adminisztrátorunk (`DBA`), biztonsági megbízottunk[1] úgy gondolja, hogy a védendő adatokhoz hozzá akar férni? A probléma súlyos, hiszen pont ők vannak abban a helyzetben, ahol jogosultságot tudnak önmaguknak adni – vagy eleve rendelkeznek vele –, hogy ezen adatokhoz akár az alkalmazás, akár az operációs rendszer szintjén hozzáférjenek. Jelen dolgozat az adatok alkalmazás- és tárolásszintű védelmét vizsgálja a rendszergazda[2] szemszögéből, majd megvizsgálja, milyen feltételeket kell teremteni a lehető legbiztonságosabb adattároláshoz.

Számos vállalkozás kínál webtárhelyet és alkalmazásszervert kapcsolt adatbázis szolgáltatással, ilyenkor minden adatunk egy harmadik fél alkalmazottainak szakértelmére van bízva. Lehet találni önmagában adatbázis-szervert szolgáltatásként, ekkor interneten keresztül csatlakozhatunk hozzá, annak minden veszélyével, ezzel már egy negyedik felet is bevontunk az adatainkhoz hozzáféréssel rendelkezők körébe.

Furcsa módon az elmúlt években–évtizedekben nem terjedt el az adatbázis-kezelők közvetlen támadása (amikor a kommunikációs protokoll hibáit aknázzák ki), melyek így egyrészt kevésbé vannak auditálva biztonsági szempontból, másrészt a védelem hangsúlya ilyenkor eltolódik a felhasználók irányába. Sajnos egyes adatbázis-kezelőkben a jogosultságok kezelési rendszere túlzottan megengedő (lásd pl. a 2.1.2. fejezetet a 6. oldalon), ami kiskapuk lehetősé-

[1] Pl. a grsecurity `secoff` (security officer) szerepe

[2] Fontos megjegyezni, hogy a szerepről és nem a mögötte álló személyről van szó, ugyanis az esetek túlnyomó részében illetéktelen személyek ilyen szintű jogosultságot szerezve tulajdonítanak el adatot, emiatt nem asszociálhatunk azonnal a rendszergazda személyére.

gét hordozza magában. Ennek egyik fő oka az, hogy senki sem exponál (tudatosan) adatbázis-kezelőt az internetre. Az adatbázisok általában webes adminisztráló programokon keresztül érhetőek el[3], így a külső támadóknak érdemesebb ezekre fókuszálniuk. Legtöbb esetben az adatbázisok tartalmához az ilyen hibásan megtervezett alkalmazásokon keresztül jutnak el a támadók. Ennek kivédése a tervezők, fejlesztők, és adminisztrátorok közös érdeke és felelőssége.

Összegyűjtve azokat a kérdéseket, melyekre adott válaszok világosan megmutatják, hogy hogyan határozza meg az alkalmazás, hogy ki férhet hozzá az adatok tetszőlegesen megadott részhalmazához, a következő témaköröket kaptam:

1. Milyen általános biztonsági megoldásokat használ az adatbázis-kezelő?
   (a) Hogyan különbözteti meg a felhasználókat?
   (b) Hogyan állapítja meg a felhasználók jogait?
   (c) Milyen részletességgel lehet megadni a felhasználó jogait?
2. Milyen lépéseket tesz a jogosultságok megkerülése ellen?
3. Milyen lépéseket tesz a kliens-szerver kapcsolat lehallgatása ellen?
4. Milyen titkosított tárolási eszközök állnak rendelkezésre?

1.1. A vizsgált adatbázis-kezelők

Jelen dolgozatban a két legnépszerűbb (ráadásul nyílt forrású) relációs adatbázis-kezelő rendszernek és a legelterjedtebb zárt forráskódú, kereskedelmi rendszer ingyenes próbaváltozatának[4] teszem fel kérdéseimet.
Eme adatbázis-kezelők az alábbiak:

- MySQL 5.1 [MyS08]
- PostgreSQL 9.0 [Gro10]
- Oracle 10g XE [Ora06b]

[3] Például phpmyadmin, phppgadmin
[4] Az XE változat erőforrás-limitált.

1.1.1. MySQL 5.1

A MySQL[5] fejlesztése 1995-ben az mSQL nevű adatbázis-kezelő alapján indult, majd a fejlesztők elhatározásából az alapoktól kezdve készítettek egy relációs adatbázis-kezelő rendszert, melynek API-ját az mSQL hasonlatosságára építették fel, jelentősen megkönnyítve különböző alkalmazások mSQL-ről MySQL-re portolását. Eme dolgozat írása alatt a Sun Microsystems felvásárolta a MySQL AB-t, majd az Oracle Corporation a Sun-t, így az Oracle kezébe került a MySQL sorsa.

A fejlesztők az egyszerűséget és gyorsaságot tartják szem előtt, ami miatt sok kritika éri a MySQL-t, főleg a más adatbázis-kezelő mellett elkötelezettektől. A legtöbben bizonyos – ritkán használt – funkcionalitásokat hiányolnak belőle, melyeket nemrég már elkezdtek implementálni, de az átlagos relációs adatbázis felhasználó igényeit ezek nélkül is teljességgel lefedi, s emellett csak kevés, ritkábban használt funkcionalitásra fordít erőforrást, hogy ne vonja el a fejlesztőket a valóban fontos feladatoktól.

Annak ellenére, hogy az 5.1.51-es verzió a jelenlegi éles használatra ajánlott, és már várható az 5.5 megjelenése, igen sok – főleg kis – cég van, amely még mindig nem frissített a 4.0-4.1 verziókról az elmúlt 6-8 évben.

1.1.2. PostgreSQL 9.0

A PostgreSQL fejlesztése 1985-ben kezdődött Postgres[6] néven. Eredeti fejlesztője az Ingres relációs adatbázis-kezelő rendszere alapján – annak hibáit kijavítandó – kezdett bele egy újfajta adatbázis-kezelő készítésébe. A Postgres volt az első olyan adatbázis-kezelő, mely támogatta – például – az adattípusok használatát. Rengeteg funkcionalitást építettek bele, többek között nagyon fejlett általános replikációs alrendszere van (Slony-I), mely tartalmaz failover és kaszkád replikációs eszközöket is.

Ugyancsak a PostgreSQL egyik jellemzője, hogy nem egyetlen stabil, folyamatosan fejlesztett verziója van, hanem egyszerre az öt legutóbbi főverzió[7] aktív. (Igaz, a legtöbb alprojekt csak a mindenkori legújabb verzióhoz fejleszt új funkcionalitást.)

Jelenleg a PostgreSQL egy teljes értékű objektumrelációs adatbázis-kezelő rendszer, mely minimális migrációs befektetéssel kompatibilis az Oracle adatbázis-kezelő rendszerével.

[5] A `My` előtag az egyik alapító-fejlesztő lányának keresztneve.

[6] **Post** In**gres**

[7] Jelenleg a 9.0, 8.4, 8.3, 8.2, 8.1 kiadásokat fejlesztik párhuzamosan.

1.1.3. Oracle Database 10g XE

Az Oracle relációs adatbázis-kezelő rendszer első[8] verzióját 1979-ben adták ki, az első tranzakciókat is támogató verziója 1983-ban jelent meg.

Számos területen végzett úttörő fejlesztéseket, például az Oracle volt az első adatbázis-kezelő, mely SQL-t képes volt értelmezni[9], több processzort ki tudott használni, 64-bites rendszereken is natívan futott, s XML-t támogatott, emellett elsőként volt képes több szervert összehangolni elosztott adatbázist képezve (RAC).

Az általam vizsgált Express Edition ingyenesen, erőforrásokat erősen korlátozottan használó verzió, felhasználási megkötések nélkül. Sajnos a teljes értékű verzió ára nem teszi lehetővé kis- és induló vállalkozások számára Oracle használatát, a belépő szintű verzió éves költségei processzoronként kb. 5000 USA dollártól indulnak; ingyenesen csak nem kereskedelmi felhasználásra engedélyezett.

[8] Valójában ennek a neve `Oracle V2` volt, marketing megfontolásokból: „Who'd buy a version 1.0 from four guys in California?" – Larry Ellison, Oracle alapító, arroganciája szakmai berkekben hírhedt.

[9] Korábban, mint az ANSI szabvány, így az Oracle SQL értelmezése eltér az ANSI SQL-étől.

2. MySQL

Ez a fejezet a Debian GNU/Linux disztribúcióban található „MySQL 5.1.49-1 (Debian)” verziót mutatja be[Kof05, MyS06, Vas09].

A MySQL, az egyszerű telepíthetőség és használhatóság jegyében nem integrál más azonosítási rendszereket, így nem lehetséges PAM[10], Kerberos, vagy LDAP/Active Directory forrásokból MySQL felhasználókat közvetlenül azonosítani. Van ilyen célú törekvés, de 2006 óta még mindig tervezési fázisban van, alacsony prioritási besorolással.

Mi a feladata a jogosultságkezelő rendszernek?

A kapcsolódó felhasználót társítja az adatbázison privilégiumokkal (amennyiben lehetséges).

2.1. Milyen biztonsági megoldásokat használ?

A MySQL privilégiumai az alábbi osztályokba sorolhatók:

1. Adminisztratív
2. Adatbázis specifikus
3. Adatbázis-objektum specifikus (pl. tábla, nézet, index, tárolt eljárás, . . .)

2.1.1. Hogyan különbözteti meg a felhasználókat?

A fióknevek a felhasználónévből és a gépnévből állnak. Így különböző személyek azonos felhasználónevet használhatnak, amelyekkel egymástól különböző gépekről kapcsolódhatnak, mert a gépnév is része a fióknévhez kapcsolásnak, de ez ellenjavallt.

Amikor kapcsolódunk a szerverhez, a jogosultságkezelő a kapcsolat kezdeményezőjének címe és az átadott felhasználónév alapján azonosít minket, és amennyiben szükséges, jelszót is kér. Az azonosító adatokat a `mysql` nevű adatbázis különböző tábláiból szedi össze.

Első lépésként a jogosultságkezelő összeszedi a `mysql.user` táblából, hogy a kliens címéről kapcsolódhat-e bárki is. Létezik egy speciális jel, a `%` (százalékjel), amely bármelyik címről engedélyezi a kapcsolódást, illetve ez használható wildcardnak aldomainek maszkolására is. Ezután megnézi, hogy a kliens adott-e át felhasználói nevet. Ha nem, akkor megnézi,

[10] Pluggable Authentication Module; egységesített azonosítást tesz lehetővé

hogy van-e anonim felhasználói fiók az adott domainről vagy globálisan, ha igen, akkor megnézi, van e ilyen nevű fiók. Találat esetén megnézi, hogy van-e jelszó az adott fiókhoz, mely azért lényeges, mert ha egy fióknak nincs jelszava, akkor oda csak úgy lehet belépni, ha a kliens nem küld jelszót (vagyis üres stringet ad át). Egyezés esetén elkezdi várni a parancsokat, egyébként a szerver megszakítja a kommunikációt.

A kliens címének vizsgálata azért fontos, mert így azonos felhasználói névvel más gépről bejelentkező személyek más felhasználónak számíthatnak, így más jogokat is kaphatnak. Például, ha a „hallgato" felhasználó kapcsolódna a szerverhez a „hallg.inf.unideb.hu" gépről, akkor az 1. táblázatban látottak alapján a 3. sorral egyezne, mert a gépnév teljes egyezést mutat, a bejegyzés anonim, így már csak a jelszót kell ellenőrizni. Viszont ha otthonról kapcsolódik, akkor a 7. sor fog egyezést mutatni, mert specifikus hostnevekkel nincs egyezése, ezért a % sorok illeszthetők csak, s a felhasználónév abban a sorban egyezik. Ismeretlen helyről kapcsolódó ismeretlen felhasználók az utolsó sorral fognak egyezni, ilyen felhasználót nem kötelező létrehozni, de ha van, akkor teljesen privilégiummentesen kell tartani, kivéve, ha kifejezetten igényli a publikus hozzáférést valamilyen alkalmazás.

E megoldás sajátossága az, hogy a „localhost" és a „127.0.0.1" Host értékeket tartalmazó sorok közül mindig az IP-t tartalmazót fogja használni, mert az általános rendezési elv szerint a számok a betűk elé kerülnek, így az IP címek általában mindig a hostnevek elé kerülnek (kivéve persze, ha a hostnév is számmal kezdődik). A MySQL úgy rendezi a táblát, hogy előre kerülnek rendezetten a hostnevek és az IP címek, hátulra a %-kal jelölt, bármit elfogadó sorok, a kettő közé a wildcardot is tartalmazóak. Ezen belül van még egy rendezési szint, ahol a felhasználóneveket is rendezi, úgy, hogy az üresek hátra kerüljenek, a nem üresek pedig előre, rendezetten, lásd az 1. táblázatot.

2.1.2. Milyen részletességgel lehet megadni a felhasználó jogait?

Minden műveletre külön jogosultság adható. Korlátozható a felhasználó által elérhető adatbázisok és táblák listája, illetve táblaszinten megadható, hogy mely oszlopon mely műveletek hajthatók végre[11]. Az adatbázis-kezelő minden lekérdezéshez helyben állapítja meg, hogy elegendő jogosultságunk van-e az adott művelet végrehajtásához. A MySQL öröklődési modellje engedékeny[12], azaz ha magasabb szinten (például az adatbázis vagy szerver szintjén) van egy

[11] Így például megakadályozhatjuk bizonyos oszlopok mezőinek módosítását, miközben megmarad az olvasási jogunk, extrémebb esetekben ugyanez fordítva.

[12] Például, ha globális `DELETE` jogunk van, akkor minden táblából tudunk törölni!

Host	User	Password
127.0.0.1	root	*9093C363D429AB69F514619520D4C5ED899A136C
hallg.inf.unideb.hu	admin	*4ACFE3202A5FF5CF467898FC58AAB1D615029441
hallg.inf.unideb.hu		
localhost	root	
onik.inf.unideb.hu	root	
%.inf.unideb.hu	root	*9093C363D429AB69F514619520D4C5ED899A136C
%	hallgato	*56ED94A2DDE7EB602278C7365156D8C3568897B5
%	root	*9093C363D429AB69F514619520D4C5ED899A136C
%		

1. táblázat. MySQL felhasználói tábla rendezett részlete

műveletre jogunk, akkor azt alacsonyabb szinten nem tudjuk megvonni, ez arra az esetre is vonatkozik, ha explicit módon azon a szinten nem lett megadva a jog. A szerver globális szintjén megadott jogosultságokat `superuser` privilégiumoknak hívják. A MySQL tárgyalt verziójában adható jogosultságok az A függelékben találhatóak, a 36. oldalon.

2.1.3. Hogyan állapítja meg a felhasználók jogait?

Az adatbázis-kezelő minden lekérdezésnél megállapítja, hogy a felhasználó jogosult-e az adott műveletre, így kvázi azonnal életbe léphetnek a felhasználó ténykedésével párhuzamosan történő jogkör módosítások. A jogosultságok a már említett `mysql` nevű adatbázisban vannak. Ezen adatbázis táblái közül számunkra lényegesek a `host`, a `user`, a `db`, a `tables_priv` és a `columns_priv` táblák. A `superuser`-ek kivételével semelyik felhasználó sem férhet hozzá ezen táblákhoz, még olvasásra sem.

A jogosultság megállapításának a menete a következő:

1. Adminisztratív művelet?
   (a) Jogosultság ellenőrzése a `mysql.user` táblában (mert ehhez `superuser` jogosultság kell).
   (b) Ha megvan a jog, a műveletet végrehajtja.

2. Adaton végrehajtott művelet?

   (a) `superuser`, vagy globális jogosultság ellenőrzése. Ha valakinek van joga az egész szerveren ehhez a művelethez, akkor nem kutat tovább az adatbázis-kezelő további jogosultságokért.

   (b) A `db` tábla ellenőrzése, a kapcsolathoz tartozó `host`, `db` és `user` adatok alapján. Ha van, a találat meghatározza az adott adatbázisra vonatkozó jogokat.

   (c) Ha még mindig nincs jogunk a kiadott parancshoz, megnézi a `tables_priv` és a `columns_priv` táblákban, hogy van-e jogunk az adott művelethez.

   (d) Ha itt sincs feltüntetve a szükséges jog, akkor nem hajtja végre a műveletet és hibajelzést ad.

2.2. Milyen lépéseket tesz a jogosultságok megkerülése ellen?

Sajnos tervezési okok miatt elég keveset. Egyik legfontosabb lépés, melyre ügyelni kell, hogy a `mysql` adatbázis csak `superuser` jogosultsággal legyen látható. Általános hiba, hogy a frissen létrehozott felhasználónak globális érvényű DML és DDL privilégiumokat adnak, s a későbbiekben feleslegesen adnak meg pontos jogokat, mert elfelejtik elvenni a globális privilégiumokat. Ilyen esetben megfelelő ismeretekkel rendelkező személy bármilyen műveletet végrehajthat, ráadásul az összes felhasználó hash-elt jelszavát is lemásolhatja, későbbi feltörésre[13]. Rosszabb eset, ha írási joga is van a `mysql` táblához, ez esetben még saját felhasználót is létrehozhat SQL utasításokkal, vagy ideiglenesen más felhasználó jelszavát lecserélheti egy általa ismert forrásból készített hash-sel. Ezért létfontosságú, hogy átlagos felhasználó ne férhessen a `mysql` adatbázishoz, illetve adatvédelmi okokból csak ahhoz az adatbázishoz, táblához, oszlophoz, melyre valóban szüksége van.

A felhasználók megkülönböztetési módszerénél már olvashattuk, hogy mindig a legpontosabb találatot illeszti a felhasználóra az adatbázis-kezelő. Ez megakadályozza, hogy egy feketelistára tett gépről kapcsolódva is ugyanazokat a jogokat kaphassuk, mintha egy megbízható gépről jönnénk.

[13] MySQL 4.1 előtti verziók jelszótárolási formáját egy modern asztali gép kb. 2 másodperc alatt fejti vissza [Ano03]. Emiatt néhány éve algoritmust cseréltek.

2.2.1. Rossz szokások

Nem érdemes trükköznünk azzal, hogy a `root` a saját jogait megnyirbálja. Vagy olyan szinten jogtalanítjuk az adminisztrátort, hogy nem tudja később visszaállítani saját jogait: ekkor a minimális alkalmazás-adminisztrációs teendőit sem tudja ellátni, vagy el tudja, de akkor még vissza tudja állítani saját jogait. Előbbi esetben ráadásul némi rendszerleállásra[14] is szükség van, hogy az adminisztrátor jogait helyreállítsuk...

2.3. Milyen lépéseket tesz a kliens-szerver kapcsolat lehallgatása ellen?

A kapcsolatlehallgatás egyik módszere, amivel jelszót tudunk szerezni, az, amikor egy figyelmetlen felhasználó egy általunk is használt gépről belép a MySQL konzolba, s a jelszavát parancssori argumentumként adja meg. Ez esetben egyrészt a parancssori előzményekbe is bekerül, másrészt a processzlistában is megjelenik a parancs argumentumaként. Ezt a konzol megpróbálja eltüntetni[15], de bizonyos operációs rendszereken ez nem lehetséges.

Hálózaton keresztüli kapcsolódás esetén megvan az esélye annak, hogy a titkosított jelszót lehallgassuk menet közben. Kezdőket könnyen el lehet tántorítani a további próbálkozástól, ha engedélyezzük a kapcsolaton a tömörítést, ez megzavarja a felkészületleneket, de a profiktól csak titkosítással védhetjük meg az adatforgalmunkat.

A MySQL támogatja az adatforgalom SSL[16] kapcsolaton keresztüli titkosított átvitelét, ekkor először az SSL alrendszer felépíti a két gép közötti kapcsolatot, ezután minden MySQL forgalom titkosítottan történik. Ennek a használata szinte teljesen kizárja a sikeres lehallgatás lehetőségét. Mivel az SSL támogatja az aszimmetrikus titkosítási algoritmusokat (lásd az 5. fejezetet), megadhatjuk a publikus kulcsunkat a MySQL fiókunknak, s ezentúl X.509 algoritmusok segítségével jelszó megadása nélkül is kommunikálhatunk (feltéve, ha nálunk van a privát kulcsunk). Az adatfolyam titkosítása a kapcsolat felépítése után teljesen láthatatlan, egyetlen hátulütője a megnövekedett processzorhasználat, mivel az erős titkosítási algoritmusoknak jelentős számítási teljesítményre van szükségük.

[14] Létezik egy parancssori argumentum, amivel rá lehet venni az adatbázis-kezelőt indításkor, hogy ne vegye figyelembe a privilégiumokat tartalmazó táblákat. Ennek használata kifejezetten ön- és közveszélyes.

[15] `6725 pts/2 S+ 0:00 mysql -u root -password=x xxxxxxxx`

[16] Secure Sockets Layer protokoll

2.4. Milyen titkosított tárolási eszközök állnak rendelkezésre?

Jelenleg nincs olyan lehetőség, mellyel elérhetnénk azt, hogy az adat titkosítva tárolódjon a merevlemezen, de a titkosítás kulcsa ne legyen elérhető egy rendszergazdai jogokkal rendelkező felhasználó számára. Egyetlen megoldás, ha az adatot a kliens már titkosítva adja át, s a titkosítás kulcsát nem a szerveren tárolja. Ez esetben a klienst használó alkalmazásnak kell megoldania a küldött adat titkosítását, s az érkező adat visszafejtését. A titkosított tárolás miatt viszont az SQL nyelv előnyeinek számító funkcionalitások (szűrés, rendezés, csoportosítás, aggregát függvények, ...) igen nagy része működésképtelenné válik, mivel a titkosítás lényegéből fakadóan nem tud az adatbázis-kezelő az adatra nézve például rendezési szempontból helytálló jóslatokba bocsátkozni. Félmegoldásként csak az érzékeny adatokat kell titkosítani, s titkosítatlan metaadatokat tárolni róluk, ami alapján kereshetőek, szűrhetőek. A kliensoldali titkosítás hátulütője, hogy azonos szerveren tárolt webalkalmazásokból értelmetlen használni, mivel abba is beépülhet egy lehallgató kód, mellyel elfogható a titkosítás kulcsa.

2.5. Adatbázis-kezelő specifikus esetek

A MySQL támogatja az AES blokk-kódoló titkosítási algoritmust[17], ezzel függvényhívásban tudunk titkosítani, vagy visszafejteni, a kulcs birtokában. Hátulütője, hogy a kulcsot függvényparaméterként át kell adni a szervernek, ahol a lekérdezések figyelésével az elfogható.

Habár többféle adattároló motort támogat és használ is, ezek közül kettőt használnak az adatok lemezen tárolására az esetek túlnyomó részében:

1. MyISAM[18]: Az IBM egy korai fejlesztéséből nőtt ki, nem támogatja a tranzakciókat, direkt olyan egyszerűre tervezték, mint egy faék. Ebből kifolyólag a lemezen tárolt formátuma is meglehetősen egyszerű.

   Statikus formátum esetében minden mező szélessége és, ebből eredően minden sor mérete is, adott – folytonosan feldolgozható a sorméret ismeretében. Dinamikus formátum esetén minden sor fejléce tárolja az aktuális sor méretét, illetve ekkor már töredezhetnek a sorok, megjelölve, hogy hol található a sor fennmaradó része. Mindkét formátum sorfolytonos, egy hexadecimális nézetre képes szövegszerkesztővel akár kézi erővel is egyszerűen kinyerhetők a tárolt adatok az adatfájlból.

[17] Lásd a D. fejezetet a 39. oldalon.

[18] Indexed Sequential Access Method

2. InnoDB: Az Innobase fejlesztette ki (mely később az Oracle tulajdonába került), megbízható de gyors tranzakciós motornak. Az adat fürtözött indexben van eltárolva, így ha az elsődleges indexen megyünk végig, az adat ugyanazon lapon lesz, melyen a hozzá tartozó index is van. Ha nagyon nagy a tábla, akkor fájl I/O-t spórolunk, mert csak egy lapot kell a memóriában tartani az index bejárásához és az adat kinyeréséhez. Ha nincs elsődleges kulcsként használható oszlop, az InnoDB készít egy rejtett oszlopot erre a célra, ugyanis az adatok csak az indexeken keresztül érhetők el, effektíve az indexet tároló B-fa leveleihez vannak kapcsolva. Továbbá a tranzakciók közben verziózza a lapokat, melyek előfordulhatnak magában az adatterületen, de a tranzakciós logban is. Amikor egy lekérdezés adatot kér, a motor megnézi, hogy az adott táblából igényelt lapoknak hol található a legfrissebb hozzáférhető verziója.

3. PostgreSQL

Ez a fejezet a „PostgreSQL 9.0.1-1” verziót mutatja be[DD03, WD02, SM05].

3.1. Milyen biztonsági megoldásokat használ?

3.1.1. Hogyan különbözteti meg a felhasználókat?

A PostgreSQL a 8.1-es verzió óta szerepeket definiál, mely viselkedhet felhasználóként és csoportként is (8.1 előtt ez két külön típus volt). A szerepek a saját tulajdonú objektumok használatát megengedhetik más szerepeknek vagy tagságot adhatnak nekik (ekkor a másik szerep jogosultsági köre kibővül).

A PostgreSQL túllép a MySQL-ben látott egyszerű „név@cím” összerendelésen, és képes jelszó nélküli belépéseket úgy is kezelni, hogy a felhasználó operációs rendszerbeli nevét párosítja egy PostgreSQL szerep nevével. Egy konfigurációs fájl alapján állapítja meg, hogy mely címekről, mely szerepek kapcsolódhatnak az adatbázis-kezelőhöz. Az azonosítást ezen kívül elvégezheti bármely GSSAPI[19] szabványt támogató adatforrásból, SSPI protokollon keresztül Microsoft Windows Active Directory-ból, MIT Kerberos módszerrel (bár az előző kettő módszer ebből fejlődött ki, egy ideje nem támogatja ezt), hagyományos LDAP adatbázisból[20], RADIUS szervereken keresztül, illetve X.509 tanúsítványokkal (ez kényelmes, ha amúgy is SSL titkosított kapcsolatot használunk).

3.1.2. Milyen részletességgel lehet megadni a felhasználó jogait?

Maguk a szerepek privilégiumainak megadása a szerveren belül történik. A létező szerepek megtekintéséhez a `pg_roles` rendszertáblát kell lekérdezni. Jogok módosítását SQL-en keresztül tehetjük meg.

3.1.3. Hogyan állapítja meg a felhasználók jogait?

A szerepekhez rendelt jogok a `postgresql` adatbázis `pg_catalog` rendszersémájában elhelyezett táblákon keresztül vannak adatbázison belülről elérhetővé téve. Ezekhez a rendszertáblákhoz kizárólag a `postgres` felhasználó (az adatbázis-adminisztrátor) fér hozzá, de ő is csak betekintés céljából (ellenben a MySQL-lel, ahol rendes táblaként lekérdezhetőek).

[19] IETF-RFC 2743, Generic Security Services Application Program Interface

[20] Érdekes módon az LDAP adatbázisokat előszeretettel tárolják RDBMS-ekben, főleg MySQL-ben.

Az authentikáció kezdete a `pg_hba.conf` nevű fájl, melyben szabályok adják meg, ki és honnan csatlakozhat, s milyen authentikációs módszert várunk el tőle. Ekkor az adatbázis-kezelő még nem ellenőriz jelszót, pusztán a kapcsolódás elfogadását mérlegeli. Négyféle módon lehet kapcsolódni:

- `local`: csak helyben, Unix Socket-en keresztül, ez a leggyorsabb,
- `hostnossl`: titkosítatlan TCP/IP kapcsolat,
- `hostssl`: SSL-lel titkosított csatornán TCP/IP felett,
- `host`: az előző kettő kombinációja.

Ezek a kapcsolódási jellemzők adják meg, hogy az adott bejegyzés milyen autentikációs módszerre vonatkozik. A MySQL-lel szemben a PostgreSQL nem rendezi ezt a listát, hanem a megadott sorrendben végigmegy a konfigurációs fájlban, s sorba veszi az aktuális kapcsolat típusának megfelelő sorokat az első oszlop alapján. Első találatnál megáll, ha azzal a módszerrel sikertelen, vagy nem talált illeszthető bejegyzést, akkor a kapcsolat megszakad.

A következő oszlop a kapcsolatnak megengedett adatbázis neve. PostgreSQL-ben minden adatbázis külön folyamatban fut, köztük átnyúlni nem lehetséges (hasonlóan az Oracle-höz, ellenben a MySQL-lel, ahol a tábla nevét az adatbázis nevével prefixelve más adatbázisban is lehet dolgozni). Több adatbázis nevét vesszővel elválasztva lehet megadni, melyek mellett az alábbi kulcsszavak használhatóak:

- `all`: az összes adatbázist jelenti,
- `sameuser`: a kért adatbázis neve megegyezik a felhasználóéval,
- `samerole`: a kért adatbázis nevével egyező nevű szerepnek tagja a felhasználó,
- `samegroup`: a `samerole` régi neve de ez elavult, idővel el fog tűnni,
- `replication`: speciális eset, replikációs kapcsolatok használják,
- `@`*fájlnév*: a *fájlnév*ben megadott fájl tartalmazza az engedélyezett adatbázisok nevét, ez alternatívája a vesszővel elválasztott listának.

Jó tudni, hogy ha ezeket a kulcsszavakat idézőjelek közé tesszük, elveszítik speciális jelentésüket, így hivatkozhatunk például a `replication` nevű adatbázisra is (bár ez nem célszerű).

A harmadik oszlop a felhasználónév, ahol az elfogadott felhasználónevet adhatjuk meg, vagy azt a csoportot, melynek tagja a kívánt felhasználó (ekkor + jellel prefixeljük a nevet). Valójában nincs különbség a felhasználók és csoportok között (hisz mindkettő csak szerep). A felhasználó egy olyan szerep, melynek eleve van `LOGIN` (régebben `CONNECT`) privilégiuma, a csoport megadásakor pedig annyit kötünk ki, hogy a megadott szerep bármennyi öröklődési lépésben, de legyen kapcsolatban a felhasználónévvel. Több nevet vesszővel elválasztva lehet megadni, illetve itt az `all` kulcsszó minden névre illeszkedik. Ha az `@` jellel prefixelünk egy fájlnevet, akkor a nevek listája onnan kerül beolvasásra.

A negyedik (és néha az ötödik) oszlop tartalmazza az elfogadott klienscímeket. Kétféleképp lehet megadni a címet: vagy egy IP cím CIDR maszkmérettel, vagy a negyedik oszlopban az IP cím, az ötödikben pedig az alhálózati maszk. Kizárólag IP cím adható meg, hostnév megadására nincs mód. Itt is megadhatók kulcsszavak:

- `samehost`: a szerver összes címe,
- `samenet`: a szerver hálózatának egy alhálózata.

Természetesen IPv6 címek megadására is van lehetőség.
Az ötödik (vagy hatodik) oszlop az authentikáció módját adja meg. Ez lehet:

- `trust`: további ellenőrzés nélkül beenged,
- `reject`: további ellenőrzés nélkül elutasít,
- `md5`: MD5-tel titkosított jelszót kér,
- `password`: titkosítatlan jelszót kér,
- `gss`: GSSAPI-n keresztül azonosít, Single Sign-On (SSO) valósítható meg vele,
- `sspi`: SSPI protokollal, Active Directoryn azonosít, csak Windowson (ezzel is lehet SSO),
- `radius`: RADIUS szervereken keresztül,
- `krb5`: Kerberos szervereken keresztül (elavult, előző három leváltja),

- `ident`: ident protokollon keresztül visszakérdez a kliens gépére, hogy a küldött név egyezik-e a valódival,
- `ldap`: LDAP szervereken keresztüli autentikáció(ezzel is lehet Active Directory-t lekérdezni),
- `cert`: az SSL tanúsítvány elismerése jelenti az azonosítást (a „Common Name” mezőt veszi alapul felhasználónévnek),
- `pam`: PAM-on keresztül, az operációs rendszer által azonosít.

Utolsó oszlopként paramétereket adhatunk át az adott autentikáló ágensnek. Ez mind még csak a kapcsolódás volt. Ha talált illeszkedő bejegyzést, akkor abban a sorban megadott autentikációs módszerrel megpróbálja megállapítani, hogy a kapott felhasználónév és jelszó érvényes-e, ettől függ a kapcsolódás sikere vagy megszakadása.

3.2. Milyen lépéseket tesz a kliens-szerver kapcsolat lehallgatása ellen?

A PostgreSQL támogatja az SSL titkosított adatátvitelt. Konfigurálható egy szerep úgy, hogy kizárólag SSL kapcsolatokat fogadjon el. Ám ellentétben a MySQL-lel, ahol a kliensoldali tanúsítvány csak titkosításra használt, itt a titkosított csatorna létrehozásához szükség van egy, a szerver által is elismert tanúsítványra a kliensprogram oldalán is, mert mind a két végpont azonosítja magát a másik fél felé.

3.3. Milyen titkosított tárolási eszközök állnak rendelkezésre?

A PostgreSQL-hez használható `pgcrypto` csomag többféle titkosító algoritmust nyújt, melyeket függvényként meg lehet hívni SQL-ben, ennek hátulütője, hogy a jelszót is el kell küldeni a szervernek az SQL-be beágyazva (erre a dokumentáció külön felhívja a figyelmet).

3.4. Adatbázis-kezelő specifikus esetek

Jó tudni, hogy minden szerepnél külön elmenthetünk konfigurációs értékeket, melyeket minden sikeres kapcsolódás után beolvas az adatbázis-kezelő.

A PostgreSQL-nek egy adattároló formátuma van. Minden tábla és index fix méretű (általában 8KByte) lapokon helyezkedik el, ezek a lapok logikailag egyenlőek, egy új sor bármelyik

lapra kerülhet. A lapnak 5 fő része van. A lap fejléce általános információkat tárol és hivatkozik, majd egy keresőtábla tartalmaz offset-et minden sorhoz, s ez tárolja, hogy mely tranzakcióhoz tartozik a sor, illetve melyik verzió, mely alapján vissza lehet görgetni arra az állapotra. Ezután szabad terület következik, melyet felülről a keresőtábla, alulról az új sorok töltenek fel (heap). A sorok simán egymás elé vannak pakolva. Az ötödik terület speciális, csak az indexek használják, táblához tartozó lap esetén üres marad.

4. Oracle XE

Habár a jelenlegi legújabb főverzió a „11g Release 2 (11.2.0.2.0)”, ez a fejezet az „Oracle Database 10g Express Edition Release 10.2.0.1.0” verziót mutatja be. Nagyon sok cég még a „8i” főverzióról sem váltott újabb kiadásokra[Lon04].

4.1. Milyen biztonsági megoldásokat használ?

Az Oracle úttörő munkát végzett a relációs adatbázisok fejlesztésének terén, számos témakörben birtokolja az elsőséghez tartozó dicsőséget. Az adatbázis-kezelő minden adatbázishoz külön példányt (instance) futtat. Ezek a példányok külön rendszermemóriával (System Global Area – SGA) és processzmemóriával (Process Global Area – PGA) rendelkeznek, indulás után kizárólagosan használják a hozzájuk tartozó adatbázist. Ez a megoldás a PostgreSQL-hez hasonlatos, egy kompromittálódott példány nem lát át a többi példány területére.

4.1.1. Hogyan különbözteti meg a felhasználókat?

Bejövő kapcsolatok azonosításához nevet és jelszót kér, illetve meg kell adni, mely példányhoz kapcsolódunk. A felhasználónév egyedileg azonosítja a felhasználót. A nevet és a jelszót is nagybetűsre konvertálja, illetve következetesen minden objektum és szerep neve nagybetűs, így a jelszavaknál a kis- illetve nagybetűk használatából eredő kombinációs nyereség elveszik[21].

4.1.2. Hogyan állapítja meg a felhasználók jogait?

A felhasználónévhez társított privilégiumok alapján dönti el az adatbázis-kezelő, hogy a felhasználó jogosult-e a kért műveletre. Egyik ilyen privilégium a „CONNECT”, mely engedélyezi a távoli kapcsolatot az adatbázis-kezelőhöz. Számos privilégium adható meg, melyekkel az adatbázis-kezelő használatának minden aspektusa leírható.

A jelszó ellenőrzése többféleképpen történhet:

- Az adatbázis-kezelőn belül.
- Az operációs rendszer által[22].
- Oracle Advanced Security[23] (OAS) terméken keresztül, amely pl. Kerberos-t használ.

[21] Ez egy átlagos felépítésű jelszó esetén $10^2 \cdot (2 \cdot 26)^8$ helyett csak $10^2 \cdot (26)^8$, ez 3 nagyságrendű gyengülés.

[22] `ALTER USER <username> IDENTIFIED EXTERNALLY`

[23] Csak Enterprise Edition verzió mellé rendelhető kiegészítő.

- SSL-en keresztül, tanúsítvány elfogadásával (Enterprise Manager megléte esetén), ekkor könyvtárszolgáltatás (pl. LDAP vagy Oracle Internet Directory) tartalmazza a felhasználó engedélyeit.
- Proxy, illetve köztes adatbázis réteg által adott felhatalmazás segítségével. Ekkor egy Oracle termék middle-tier alkalmazás (pl. Net8) továbbadja a saját maga által elvégzett autentikáció eredményét, amit az adatbázis-kezelő elfogadhat.

4.1.3. Milyen részletességgel lehet megadni a felhasználó jogait?

Több, mint 100 privilégium írja le a felhasználó lehetőségeit, illetve táblaterenként kvóta is megadható, ezek részletezése túlmutat e dolgozaton. Profilok használatával kiterjedt erőforrás felügyelő kvótarendszer alkalmazható az egyes felhasználók munkamenetén és lekérdezésein. Szerepek és profilok használatával privilégiumcsoportok nevesíthetők, így felhasználói csoportokat könnyebben lehet privilégiumokkal ellátni.

4.2. Milyen lépéseket tesz a jogosultságok megkerülése ellen?

A szerepek külön jelszóval láthatók el, így a felhasználónak újabb jelszót kell megadnia, ha aktivál egy szerepet a munkamenetében. Az auditálás megfelelő beállításával a DBA értesülhet arról, ha valaki sikertelenül próbál szerepekbe belépni, vagy lekérdezéseket, módosításokat végrehajtani. Ellentétben például a MySQL-lel, ahol nincs lehetőség auditra, itt bármely sikertelen próbálkozás a DBA postafiókjában köthet ki.

4.3. Milyen lépéseket tesz a kliens-szerver kapcsolat lehallgatása ellen?

Az Oracle Net Services szolgáltatás (része az Oracle Advanced Security terméknek) lehetővé teszi, hogy a kliens és a szerver között SSL titkosított kapcsolat jöjjön létre. A titkosítatlan protokoll önmagában ellenőrzőösszegekkel védekezik a csomagok módosítása ellen, illetve azonosítókkal jelöli a csomagokat, hogy követhetőek legyenek, s érzékeli, ha egy csomagot újra elküld valaki. A titkosított kapcsolat felépítéséhez kliens- és szerveroldali konfigurációk beállítása, átírása szükséges. A Net8 kapcsolatkezelő middleware is képes transzparensen létrehozni titkosított kapcsolatokat.

Ha a rendszeradminisztrátor hajlandó felhasználót létrehozni az adatbázis-kezelőt futtató szerveren (ritka), vagy van VPN hozzáférésünk az adatbázis-kezelőhöz, akkor OAS nélkül is

készíthetünk SSL által titkosított csatornát[24] a szerverig, teljesen transzparens módon. Ez a megoldás minden adatbázis-kezelőre és alkalmazásszerverre használható.

4.4. Milyen titkosított tárolási eszközök állnak rendelkezésre?

Adatok titkosított tárolását segíti a DBMS_CRYPTO csomag, mely szimmetrikus titkosítást és hash készítést elvégző függvényeket nyújt. Ezzel a készlettel az alkalmazás saját hatáskörben végezhet el titkosító lépéseket. A csomag teljeskörűen NIST kompatibilis, és minden, jó minőségű algoritmust tartalmaz.

Transzparensen titkosított táblaterek tetszőleges adatot tárolhatnak, a táblatér létrehozásakor meg kell adni a kulcstároló (Oracle Wallet Manager termék) helyét, ahol a rendszer tárolni fogja a jelszót a táblatérhez. A kulcstároló külön jelszóval védett. A táblatér jelszavának megváltoztatásához a kulcstárolót meg kell nyitni. Ezen kívül a táblatér használata semmiben nem különbözik a hagyományostól. Természetesen a titkosításoknak jelentős számítási igényük van, s ez a titkosított táblaterek használatakor meg fog látszani a rendszer válaszidejében és erőforráshasználatában.

4.5. Adatbázis-kezelő specifikus esetek

Az Oracle többek között kiterjedten használja a PL/SQL nevű nyelvet az adatbázis-kezelő kihasználására. E nyelv többek között függvénycsomagok készítését is lehetővé teszi, amikkel úgy lehet függvényeket és tárolt eljárásokat megosztani felhasználókkal, hogy azok nem látják a függvény forrását. Alaptelepítésben 175 csomag több mint ezer függvényt ad publikus láthatósággal. Ezeket a csomagokat érdemes hozzáférhetetlenné tenni.

A rendszer auditálási képességei híresek; lehet totális és végletekig finomhangolt módban is használni, a DBA figyelheti a teljes rendszerhasználatot, megfigyelhet csak néhány felhasználót, de készíthet megfigyelést szokatlan tevékenységek elfogására is.

A teljesítményelemzésekhez a rendszer ad nézeteket (dynamic performance v$ views), melyek részletes adatokkal szolgálnak a lefutott lekérdezésekről. Ezek egyik hátulütője, hogy bennük az összes nem túl régi lekérdezés jelen van részletes értékeléssel (például a v$sql tábla).

[24] SSH munkamenet képes port-továbbításra az SSL által titkosított kapcsolaton belül távoli gépre és vissza, ez az úgynevezett „protocol tunneling”. Ezzel a módszerrel átmenő továbbítást is végezhetünk, egy távoli gépen keresztül egy harmadik szerverre, ekkor csak a köztes gépik titkosított a forgalom, kivéve ha a köztes gépen nem építünk ki egy újabb titkosított kapcsolatot.

Mindenki, aki ki tudja olvasni e nézet tartalmát, részletes információkhoz juthat minden felhasználó tevékenységét illetően.

Leggyakrabban használt (és hivatalosan javasolt) megoldás a jogosultságok finomhangolására a PL/SQL csomagok használata. Az alkalmazás felhasználójának csak csomagok bizonyos tárolt eljárásaihoz van végrehajtási joga, melyekkel paramétereken keresztül kommunikál. A csomag tartalmazza az üzleti intelligenciát, mely végrehajtja a kért műveleteket, illetve ellenőrizheti a hívó jogosultságát. Ilyenkor az adatbázis felépítése teljesen ismeretlen marad a csomag felhasználója előtt, az implementációt elrejti a csomag belső működése, mely saját jogosultsági szintekkel operál az adatbázison.

5. Adattitkosítási megoldások

Látható, hogy az adat eltulajdonítása szinte bárhol, bármikor megtörténhet. Ennek kivédésére jöttek létre a titkosító algoritmusok. Három csoport érdekes számunkra: a hash, a szimmetrikus és az aszimmetrikus kulcsú titkosítás.

5.1. Hash

A hash nem igazi titkosítás, inkább egy bitfolyamnak egy kisebb, fix méretű értelmezési tartományra való leképezése. Elsődleges felhasználási területe adatfolyamok kivonatolása, ellenőrzőösszeg jelleggel. Elméletileg két adatfolyam bitről bitre megegyezik, ha a rájuk előállított hash-ek megegyeznek. Ez a leképezés egyirányú, a (jó minőségű) hash-ből nem lehet visszafejteni az eredeti tartalmat.

Mivel tetszőleges méretű adatot az adott algoritmusban definiált, véges méretű adatfolyamra képez le a hash, belátható, hogy kellően nagy és jól megválasztott bemenetek esetén előfordulhat ütközés (hash collision), így elérhető például, hogy két dokumentum tartalma különböző, de az egyik digitális aláírásában megadott hash egyezik a másik dokumentum hash-ével [WYY05]. Két általánosan elterjedt hash algoritmus az MD5 és az SHA. Gyakran úgy növelik a hash megbízhatóságát, hogy egyszerre több algoritmust is használnak, például az aszimmetrikus kulcsú titkosításnál a kulcsok integritását MD5 és SHA1 hash együttes felhasználásával biztosítják.

A hash egyik elterjedt felhasználási területe a jelszavak tárolása. A jelszót nem tároljuk el, csak a hash-ét. Később, ha ellenőrizni akarjuk, hogy a felhasználó jó jelszót adott-e meg, összehasonlítjuk a tárolt hash-t a felhasználó által adott jelszó hash-ével. Egyezés esetén feltételezzük, hogy a két jelszó is megegyezik. Példáért lásd az 1. táblázatot a 7. oldalon.

5.2. Szimmetrikus kulcsú titkosítás

Szimmetrikus kódolás és dekódolás esetén a kódoló és a dekódoló kulcs megegyezik. Választanunk kell egy olyan jelsorozatot, amit jó esetben csak a küldő és a fogadó ismer (shared secret). Két alverziója létezik, a blokk-kódoló és a folyamkódoló titkosítás. A blokk-kódoló algoritmus fix méretű csomagokra alkalmazza a transzformációt az adott kulccsal. Ha az adatcsomag nagyobb mint a blokkméret, blokkméretnyi darabokra osztja fel az adatcsomagot. Legismertebb szimmetrikus kulcsú algoritmusok a DES és az AES.

A blokk-kódoló két algoritmust használ, egyiket a kódoláshoz, másikat a dekódoláshoz.

Mindkét algoritmus két paramétert fogad, a bemenetet és a kulcsot. Minden kulcshoz a dekódolás a kódolás inverz függvénye.

$$E_K(M) = C$$
$$E_K^{-1}(C) = M$$

Ahol M az adat, C a titkosított adat, K a kulcs, s E a titkosító transzformáció függvénye. Minden K kulcsra E_K egy permutáció a bemeneten, így effektíve minden kulcs $2^n!$ lehetőség közül választ egyet (n a kimenet bitben mért hossza).

Az adatfolyam-kódoló is sorban kódolja a blokkokat, de minden blokk után az előző blokkot felhasználja a kulcs módosítására, így a bemenet is befolyásolja a titkosítás menetét. Ezen belül is számos, különböző megoldást alkalmaznak, melyek ismertetésére ebben a dolgozatban nem térek ki.

5.3. Aszimmetrikus kulcsú titkosítás

Az aszimmetrikus titkosítás alapgondolata az, hogy nem egyetlen, közös kulcs van, amit kapcsolatkezdeményezéskor meg kell beszélnie a két végpontnak egymással, hanem korábban, egyéb módokon eljut a két végpontra a kulcspár megfelelő eleme.

A kulcspárnak két eleme van: a publikus kulcs és a privát kulcs. A publikus kulcs bárkinek a kezére juthat, egyetlen felhasználási módja a privát kulcs tulajdonosának küldendő üzenetek titkosítása. A publikus kulccsal titkosított adatot csak a privát kulccsal lehet dekódolni. Ugyanígy, a privát kulccsal titkosított adatot pedig csak a publikus kulccsal lehet dekódolni. Ezzel lehet egyik irányba titkosságot, másik irányba hitelességet biztosítani.

Legelterjedtebb megvalósítása az RSA algoritmus. E kódolás erőssége matematikailag megalapozott. Alapelve, hogy tetszőlegesen nagy prímszámot elenyésző idő alatt lehet előállítani, míg két, közel azonos méretű prímszám szorzatát faktorizálni csak évezredes nagyságrendű időigénnyel lehet. Jelenleg több, a prímfaktorizáció bonyolultságára épülő algoritmus is használatban van.

Az RSA kulcspár előállításának menete:

- Keresünk két, hasonló méretű, nagy, de véletlenszerű prímszámot, p-t és q-t. (Általában úgy, hogy a szorzatuk adott bitszámú egész legyen.)
- Kiszámoljuk e két prím szorzatát: $n = pq$. Ezt nevezzük később modulusnak.

- Kiszámoljuk $\varphi(n)$[25] $= (p-1)(q-1)$-et, majd választunk egy olyan egész e-t, amire $1 < e < \varphi(n)$ és $gcd(e, \varphi(n)) = 1$, azaz e és $\varphi(n)$ relatív prím. Mivel p és q is prím, minden náluk kisebb szám relatív prím n-hez képest. Általában a 65537 kielégíti ezt a feltételt.

- Ezt az $e = 65537$-et, mint publikus exponenst publikáljuk, a $\varphi(n)$-t titokban tartjuk.

- Megállapítjuk a $d = e^{-1} \bmod \varphi(n)$-t[26], ezt megtartjuk a privát kulcs exponensének.

- Ekkor a publikus kulcs az (n, e) párosból fog állni, míg a privát kulcs a (p, q, d) hármasból. Általában n-t is tárolják, hogy ne kelljen minden alkalommal újraszámolni.

Észrevehető, hogy ha valaki az n-ből emberi idő alatt meg tudja határozni p-t és q-t, elő tudja állítani $\varphi(n)$-t, (e publikus), és d-t, ez alapján gyakorlatilag ismeri a teljes privát kulcsot. Ez alapjaiban rengetné meg az algoritmust.

A titkosítás lépései:

- Megszerezzük a címzett publikus kulcsát.

- Átalakítjuk az üzenetünket egész számok sorozatára, ahol $0 < m < n$.

- Kiszámoljuk az adott értékre vonatkozó titkosított értéket: $c = m^e \pmod{n}$.

- Sorba rendezzük az értékeket, majd elküldjük a címzettnek.

A visszafejtés igen egyszerű: kiszámoljuk az egyes értékeket: $m = c^d \pmod{n}$, minden értékre, melyet utána újra sorba állítunk.

Üzenetek aláírása: A fentebb említett folyamat megfordítása. A küldő a privát kulcsával titkosítja az üzenetet, melyet később mindenki dekódolhat a publikus kulcs birtokában. Ezzel igazolható, hogy a privát kulcs tulajdonosa írta az üzenetet. (Legalábbis az ő privát kulcsával lett titkosítva...)

Példáért lásd az E függeléket a 39. oldalon.

[25] Ez az Euler-függvény, mely megadja, hogy n-nel bezárólag hány, n-nel relatív prímszám van.

[26] Moduláris multiplikatív inverz

5.3.1. A publikus kulcs infrastruktúra, azaz a PKI

A PKI egy szabálygyűjtemény, mely leírja, hogyan kell digitális tanúsítványokat hitelesíteni, tárolni, terjeszteni, használni, illetve visszavonni. A rendszer lényege, hogy a kulcspárunk publikus felét egy megbízható(nak vélelmezett) harmadik fél hitelesíti, akit mindenki elfogad. Ez lehet a cég IT részlege, de akár a VeriSign[27] is, felhasználási igényektől (és költségvetéstől) függően. Ekkor a privát kulcsunkkal történő hitelesítés úgy nyerhet bizonyító erőt, hogy a publikus kulcs, mellyel a digitális aláírást visszafejthetjük, egy megbízható fél hitelesítette. Ez a harmadik felet (röviden: CA, azaz Certificate Authority) tanúsítványok láncolatával hitelesítik a gyökértanúsítványig, mely elismerése közös megegyezéssel történik. E láncolat segítségével nem szükséges minden CA-nak saját gyökértanúsítványt létrehoznia, elég a saját tanúsítványát oly módon elismertetnie más CA-val, hogy azt feljogosítsák tanúsítványok hitelesítésére. Ez különösen jól jön cégek belső tanúsítvány-alapú megoldásaihoz.

5.4. Az adat útja

Az adatnak valahogyan el kell jutnia a feladótól a címzettig. Ez történhet adathordozón, hagyományos szállítással, de történhet hálózaton keresztül, elektronikusan is.

5.4.1. Hálózati protokollok + SSL/TLS

Az SSL (Secure Socket Layer), illetve az azt leváltó TLS (Transport Layer Security) adatfolyamot titkosít, így szinte bármilyen adatátvitelt bújtathatunk benne a két végpont között. Minden SSL/TLS kapcsolat kézfogással kezdődik. Először megállapítják, melyik a legerősebb hash függvény, amit mindkét fél ismer. Majd a szerver elküldi a digitális tanúsítványát (mely tartalmazza a szerver publikus kulcsát is), melyet a kliens ellenőriz(het), hogy érvényes-e, nem lett-e visszavonva, egyáltalán a szerveré-e. Itt lép be a trükk, amit az aszimmetrikus titkosítás tesz lehetővé: a kliens a publikus kulccsal titkosítva elküld egy véletlen számot, melyet csak a privát kulcsot birtokló tud dekódolni, mely jó esetben csak a szerver. Ezt a véletlen számot felhasználva mindkét fél generál egy kulcsot, amit a munkamenetben szimmetrikus titkosításra használnak. Látható, hogy a kulcs alapjául szolgáló véletlen szám titkosítva utazott a hálózaton, s e titkosítás kulcsa sem lépett ki a hálózatra. A munkamenet kulcsát valamelyik fél rendszeresen megújíttatja, valamilyen határ elérésekor (általában az eltelt idő vagy átvitt adat

[27] A VeriSign, Inc. egy elismert tanúsítvány hitelesítő vállalkozás, mely emellett számos más hálózatbiztonsági területen is jelen van.

függvényében), hogy egy kulcs kitalálásához ne szolgáltassanak elegendő adatot. Ilyen módon működik többek között a HTTPS és az IMAPS is, de hasonlóan lehet bújtatni SMTP, LDAP, egyéb szöveges és bináris protokollokat is SSL/TLS csatornán át.

5.4.2. SSL/TLS + kliensoldali tanúsítványok

Az SSL/TLS kézfogást ki lehet azzal egészíteni, hogy a szerver is elvárja a kliens digitális tanúsítványát. Ekkor egyrészt az azonosítás elvégezhető a kliens tanúsítványában szereplő név alapján, másrészt a szerver megbizonyosodik arról, hogy a kliens rendelkezik a privát kulccsal, ami az általa elküldött publikus kulcshoz tartozik. Ezt a megoldást például PostgreSQL-ben is használhatjuk.

6. Adatbázis réteg felett alkalmazott titkosítás

Látható, hogy számos megoldás létezik titkosításra, van miből válogatni. Van precedens arra is, hogy több módszert kombinálnak. De mielőtt nekilátunk a programozásnak, több tényezőt is figyelembe kell venni:

- Legtöbb hosting megoldásnál az alkalmazás- és adatbázisszervert ugyanaz a rendszergazda személy vagy csoport felügyeli, így a szerveroldali alkalmazásban titkosítani céltalan, különös tekintettel az interpretált szerveroldali szkriptekre.
- Míg a legtöbb esetben a jelszó csak autentikációs céllal létezik, így cseréje szinte következmény nélküli, titkosításnál a jelszó elvesztése katasztrofális adatvesztést von maga után.
- Ügyelni kell az algoritmus terjesztésére. Ha a kliens kódja szerveroldalról egyszerűen módosítható (jellemzően webes felületek), akkor bármikor beültethető kód a jelszó tárolására.
- Jó minőségű titkosításból nem lehet kinyerni lexikális információkat, azaz titkosított formában nem rendezhetőek.

Látszik, hogy olyan megoldást kell találnunk, mely a titkosítás terhét leveszi a szerver válláról, s ez két helyen lehetséges: a kliensnél, vagy egy megbízható(bb)nak tartott köztes szervernél.

6.1. Előfeldolgozás behelyettesítéssel

Ha nem akarjuk, hogy valaki lehallgassa a jelszavunkat, akkor ne küldjük el senkinek. Ebbe beletartozik az is, amikor az adatbázis-kezelő titkosító függvényeit használjuk, az esetben ugyanis az SQL lekérdezés tartalmazná a jelszót. Erre megoldás lehet, ha a lekérdezést előfeldolgozzuk, s a titkosító függvényeket lecseréljük a titkosított értékekkel.

Ekkor egy konfiguráció alapján fel kell ismernünk, hogy mely mezők védettek, s ezeket dinamikusan kell cserélnünk a lekérdezésben. Hátránya, hogy egyrészt az SQL egy környezetfüggetlen nyelvtan, melyet teljes egészében reguláris kifejezésekkel (ami reguláris nyelvtan) elég nehéz (lehetetlen) feldolgozni, értelmezőt kell írni hozzá (vagy eltekintünk a teljes szabvány támogatásától, pl. kurzorok, dinamikus kiértékelések).

Ha ezt az utat választjuk, akkor létre kell hoznunk egy konfigurációt, mely meghatározza, mely mezőket kell titkosítani. Majd ki kell terjeszteni az adatbázis-elérési réteget, hogy minden lekérdezést módosíthassunk, mielőtt az elindulna a hálózaton át. Ekkor ügyelnünk kell arra, hogy a rendezéseket csak helyben tudjuk megoldani, mert a titkosított forma lexikailag nem releváns, illetve részsztring-alapú, és intervallumkeresést nem végezhetünk ezen mezőkön.

Ez a módszer szimmetrikus titkosítást alkalmaz, mely esetén feltétel, hogy minden felhasználó, aki használja az adatbázist, ismerje ezt a jelszót. Ilyenkor a jelszót ismerők, de már használni nem jogosultak számát úgy lehet csak csökkenteni, ha a teljes adatbázist újratitkosítjuk az új jelszóval. Ez persze igen időigényes lehet.

6.2. Előfeldolgozás aszimmetrikus behelyettesítéssel

Módosíthatjuk a módszert úgy, hogy a beszúrás és a kiolvasás független művelet legyen. Ekkor megjelenik lehetőségként az, hogy a titkosításhoz szükséges jelszóval rendelkező felhasználó nem tud visszafejteni, ha nincs meg nála a visszafejtéshez szükséges jelszó, illetve fordított esetben az, aki kiolvasni tud a visszafejtésre alkalmas jelszóval, nem tud érvényes adatot beszúrni.

Erre a legalkalmasabb megoldás az aszimmetrikus kulcsú titkosítás. A beszúrást a publikus kulccsal végezzük, a kiolvasást a privát kulccsal.

Ennek a módszernek ugyanazok a hátrányai, mint az előző fejezetben említetteknek. Mindkét esetben kénytelenek vagyunk egy újabb réteget elhelyezni az adatbázis-elérésünkbe.

7. Egy közös ellenség: SQL Injection

Az adatbázisok tervezése és fejlesztése alatt minden gyártó nagy hangsúlyt fektet a biztonságra. A kommunikációs protokollok védekeznek az adatmódosítás és eltérítés ellen. De egyetlen tényezőt sosem fognak tudni teljes egészében ellenőrizni: az adatbázist felhasználó alkalmazást. Az alkalmazástól érkező érvényes és helyes parancsokat az adatbázis-kezelő elfogadja a felhasználó kifejezett szándékának, így ha az végrehajtható, további kérdezősködés nélkül végrehajtja[LAHG05, Lit07].

Ilyen jellegű támadások az alkalmazás felől érkeznek, s legitim utasításoknak álcázzák magukat. Az adatbázis-kezelő részben védekezhet úgy, hogy külön felhasználókat használ az adatbázis olvasására és írására, de szükség van az alkalmazástervezők által kidolgozott biztonságos adatbázis-elérési szabályok betartatására is. A sikeres kódbeszúrásos támadások több mint 99%-a megelőzhető lett volna, ha a fejlesztők vették volna a fáradságot a bemenet teljeskörű és alapos ellenőrzésére. A továbbiakban bemutatok néhány, jól bevált módszert, mellyel számos kicsi és nagy weboldal mögé férkőztek be támadók.

A legtöbb programozási bevezető webes szkriptnyelvekhez nem fektet túl nagy hangsúlyt az adatbázis-kezelésre, csak a lényegre szorítkozik. A példánkban most csak annyi áll, hogy

```
SELECT userid FROM login WHERE user = "%s" AND password = "%s"
```

Mivel a példák nem jeleskednek a hibakezelés és felhasználói bemenet ellenőrzésének terén, egyszerűen hozzárakják a bemenetet ehhez a sztringhez. Ez a hozzáállás sajnos megragad a programozókban, s a későbbiekben is abban a hitben maradnak, hogy ez így biztonságos. Később, mikor egy szemfüles cracker[28] rátalál a weboldalra, tüstént elkezdi feszegetni a határokat. A fenti lekérdezés – elviekben – egyetlen sorral térne vissza, ha a bejelentkezés sikeres, s nullával, ha nem. A rendszer átverése a megjegyzések használatával történhet, ha a fent (nem) nevezett személy a bejelentkező ablakban felhasználónévnek azt adja meg, hogy admin"--, akkor a lekérdezés a következő:

```
SELECT userid FROM login WHERE user = "admin" --" AND password = "mindegy"
```

ahol a kettős kötőjel a megjegyzés kezdetét jelenti, ekkor az egyetlen szűrési feltétel a felhasználónévre történik. Eggyel fejlettebb lépés, amikor már némi ismerete van a támadónak a táblaszerkezetről, ekkor akár saját felhasználót is létrehozhat. Gyenge biztonsági szintű weboldalakon egy mezőbe beírhat akár egy komplett SQL lekérdezést is:

[28] Weboldalak feltörésében valamely okból (ideológiai, anyagi) érdekelt személy vö. to crack – törni

```
"; INSERT INTO login (user, password, type)
   VALUES ("kokesz", "hekk", "Admin") --
```

Ekkor mindegy, hogy az első lekérdezés mivel tér vissza, a lényeg, hogy a második lekérdezés is sikeresen lefusson. A MySQL kliens PHP-hez készített interfésze pont ezért nem engedélyezi egy lekérdezésben több utasítás elküldését.

MSSQL Server termék esetén azzal is számolni kell, hogy Windows alatt a szolgáltatások jellemzően adminisztrátor feletti jogosultsági szinten futnak. Az MSSQL Server tartalmaz úgynevezett kiterjesztett eljárásokat, melyek prefixe „xp_". Ezek közül is a legveszélyesebb az „xp_cmdshell", mely parancssori végrehajtást tesz lehetővé. Ha egy támadó egy ilyen gépen megtudja hívni a „master..xp_cmdshell" eljárást paraméterekkel felvértezve, tetszőleges utasítást hajthat végre rendszergazdaként a gépen. Leggyakrabban egy új felhasználót hoznak létre (természetesen adminisztrátori jogokkal), hogy később távoli asztali kapcsolaton keresztül kényelmesen beléphessenek körülnézni[29]. Az MSSQL Server számos, dokumentálatlan eljárással rendelkezik (például OLE Automation-ön keresztül COM objektumokat hozhatunk létre, s azok metódusait meghívhatjuk), illetve más gyengeségekkel is rendelkezik, ami miatt nem került részletezésre e dolgozatban.

Léteznek SQL proxy-k, melyek tűzfalként funkcionálnak[30], ezek többek között feketelistákkal és jellegzetes minták felismerésével dolgoznak. Az adatbázis-kezelő rendszerek működésében felfedezhető némi visszásság, például a beágyazott megjegyzéseket némelyik egyszerűen kitörli a megjegyzést feldolgozás közben. Ez alkalmas lehet feketelisták kikerülésére, mert a DE/**/LETE **FROM** users esetén nem állapítható meg konkrét egyezés tiltott kulcsszavakkal, a lefutás mégis adatvesztést eredményez. Ez ellen hatásos, ha a megjegyzést szóközzel helyettesítjük, illetve még az alkalmazás megfosztja a kommentektől a lekérdezést, ekkor a lekérdezés szintaktikailag helytelen lesz.

Bizonyos adatbázis-kezelők támogatnak kényelmi funkciókat, melyekkel például kényes sztringeket tudunk hexadecimális reprezentációban átadni a szervernek. Kézenfekvő kihasználni ezt, hogy beolvassunk fájlokat a fájlrendszerből:

```
SELECT CONCAT('0x', HEX('c:\\boot.ini'))    /* 0x633A5C626F6F742E696E69 */
SELECT LOAD_FILE(0x633A5C626F6F742E696E69) /* Csak Windowson */
```

Egy tűrhetően megírt webalkalmazás nem ad vissza részletes hibaüzenetet, legfeljebb egy

[29] Az „**exec** sp_configure 'xp_cmdshell',0" lekérdezéssel letiltható ez az eljárás.

[30] Pl. GreenSQL Database Firewall

udvarias bocsánatkérő szöveget, amiből nem derül ki kívülállóknak, mi a hiba pontos oka.

Másodrendű támadási mód, ha nem azonnali eredményt adó műveleteket használunk ki, ezeket jellemzően kevésbé is ellenőrzik, mivel nincs közvetlen hatása a munkamenetre. Ilyen helyzet például egy regisztrációs űrlap. Ha felhasználónévnek megadjuk a következő sztringet: `'+(SELECT passwd FROM users LIMIT 1)+'`, megeshet, hogy egy jelszóhash-t vagy rosszabb esetben magát a jelszót látjuk névként a következő oldalakon. Alapvető kódbeszúrásos technikák ellen hatékony, ha a bementet escape szekvenciákkal ártalmatlanítjuk, ekkor a különleges jelentésű karakterek ártalmatlan szekvenciákká redukálódnak (mint itt az idézőjelek).

Gyakori, hogy az alkalmazás kap egy paramétert, mely vélt típusa pozitív egész szám, de nem tesz semmit ennek ellenőrzésére, betartatására. Tipikusan ilyen a webportálok listázó oldalainak paraméterezése. Egy lapozó algoritmus több, szám típusú paramétert használ az oldalak nyilvántartására. Ha ezt nem ellenőrzi, akkor az alábbi URL igen könnyen kihasználható oldalt takar: `http://shop.example.com/articles.jsp?start=3` Ha itt kicsit megpiszkáljuk a start nevű paramétert, megeshet, hogy komplett SQL lekérdezést is adhatunk neki. Példák alapján készített lapozóalgoritmusok leegyszerűsítve így néznek ki:

```
SELECT id, title, description, price FROM articles
LIMIT $start,10 ORDER BY title
```

Ha átadjuk az oldalnak az általunk fabrikált HTTP kérést, tisztább képet kaphatunk a weboldal hátteréről:

```
http://shop.example.com/articles.jsp?start=1 UNION ALL
SELECT NULL, TABLE_SCHEMA, TABLE_NAME, VERSION
FROM INFORMATION_SCHEMA.TABLES --
```

Tegyük fel, hogy sikerült elsőre. A rosszul megírt kód eredménye az lett, hogy:

```
SELECT id, title, description, price FROM articles LIMIT 1 UNION ALL
SELECT NULL, TABLE_SCHEMA, TABLE_NAME, VERSION
FROM INFORMATION_SCHEMA.TABLES --,10 ORDER BY title
```

Ezzel ki is listáztuk az összes elérhető táblát[31], de így az összes objektumról is tájékozódhatunk. Hasonló támadásnak esett áldozatul a MySQL weboldala 2011. március 27-én és a tűzfalairól híres Barracuda Networks 2011. április 11-én. Az efféle támadásokat jellemzően a támadó

[31] Az INFORMATION_SCHEMA metaadatbázist az SQL92 szabvány vezette be, metaadatot tartalmaz minden objektumról. Az Oracle ezt nem támogatja, saját formátumú Data Dictionary-t tart fenn.

publikálja, ha nem fűződik érdeke az eltitkoláshoz[32], haszonszerzési célú betörések az esetek túlnyomó részében titokban maradnak (mivel ezek az áldozat hírnevének és részvényárfolyamának is ártanak), így még találgatásokba sem lehet bocsátkozni, mennyi betörés marad eltitkolva. Pedig legutóbbi példánkban elég lenne ellenőrizni, hogy a paraméter tényleg szám-e...

Az adatbázisok hatékony kihasználásában előrelépést jelentett a paraméterezett utasítások bevezetése (prepared, vagy parametrized statements). Ez az újítás lehetővé tette azt, hogy egy hozzáférési tervet használjanak ugyanannak a lekérdezésnek különböző paraméterekkel való futtatására. Használata egyszerű, az eredeti lekérdezésben a paraméterek helyét helytartókkal (placeholder) kell megjelölni, későbbiekben csak a helytartók pozíciójához kötjük a paramétereket, és a művelet végre is hajtható. Így minden paraméter adatként fog eljutni az adatbáziskezelőhöz, s nem fog részt venni a lekérdezés összeállításában. Ezzel az eddig ismertetett technikákat egy csapásra kiütöttük, csak a paraméterként átvitt adat késleltetett hatását kell ellenőrizni, nehogy esetleg valaki felhasználónévnek egy Javascript kódot küldjön be...

Az alkalmazás tervezésekor érdemes minden kódot és üzleti logikát a saját helyén tartani, ez alól nem kivétel az adatbázis használata sem. Követendő gyakorlat, hogy az alkalmazásból állandóan elküldött lekérdezések helyett tárolt eljárásban (stored procedure) rögzítsük az üzleti logika adatbázist érintő részeit, s ezt paraméterezzük. Ekkor az alkalmazás elől elfedhetővé válik az adatbázis-szerkezet, s bizonyos adatbázis-kezelők esetén a tárolt eljáráson belüli kód más jogokkal futhat, mint a hívó, így az alkalmazás felhasználójának elfoglalása esetén sem szerezhető hozzáférés magához az adathoz. A tárolt eljárások is hívhatók paraméterezett módon, így alkalmazásuk nem okoz kényelmetlenséget az alkalmazás fejlesztésekor, továbbá rögzített a végrehajtási tervük, mivel ismert a lekérdezés minden aspektusa, nem utolsósorban egy bonyolult utasítás esetén nem kell minden egyes alkalommal a teljes utasítást elküldeni a szervernek, elég megadni az eljárás nevét és átadni a paramétereket. Nagy volumenű rendszerekben általános az akár 8000 soros lekérdezés is.

Objektum–relációs modellek (helyenként Entity framework néven jegyzik) az objektumorientált programozási nyelvekben terjedtek el. Célja, hogy minden adatbázis objektumot (pl. rekordot) egy osztály egy példánya képviseljen, s az azon elvégzett módosítások transzparensen átvezetésre kerüljenek az adatbázisba. Előnyük, hogy elfedik az aktuális adatbázis-kezelő sajátosságait, s belső mechanizmusaik kihasználják a rendelkezésre álló eszközöket, mint például a parametrizált utasításokat.

[32]Például a HBGary, Inc. informatikai biztonságtechnikai tanácsadó cég majdnem totális elpusztítása 2011. február 5-én. Állítólag egy 16 éves kislány tette.

8. Összegzés

Az előző fejezetekben megismerkedtünk az adatbázis-kezelők vezérelveivel, tervezési sajátosságaival, autentikációs moduljainak működésével, a titkosítás alapjaival, a titkosított adatátvitel módszereivel, s megoldásokkal, melyek az adatot lehető legtovább titkosítva tartják. Láttunk példát adatbázist használó alkalmazáson keresztüli támadásokra, s javaslatot tettünk ezek kivédésére.

Az adatbiztonság kényes témaköre kiapadhatatlan forrás informatikai rendszerek összetett működésének megfigyelésére, ahol átfogó rendszerszemlélet és részletekbe menő ismeretek szükségesek, hogy a rendszer talpon maradjon külső és belső fenyegetésekkel szemben is. Az alkalmazások bonyolultságának növekedésével, a kódbázis felduzzadásával arányosan nő a kihasználható hibák száma. Minden egyes hibajavítás szülhet új, eddig ismeretlen hibát, minden új eszköz tartalmazhat kevéssé átgondolt megoldásokat, melyeket türelmes támadók egyszer úgyis megtalálnak, versenyt futva a hibák kijavításával.

Ebben a dolgozatban három közismert többfelhasználós, SQL-kompatibilis relációs adatbáziskezelő-rendszert érintettünk. Mindnek vannak előnyei, hátrányai, céljai, buktatói, ami miatt nagyon nehéz lehet egy megoldás tervezésekor kiválasztani a megfelelő adatbázis-kezelőt, ezért vegyünk sorra egy szempontrendszert, mely segíthet e döntés meghozatalában.

Fő szempontok az üzleti intelligencia adatbázis-kezelőbe helyezhetősége, a kezelhetőség, a skálázhatóság, a biztonság, az eszközkészlet és a kliensoldali interfész használhatósága. Ezek alapján az alábbi következtetéseket vonhatjuk le.

A MySQL-t azoknak ajánlott, akik gyorsan akarnak felállítani egy adatbázist, mely kliensoldalon a legszélesebb körben támogatott. MySQL gyakorlatilag minden modern programozási nyelvből és keretrendszerből elérhető, adminisztrálása pedig alapszinten triviális. Ezek a jellemzők tették a tárhelyszolgáltatók első számú választásává az egész világon, amellett, hogy nyílt forrású projektként GPL licenc alatt szabadon felhasználható (amíg származtatott termékké nem alakítjuk át). Jól skálázódik minden architektúrán, és Cluster megoldását számos olyan nagy cég alkalmazza, mint az Alcatel-Lucent, a Telenor, a go2, vagy az IDC. Hátránya, hogy bonyolult tárolt eljárást nehéz benne írni, s külső felhasználóazonosítás nem lehetséges, így nagyvállalati környezetben nehézkes a használata. Tárhelyfoglalásra kvóták nem adhatók meg a felhasználóknak, csak a táblaterek mérete szab határt az adatbázisnak. Sajnos az Oracle általi felvásárlása

után számos fork[33] jött létre az 5.1-es verzióból, így felaprózva a fejlesztői gárdát. Audit képességeinek teljes hiánya pedig nagyrészt kizárja állami vagy katonai megrendelésekben való felhasználását.

A PostgreSQL-t azon felhasználóknak ajánlott, akiknek szükségük van egy olyan relációs adatbázis-kezelőre, mely tranzakciók támogatása mellett több nyelven is képes tárolt eljárásokat futtatni, s komplex adattípusokat kezel, miközben a legtöbb nyelvhez van kliensoldali támogatása. Azonosítási megoldásai szinte minden igényhez testre szabhatók, és a BSD licenc semmilyen gátat nem szab átalakításának, átírásának, üzleti felhasználásának. Neves felhasználói közt van a BASF, a Cisco, a Juniper Networks, a Skype, az NTT, a Telstra, a Genentech, a Jyväskyläi Egyetem, az Apple és számos USA-beli állami intézmény és intézet. A PL/pgSQL nevű hivatalos nyelvi plugint szándékosan úgy fejlesztik, hogy Oracle-ről a lehető legkönnyebben lehessen migrálni PostgreSQL-re. Hátránya, hogy adminisztrálása a MySQL-en nevelkedettek számára nehézkes, mert sok olyan aspektusra is ügyelni kell, ami MySQL-ben nincs jelen.

Az Oracle-t azok válasszák, akik egyrészt megfelelő méretű tőkebefektetésre hajlandóak mind a terméket magát, mind a hozzá szükséges szakembergárda és hardverpark finanszírozása terén, másrészt állami előírások, illetve a feladat sajátosságai miatt egy robusztus, több száz gépen keresztül is jól skálázható adatbázis-kezelőre van szükségük. A PL/SQL procedurális nyelvben akár a teljes üzleti logika implementálható, s az is megoldható, hogy egy tárolt eljárás HTML kódot adjon vissza, mely további feldolgozás nélkül elküldhető a kliens böngészőjének. Felhasználói között ott van a National Instruments, a Turkcell, a Vodafone, illetve a Verizon is. Hátránya, hogy a számos képesség nagyon felfújja az kódbázist és az erőforrásigényt is, ezért rossz erőforrás-tervezés esetén a válaszidők csapnivalóak lesznek.

Eme adatbázis-kezelők értékeléséből is kitűnik, hogy van átfedés a felhasználási területekben. A döntést viszont csak az adott probléma és rendelkezésre álló erőforrások ismeretében lehet meghozni.

[33] Szoftverfejlesztésben a „fork” a kódbázis lemásolása a célból, hogy a fejlesztésből kiváló csoport új irányba terelje az alkalmazás készítését.

9. Irodalomjegyzék

[Ano03] Anonym. http://www.securiteam.com/tools/5WP031FA0U.html, 2003.

[DD03] Korry Douglas and Susan Douglas. *PostgreSQL*. Sams, 2003. ISBN 9780735712577.

[Gro10] PostgreSQL Global Development Group. PostgreSQL version 9.0.1. http://postgresql.org/, 2010.

[Kof05] Michael Kofler. *The Definitive Guide to MySQL 5*. Apress, 2005. ISBN 9781590595350.

[LAHG05] David Litchfield, Chris Anley, John Heasman, and Bill Grindlay. *The Database Hacker's Handbook: Defending Database Servers*. Wiley, 2005. ISBN 9780764578014.

[Lit07] David Litchfield. *The Oracle Hacker's Handbook: Hacking and Defending Oracle*. Wiley, 2007. ISBN 9780470080221.

[Lon04] Kevin Loney. *Oracle Database 10g: The Complete Reference*. Oracle Press–McGraw-Hill/Osborne, 2004. ISBN 0072253517.

[MyS06] MySQL AB. *MySQL Administrator's Guide and Language Reference (2nd edition)*. MySQL Press, 2006. ISBN 9780672328701.

[MyS08] MySQL AB. MySQL 5.1. http://mysql.com/, 2008.

[Ora03] Oracle Corporation. Oracle Identity Management 10g, 2003.

[Ora06a] Oracle Corporation. Oracle 10g Administrator's Guide. http://download.oracle.com/docs/cd/B19306_01/server.102/b14231.pdf, 2006.

[Ora06b] Oracle Corporation. Oracle Database 10g Express Edition. http://www.oracle.com/technetwork/database/express-edition, 2006.

[SM05] Richard Stones and Neil Matthew. *Beginning Databases with PostgreSQL: From Novice to Professional*. Apress, 2005. ISBN 9781590594780.

[Vas09] Vikram Vaswani. *MySQL Database Usage and Administration (1st edition)*. McGraw-Hill/Osborne Press, 2009. ISBN 9780071605496.

[WD02] John C. Worsley and Joshua D. Drake. *Practical PostgreSQL*. O'Reilly Media, 2002. ISBN 9781565928466.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. *Advances in Cryptology – CRYPTO 2005*, 3621/2005:17 – 36, August 2005.

A. MySQL privilégiumok

Privilégium	Jelentés
SELECT	Lekérdezés táblából az adott hatókörben
INSERT	Beszúrás táblába az adott hatókörben
UPDATE	Létező sorok módosítása táblában
DELETE	Sorok törlése táblákból
CREATE	Adatbázis, tábla, vagy index létrehozása
DROP	Adatbázis, tábla, vagy nézet törlése
REFERENCES	(Kulcsszó, de még nem használt)
EVENT	Belső ütemező kezelése
ALTER	Táblák definíciójának megváltoztatása
INDEX	Indexek kezelése létező táblákon
CREATE TEMPORARY TABLES	Ideiglenes táblák létehozása
LOCK TABLES	Táblák zárolása (csak amikhez SELECT is van)
TRIGGER	Létező táblákon triggerek kezelése
CREATE VIEW	Nézet létrehozása
SHOW VIEW	Létező nézet definíciójának megtekintése
CREATE ROUTINE	Tárolt eljárás vagy függvény létrehozása
ALTER ROUTINE	Tárolt eljárás vagy függvény módosítása
EXECUTE	Tárolt eljárás vagy függvény végrehajtása
FILE	Szerveren található fájlok írása, olvasása

2. táblázat. Alapvető MySQL privilégiumok

Privilégium	Jelentés
GRANT OPTION	Jogosultságok továbbadásának lehetősége
CREATE USER	Felhasználók kezelése
PROCESS	Aktuális műveletenek megjelenítése
RELOAD	Privilégiumtáblák újratöltése a tárolt változatból
REPLICATION CLIENT	Replikációs állapot megtekintése
REPLICATION SLAVE	Replikációs szolgaszerverek különleges joga
SHOW DATABASES	Összes adatbázis megtekintése
SHUTDOWN	Konzolos leállítás lehetősége
SUPER	Teljhatalom minden konfiguráció felett
ALL [PRIVILEGES]	Minden privilégium, kivéve a GRANT OPTION
USAGE	Jelentése: Semmilyen globális jog

3. táblázat. Egyéb MySQL privilégiumok

B. PostgreSQL privilégiumok

Privilégium	Jelentés
SELECT	Lekérdezés táblából az adott hatókörben
INSERT	Beszúrás a táblába az adott hatókörben
UPDATE	Létező sorok módosítása a táblában az adott hatókörben
DELETE	Törlés a táblából az adott hatókörben
TRUNCATE	Tábla ürítése az adott hatókörben
REFERENCES	Idegen kulcsok kezelése
TRIGGER	Triggerek kezelése
CREATE	Objektum létrehozása az adott hatókörben
CONNECT	Kapcsolódás a szerverhez
TEMPORARY	Ideiglenes táblák használata
EXECUTE	Tárolt eljárások futtatása
USAGE	Séma használata en bloc

4. táblázat. PostgreSQL privilégiumok

C. Oracle érdekességek

Az alábbi lekérdezés visszaadja az összes olyan csomagot és függvényt, amelyet minden felhasználó használhat, s így biztonsági problémát jelenthet[Lit07].

```
SELECT *
FROM dba_tab_privs p, all_arguments a
WHERE grantee = 'PUBLIC'
AND privilege = 'EXECUTE'
AND p.table_name = a.package_name
AND p.owner = a.owner
AND a.position = 0
AND a.in_out = 'OUT'
ORDER BY p.owner, p.table_name, p.grantee
```

D. Példa szimmetrikus kulcsú titkosításra

A legújabb szimmetrikus kulcsú titkosító algoritmus, az AES, 9[34] menetes kódolást alkalmaz. Minden menet 4 transzformációból áll. Az első lépésben (SubBytes) egy S-BOX[35] nevű kétdimenziós mátrix alapján minden bájtot lecserél. Második lépésben (ShiftRows) minden sort a sorszámának megfelelő számú bájttal forgat körbe. Tehát az első sort, melynek 0 az indexe, nem forgatja, a második sort eggyel forgatja balra, tehát a bal szélső elem a jobb szélső lesz, a harmadik sort kettővel forgatja el, s így tovább. A harmadik lépés a MixColumns, ahol minden oszlopvektort megszorozza egy adott mátrixszal. A negyedik lépésben (AddRoundKey) az adott menet kulcsát oszloponként hozzáadja a bemenethez. Ez történik meg kilencszer. A tizedik, végső körben a harmadik, MixColumns lépés kimarad.

A kulcs menetenként úgy változik, hogy először a kulcs negyedik szavát (word) elforgatjuk eggyel, alkalmazzuk rá az előbb említett SubBytes transzformációt, majd kizáró vagyoljuk az első szót és az Rcon mátrix első oszlopát. Ez lesz a round key. A többi 3 körben mindig a néggyel ezelőtti szót adjuk hozzá az előző kulcshoz. Viszont minden negyedik alkalommal, amikor új round key-t generálunk (az AddRoundKey mindig az aktuális menet kulcsát használja), újrajátsszuk az első kulcs létehozását az aktuális round key-en. Összesen 10 round key jön így létre.

Az AES titkosítás lépésenkénti bemutatása több oldalt tenne ki, így annak közlésétől eltekintek.

A dekódolás a kódolás algoritmusának fordított irányban való alkalmazása.

E. Példa aszimmetrikus kulcsú titkosításra

E.1. Kulcs előállítása

Generáljunk két véletlenszerű, de nagyjából egyforma méretű prímszámot. Példánk kedvéért legyen $p = 107$, $q = 311$. Állítsuk elő a modulust: $n = pq$. Számoljuk ki $\varphi(n) = (p-1)(q-1)$-t, majd válasszunk egy e modulust melyre igaz, hogy $1 < e < \varphi(n)$ és $gcd(e, \varphi(n)) = 1$. Válasszuk most a 3-at.

[34] 128 bites kulcs esetén 9 menetes, 192 bit esetén 11 menetes, 256 bits esetén 13 menetes.

[35] Substitution Box, helyettesítőtáblázat, a minták felismerését nehezíti meg

$$
\begin{aligned}
p &= 107 \\
q &= 311 \\
n &= pq = 107 \cdot 311 = 33277 \\
\varphi(n) &= (p-1)(q-1) = 106 \cdot 310 = 32860 \\
e &= 3 \\
d &= e^{-1} \pmod{\varphi(n)} = 3^{-1} \pmod{32860} = 21907
\end{aligned}
$$

Ekkor a publikus kulcsunk: ($n = 33277$, $e = 3$), a privát kulcsunk: ($n = 33277$, $p = 107$, $q = 311$, $d = 21907$).

E.2. A kulcspár használata

Tegyük fel, hogy a bankunk el akarja küldeni nekünk az aktuális folyószámla-egyenlegünket, ami legyen most $m = 4242$. Ekkor a bank, a nála levő publikus kulcsunkkal titkosítja ezt a számot:

$$c = m^e \pmod{n} = 4242^3 \pmod{33277} = 28160$$

Mikor mi megkapjuk a $c = 28160$ tartalmú üzenetet, privát kulcsunkkal dekódoljuk:

$$m = c^d \pmod{n} = 28160^{21907} \pmod{33277} = 4242$$

Így vissza is kaptuk az eredeti üzenetet. Jó tudni, hogy az üzenet nem érheti el n-t, tehát ha az egyenlegünk 424242 lenne, akkor ezt a számot ketté kellene bontani, két részletben titkosítani, átvinni sorrendben, fogadó oldalon sorban dekódolni majd összevonni.