HADMÉRNÖK

III. Évfolyam 4. szám - 2008. december

Fleiner Rita
Budapesti Műszaki Főiskola
fleiner.rita@nik.bmf.hu

SQL INJEKCIÓRA ÉPÜLŐ TÁMADÁSOK ÉS VÉDEKEZÉSI LEHETŐSÉGEK

Absztrakt

A publikáció célja az adatbázis háttérrel rendelkező alkalmazások biztonságát fenyegető támadások egyik típusának, az SQL injekciónak a bemutatása, ismertetve a támadás alapjait, típusait és az ellene való védekezési lehetőségeket. Az SQL injekciós támadás dinamikusan szerkesztett SQL utasításba illeszt káros tevékenységet megvalósító kódot, mely az adatbázisokra épülő alkalmazásoknak, különös tekintettel a webes alkalmazásoknak a sérülékenységét kihasználva valósulhat meg.

In this paper we describe the SQL injection based attack threatening the security of applications with database backend. The bases and the types of the attack are studied, and the possibilities of the prevention and detection are considered. SQL injection is a class of attacks where un-sanitized user input is able to change the structure of a dynamically built SQL query, injecting malicious code into the database.The exploitation can occur through the vulnerability of application, especially web application that utilize a database server.

Kulcsszavak: *adatbázis biztonság, adatbázis védelem, fenyegetettség, informatikai biztonság, SQL injekció ~ database security, database protection, vulnerability, IT security, SQL injection*

Bevezetés

Napjainkban az informatikai szolgáltatások jelentős része adatbázisokban tárolt információk kezeléséhez, rendelkezésre bocsátásához kapcsolódik. Az adatbázisok kezelésének, elérésének legelterjedtebb módja az SQL (szabványos adatbázis kezelő nyelv) alapú hozzáférés.

Az adatbázisok lekérdezésére alapuló támadás az SQL injekció, melynek igen széleskörű következményei lehetnek, mint például bizalmas információk kiszivárgása, adatok integritásának megsértése, hozzáférési szabályok felülírása, távoli parancsok lefuttatása és szolgáltatás megtagadása típusú támadás (DoS) indítása. A támadó megszerezheti az alkalmazás mögött álló teljes adatbázis tartalmát, megváltoztathatja az adatbázis felépítését

(sémáját), illetve tartalmát, sőt az adatbázis szervert futtató számítógépet is kompromittálhatja. A médiában is nagy visszhangot kiváltó események, mint például kreditkártya információk megszerzése illetve személyes adatok kiszivárogtatása, gyakran SQL injekcióra épülve jönnek létre.

Jelen publikáció célja az adatbázisok biztonságát fenyegető támadások egyik fontos típusának, az SQL injekciónak a felvázolása, elemezve a támadás alapjait, típusait és az ellene való védekezés lehetőségeit. Az 1. fejezetben bemutatom az SQL injekciós támadások fogalmát, lényegét és a következményeinek típusait, a 2. fejezetben a támadás elleni egyik legrégebbi védekezés következményeként kialakult típust, a bekötött szemű SQL injekció sajátosságait, módszereit ismertetem, a 3. fejezetben elemzem az SQL injekció elleni védekezés lehetőségeit, majd a 4. fejezetben összegzem a tárgyalt információkat.

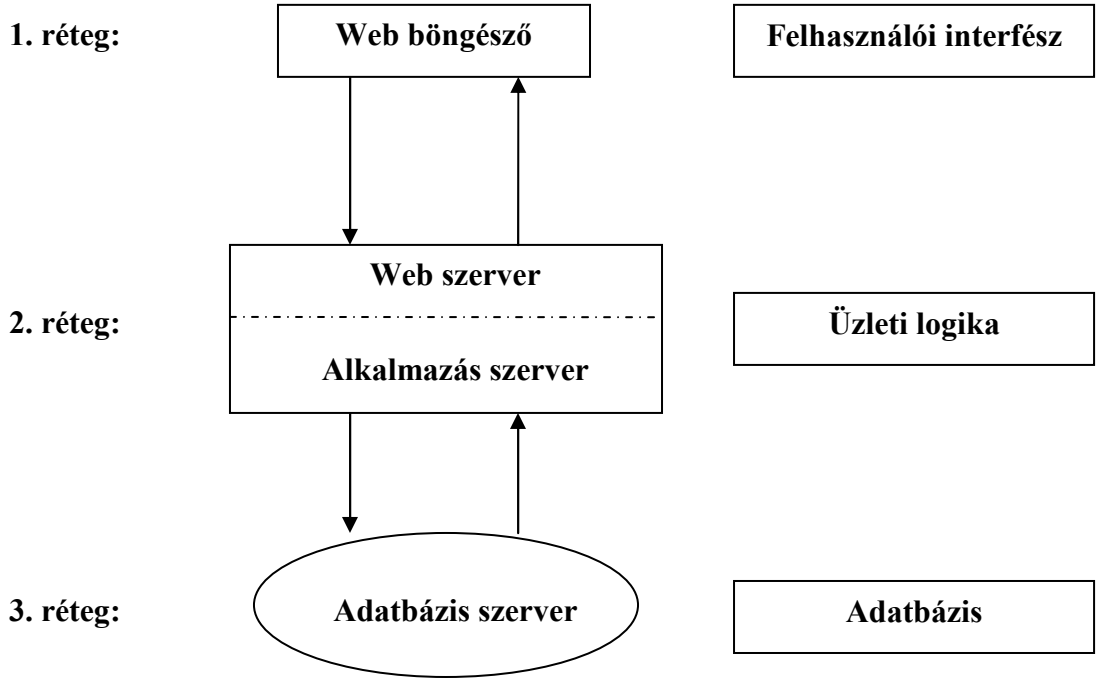
SQL injekció bemutatása

Az SQL injekciós támadás lényege egy tervezett tartalmú, dinamikusan szerkesztett SQL utasításba káros tevékenységeket megvalósító SQL utasítások beillesztése (hozzáfűzése). Ezen típusú támadás az adatbázisokra épülő alkalmazásoknak, különös tekintettel a webes alkalmazásoknak a sérülékenységét kihasználva valósul meg. A sérülékenység oka mögött legtöbbször magának az alkalmazásnak a programozási hibája áll, nem pedig az alkalmazás mögött álló adatbázis környezet vagy annak telepítési konfigurációja. Ez is indokolja, hogy az SQL injekció elleni védekezést nem kizárólag az informatikai infrastruktúra biztonságának védelmében kell keresni, fontos magára az alkalmazásra és annak kódjára is figyelmet szentelni.

Azt, hogy a publikációban tárgyalt információs támadás napi jelentőséggel bír a következőkkel támasztom alá. A jelentősebb számítógépes sebezhetőségeket, hibákat adatbázisokban tárolják, például a MITRE által üzemeltetett és karbantartott CVE (Common Vulnerabilities and Exposures) lista, amit az USA Belbiztonsági Minisztériuma szponzorál. A CVE adatbázis az ismert, napvilágra került sebezhetőségek leírását tartalmazza, azokhoz egyedi azonosítót (CVE number) rendelve és megcélozva azt, hogy a sebezhetőségek elnevezéseit illetően szabvánnyá váljon az őket tartalmazó adatbázisok között. A CVE listában tárgy szerint SQL injekcióra keresve körülbelül 200 bejelentett sérülékenységet kapunk a 2008-as évben.

A pontosabb megértéséhez először egy ábra segítségével bemutatom az adatbázis háttérrel rendelkező webes alkalmazások architektúrájának 3 szintjét, majd ismertetem a támadás lényegét. Felvázolom a támadás céljainak három különböző típusát. Utána 2 konkrét példát mutatok tipikus SQL injekciós támadásra a felhasználói paraméterek és az általuk előállított SQL lekérdezés szemléltetésével.

A webes alkalmazások architektúrájának legfelső szintjén felhasználói interfész szerepét ellátó web böngésző található. A középső szinten az üzleti logikát képviselő web szerver és alkalmazás szerver állnak. A web szerver a listener szolgáltatás segítségével fogadja a kliens gépen lévő böngésző által küldött kéréseket, amiket az általában ugyanazon fizikai gépen lévő alkalmazás szerverhez továbbít. Az alkalmazás szerver SQL utasítások elküldése útján kezdeményezi az adatbázisban tárolt adatok elérését az architektúra legalsó szintjén lévő adatbázis szervertől.



1. ábra: A 3-rétegű webes alkalmazás architektúrájának felépítése

Az alkalmazás a felhasználótól bekért paraméterek segítségével állítja elő az SQL szerverhez eljuttatandó lekérdezést, amit egyetlen sztring fog képviselni. A támadás lényege abban rejlik, hogy az alkalmazáson keresztül meg lehet változtatni az SQL lekérdezés szintaktikáját, mivel egyetlen sztring tartalmazza a lekérdezési utasítás szintaktikáját és a felhasználó által megadott paramétereket. Ezt a sztringet küldi el az alkalmazás az adatbázis szerverhez, általában input paraméterek ellenőrzése nélkül. A támadó a paraméter értékének olyan kompromittáló karaktersorozatot ad meg, ami megváltoztatja az eredeti lekérdezés szintaktikáját, ezáltal a támadó egészen más feladatot valósít meg, mint amit a programozó a kódjával szándékozott volna. Meg kell említeni, hogy a dinamikusan előállított utasítás nem csak a felhasználótól bekért adatokból építkezhet, hanem a kliens gépen futó böngészőn megjelenő űrlapok rejtett mezőiből, továbbá a szerver oldali munkamenet vagy alkalmazás változóiból, mert ezek sütibe (cookie) ágyazva megtalálhatóak a felhasználók gépén, és viszonylag könnyen hamisíthatók.

Az SQL injekciós támadás bármely adatbázis platformon végrehajtható. A támadást a célja szerint a következő három típusba sorolhatjuk:

1. Adatokhoz való jogosulatlan hozzáférés

A támadó az alkalmazás kijátszása által eléri, hogy az adatbázisból számára nem megengedett adatokat kérdezzen le, beleértve jelszavakat és felhasználói azonosítókat is. A jogosulatlan hozzáférés gyakran a felhasználói hitelesítés kijátszásával jön létre.

2. Adatbázis módosítása

A támadó adatokat szúrhat be, módosíthat, illetve törölhet ki az adatbázisból, illetve megváltoztathatja az adatbázis struktúráját, sémáját. Azonosítók, jelszavak, adatbázis hozzáférések létrehozása, módosítása és törlése is ebbe a kategóriába tartozik.

3. Adatbázis szervert futtató számítógép kompromittálása
SQL injekcióval el lehet érni új felhasználó felvételét az adatbázis szervert futtató gépen, ami által a támadó a számítógéphez hozzáférési jogot tud nyerni és akár operációs rendszer szintű parancsokat hajthat végre SYSTEM jogosultsággal.

A támadás pontosabb megértéséhez nézzük a [1]-ben ismertetett példákat:

1. Példa: Hitelesítés kijátszása

Az alkalmazás kódrészlete:

```
SqlQry = "SELECT * FROM Users WHERE Username = '" &
Request.QueryString("User") & "' AND Password = '" &
Request.QueryString("Pass") & "'"

LoginRS.Open SqlQry, MyConn

If LoginRS.EOF Then Response.Write("Invalid Login")
```

Jóhiszemű felhasználó esetén, John felhasználó név és Smith jelszó megadása utána következő SQL lekérdezés jön létre:

```
SELECT * FROM Users WHERE Username = 'John'  AND Password =
'Smith'
```

A támadó jelszónak megadja a X' OR '1'='1 sztringet, akkor a következő SQL lekérdezés keletkezik:

```
SELECT * FROM Users WHERE Username = 'John' AND Password = 'X' OR
'1'='1'
```

Ez az adatbázis lekérdezés tetszőleges felhasználó név megadása mellett a teljes Users tábla tartalmát kilistázza, ami teljesen különbözik attól, ami a programozó szándéka lett volna.

2. Példa: Adatok megszerzése UNION SELECT típusú támadással

Az SQL nyelvben a UNION SELECT utasítás lehetőséget ad arra, hogy több táblából hasonló információkat kérdezzünk le és fűzzünk össze. A két (vagy több), helyesen felépített, UNION-nal összefűzött SELECT utasítás oszlopneveinek nem kell egyeznie, viszont a SELECT-eknek azonos számú és típusú attribútumokból kell felépülniük. A UNION SELECT SQL utasítás segítségével súlyos kiaknázás jöhet létre, példának tekintsük a következő kódrészletet:

```
SqlQry = "SELECT * FROM Products WHERE ProdDesc LIKE " & "'%"
Request.QueryString("SearchTerm") & "%'"

ProdsRS.Open SqlQry, MyConn
```

A következő SQL lekérdezés jön létre a matrix paraméter megadása esetén:

```
SELECT * FROM Products WHERE ProdDesc LIKE '%matrix%'
```

Azaz kiválasztja (majd megjeleníti a böngészőben) azokat a termékeket, melyek neveiben szerepel a „matrix" szótöredék.

Ha a támadó a következő sztringet adja meg paraméternek:

```
XXX' UNION SELECT 1, 1, Username + ' : ' + Password, 1, CCNumber, 1
FROM Users --
```

A következő SQL lekérdezés keletkezik:

```
SELECT * FROM Products WHERE ProdName LIKE '%XXX' UNION
SELECT 1, 1, Username + ' : ' + Password, 1, CCNumber, 1 FROM Users --
%'
```

Ez az utasítás kiválasztja (majd megjeleníti a böngészőben) azokat a termékeket, melyek nevei „XXX"-re végződnek (valószínűleg nem lesz ilyen találat), továbbá a felhasználók neveit, jelszavait és kreditkártya számait.

Bekötött szemű SQL injekció

Az SQL injekciók elleni védekezés legrégebbi formája a böngészőben megjelenő adatbázis szerver hibaüzeneteinek letiltása és helyettesítése egy felhasználóbarát, általános üzenettel. Az ilyen feltételek melletti SQL injekciókat bekötött szeműnek (angolul blindfolded-nek) nevezzük. Az adatbázis üzenetek információt tartalmaznak az adatbázisról, amiket a támadó ügyesen ki tud használni a sikeres támadáshoz szükséges bemeneti paraméterek elkészítéséhez. Az üzenetek letiltását egy egyszerű web konfigurációs beállítással lehet elérni. Ez a módszer azonban nem szolgál teljes védelemmel, igazából a sérülékenység eltitkolását próbálja elérni, miközben a támadás csak bonyolultabbá, de nem lehetetlenné válik.

A következőkben [1] alapján szemléltetem azt a technikát, amivel sikeres bekötött szemű SQL injekciókat lehet felderíteni és végrehajtani. A módszer lényege tesztek lefuttatása bizonyos dolgok kiderítése érdekében. A tesztek lefutása során vizsgáljuk, hogy hibába ütköztünk-e vagy sem, majd ebből vonunk le következtetéseket a sikeres SQL injekció végrehajtása érdekében. Ezek segítségével:

1. SQL injekcióra épülő támadás lehetőségét tudjuk felderíteni

2. Kitalálhatjuk az injekció szintaktikáját és az adatbázis típusát

3. UNION SELECT típusú támadást tudunk felépíteni

4. Ezek után más típusú injekciós támadásokat (pl. WHERE feltételre épülő vagy utasítás befecskendező) már könnyű megalkotni.

1. lépés: injekciós támadás lehetőségének feltérképezése

Ebben a lépésben a tesztjeink célja az, hogy megállapítsuk, tudunk-e egy adott webes alkalmazás esetén egyáltalán SQL injekciós támadást kezdeményezni. Ehhez az alkalmazás adatbemeneti mezőjét furfangosan töltjük ki, majd megfigyeljük, hogy ez hibát generál-e. Például:

- Az **5** helyett **(6-1)**-et írunk
- A ’teszt’ karakter sorozat helyett MS SQL-ben ’te’+’szt’, Oracle-ben pedig ’te’||’szt’ inputokat írunk (ez a teszt az adatbázis platformjáról is információt nyújthat)
- Dátum helyett az adatbázis dátum függvényét írjuk be.

Ha a tesztekben az összetartozó párok esetén ugyanazt az eredményt kapjuk, akkor megtudtuk, hogy az alkalmazás sérülékeny az SQL injekcióra, majd a 2. lépés következik. Ha hibaüzenetet kapunk, akkor tudjuk, hogy az inputunk nem lett átadva az SQL elemzőnek, azaz bemeneti paraméter ellenőrzést tartalmaz az alkalmazás kódja, tehát SQL injekcióra nincs lehetőség.

2. lépés: a támadáshoz szükséges szintaktika felépítése

Célunk a támadáshoz megfelelő szintaktika megtalálása méghozzá a SELECT utasítás WHERE feltételen belül. Meg kell állapítanunk, hogy a WHERE feltételben tudjuk-e használni a már megismert *OR ‘1’=‘*1 trükköt, tudjuk-e a megjegyzés jelet (--) használni a lekérdezés felépítésében, kell-e zárójelek beépítésével játszani. Teszteléskor az inputhoz hozzáfűzzük például az

- OR 1=2 vagy AND 1=1 sztringeket,
- Megjegyzés sztringet (--),
- Nyitott zárójeleket.

Ezután megfigyeljük, hogy kapunk-e hibát. Néhány próbálkozás után a WHERE feltétel szintaktikáját felderítettük, majd kedvünkre manipulálhatjuk azt.

3. lépés: UNION SELECT típusú kiaknázás felderítése

A UNION SELECT szintaktika használatához ismernünk kell az eredeti lekérdezésben szereplő attribútumok számát és típusát. SQL nyelvben háromféle alaptípusról beszélhetünk, (szám, sztring és dátum), ezeket kell majd az egyes attribútumok esetében meghatároznunk.

- Az attribútumok számának megállapítása:
  Az attribútumok számának megállapításához az ORDER BY záradékot hívjuk segítségül, ami az eredménylistában rendezést valósít meg, méghozzá a megadott attribútum alapján. Az attribútumot kétféleképpen lehet megadni, vagy a nevével vagy pedig a lekérdezésben betöltött helyének a sorszámával. Mi az utóbbit fogjuk használni.
  Például, ha ORDER BY 6 esetén hibát kapunk, míg ORDER BY 5 esetén nem, levonhatjuk a következtetést, hogy 5 darab attribútum szerepel a lekérdezésben.

N attribútum esetén $\log_2 N$ tesztre van szükségünk az attribútumok számának eltalálásához.

- Az attribútumok típusának megállapítása:
  Először NULL értéket állítunk be az attribútumok típusának, ez minden esetben helyes lesz, majd egyesével próbáljuk a típusokat kitalálni. Az attribútum helyett 1-et, illetve '1'-et írva hibaüzenet hiányában szám, illetve sztring típusról van szó. N attribútum esetén 3*N tesztre van szükségünk az attribútumok típusának kitalálásához.

4. lépés:

Az előző pontokban felvázolt technikával összegyűjtött információk birtokában más típusú injekciós támadásokat (pl. WHERE feltételre épülő vagy utasítás befecskendező) viszonylag könnyűszerrel végre lehet hajtani.

SQL injekciós támadások elleni védekezés

Ebben a részben bemutatom az SQL injekciós támadás elleni védekezési lehetőségeket, azokat három kategóriába csoportosítva aszerint, hogy az informatikai infrastruktúra melyik szintjén valósulnak meg

Az SQL injekcióra épülő támadás az alkalmazásban rejlő biztonsági rést használja ki lehetőséget adva arra, hogy a felhasználó által megadott bemeneti paraméter a szükséges formai ellenőrzés nélkül kerüljön be az adatbázis lekérdezést meghatározó sztringbe. Az alkalmazások fejlesztői általában nem információ biztonsági szakemberek, a program írása közben nem biztonsági kérdésekre koncentrálnak, inkább az idő korlátra és a helyesen futó kódokra helyezik a figyelmet. Ez jelentősen hozzájárul ahhoz, hogy az általuk írott programok biztonsági réseket tartalmaznak. A hibákat és problémákat sokszor nem ők, hanem az IT biztonsággal foglalkozó szakemberek észlelik és próbálják utólagos eszközökkel megoldani. A hálózat szintjén megvalósítható védekezés eszközei, mint például a tűzfalak és a hálózati szintű IPS, IDS rendszerek nem nyújtanak megfelelő védelmet, hisz ezen a szinten a kiaknázást megvalósító, rosszindulatú lekérdezés nem különböztethető meg a normális típusútól. A teljes biztonságot az jelentené, ha már a kódok írásakor tudatosan tervezés és megvalósítás történne. Ebben a fejezetben megvizsgáljuk, hogy az említett két szakterület a maga szintjén milyen védekezési módokkal tud élni.

Egy másik érdekes kérdés, hogy a támadások elleni védekezésben gyakran használt behatolás érzékelő (IDS) és behatolás megelőző (IPS) rendszerek az adatbázis védelemben milyen szerepet tudnak betölteni, és ehhez milyen technikát használnak. Bemutatok három különböző módszerre alapozott behatolás érzékelő és behatolás megelőző rendszert. Adatbázis biztonság szempontjából alkalmazott IDS és IPS rendszerek a megfigyelő pontjaikat, más néven szenzoraikat nem a hálózati szinten, hanem az architektúrában magasabban elhelyezkedő adatbázis szerverben vagy a szerver előtti proxy-n helyezik el. A szenzort úgy építik fel, hogy az SQL adatfolyamot figyel meg.

Alkalmazás szintjén megvalósítható védekezés

Az előzőekben már láttuk, hogy kiemelten fontos az alkalmazások programkódjának készítésekor tudatában lenni a biztonsági kérdéseknek, illetve érdemes meghatározott technikákat használni a sérülékenység kiküszöbölése érdekében. A [2], [3], [4], [5] munkákban különböző programozási nyelvekhez kötötten (pl. java, php, asp.net) találunk SQL injekciós támadásra példákat, majd az egyes nyelvek esetén kivédési módszereket találunk, melyek a helyes programozási gyakorlatot szemléltetik. A konkrét programozási

nyelvektől függetlenítve a támadás kiküszöbölése érdekében a következő programozási technikákat célszerű ismerni és alkalmazni:

Alkalmazás számára megadott inputok ellenőrzése

Az inputokat ellenőrizni kell típus, hossz, formátum és tartomány alapján. Érdemes meghatározni a lehetséges helyes karakterek halmazát, majd ettől eltérő bemenő karakter esetén az adatbázis lekérdezését meg kell szakítani. A szakirodalom kiemeli, hogy fontos a helyes karakterek megállapítása, szemben a helytelenek halmazával, mivel az utóbbi sokkal tágabb, ezért annak a valószínűsége is nagyobb, hogy valamit kifelejtünk közülük. A web illetve alkalmazás szerver szintjén reguláris kifejezések és ezek rutinjai segítségével lehet az input ellenőrzést végrehajtani. Azaz léteznek a programozási nyelvek által nyújtott előre megírt eljárások és függvények, amiket az alkalmazás programozója felhasználhat a bemeneti változók értékeinek érvényesítésére.

Tárolt eljárások és dinamikus SQL utasítások számára az input értékét ne közvetlenül, hanem paraméterek formájában adjuk meg, aminek az értékét le lehet ellenőrizni a tárolt eljárás vagy dinamikus utasítás futtatása előtt, ezzel kivédhetjük a rosszindulatú kód befecskendezését.

Ismert módszer az eszképelés használata is, amivel a veszélyes karakterek, például idézőjelek becsempészését tudjuk elkerülni. Az inputban található egyszeres idézőjel veszélyes lehet, mert rosszindulatú kód bevezetésére adhat lehetőséget. A programozási nyelvek rendelkeznek eszképelő rutinokkal (például MySQL nyelvben a mysql_real_escape_string() függvény), melyek az inputot úgy alakítják át, hogy a veszélyes karaktereket megfelelőekkel helyettesítik (például az egyszeres macskakörmöket megduplázzák).

Az adatbázis szerver hibaüzeneteinek elrejtése

Nem célszerű hibaüzenetekben adatbázisra jellemző információt - különösen szerkezetit – kiírni. A bekötött szemű SQL injekció tárgyalásakor azonban láttuk, hogy ez a védekezési módszer nem szolgáltat teljes védettséget.

Adatbázis szerver szintjén megvalósítható védekezés

Adatbázis hozzáférés helyes beállítása

Ahhoz, hogy egy alkalmazás az adatbázist elérhesse, létre kell hozni az adatbázis szerveren egy adatbázis felhasználót, akinek a nevében ő az adatbázist meg tudja szólítani. Minden adatbázis alapból egy felhasználó tulajdona lesz, méghozzá azé, aki a létrehozó utasításokat lefuttatta. Annak érdekében, hogy az adatbázishoz a tulajdonoson és a superuseren kívül mások is hozzáférjenek, jogosultsággal kell őket ellátni. Az alkalmazásnak nem szabad tulajdonosként vagy superuserként csatlakoznia az adatbázishoz, mert ezek bármilyen utasítást és lekérdezést tetszés szerint futtathatnak, pl. a szerkezeti módosítást (táblák megszüntetése) vagy táblák komplett törlése.

Mindig a lehető legkevesebb jogosultsággal rendelkező, testreszabott felhasználókat célszerű használni, melyek mindegyike az adatbázis manipulációnak egy-egy különböző nézőpontjáért felelősek. El kell kerülni, hogy különböző alkalmazások számára ugyanazt a felhasználót használjuk egy adott adatbázis eléréséhez. Tehát minden alkalmazás számára önálló felhasználót hozzunk létre az adatbázis szerveren az adatok eléréséhez. Ekkor, ha a behatoló meg is szerzi valamelyik jogosultságot (hitelesítési információt = felhasználói név + jelszó), akkor is csak akkora változást tud okozni, mint az alkalmazás maga. Célszerű továbbá az adatbázis kezelő rendszer összes alapértelmezett felhasználónevét és a hozzájuk tartozó

jelszavakat törölni, ha szükségünk van ugyanolyan jogkörű felhasználóra, akkor hozzuk azt létre magunk új névvel és jelszóval. Töröljük ki a beépített táblákat és tárolt eljárásokat is.

Audit logok használata

Fontos feladat a lekérdezések, hozzáférések naplózása az üzleti logika szintjén, hisz az adatbázis szerveren egyetlen, az alkalmazáshoz kötött felhasználó jelenik csak meg. Nyilvánvalóan a naplózás nem tud megakadályozni egyetlen ártalmas próbálkozást sem, de segítséget nyújthat annak felderítésében, hogy melyik alkalmazás és ki által lett kijátszva. A naplózás megtervezésénél át kell gondolni, milyen információkat tárolunk el. Általánosságban elmondható, hogy minél több részletet jegyzünk, annál biztonságosabb a rendszerünk.

IPS és IDS alapú védekezés

A behatolás-érzékelő rendszerek (IDS) olyan hardver vagy szoftver eszközök, melyek érzékelik a lehetséges behatolásokat, támadásokat, majd értesítik a megfelelő személyeket, esetleg válaszlépéseket tesznek (részletesebben lásd pld. [6], [7], [8]). Ezek a rendszerek figyelik a számítógépeken zajló folyamatokat, forgalmat, gyanú esetén saját szabályrendszerük alapján eldöntik, hogy egy adott tevékenység a védett rendszeren illegális tevékenységnek minősül-e. Pozitív válasz esetén a rendszer valamilyen formában értesíti a felhasználót vagy a rendszergazdát. Az IDS rendszereket két nagy csoportba oszthatjuk az alapján, hogy a kiértékelendő információt milyen módon gyűjtik össze. A hálózat-alapú IDS-ek a számítógép hálózat forgalmát monitorozzák, míg a hoszt-alapú IDS-ek számítógépeken futnak és az operációs rendszer, illetve a gépen futó alkalmazások viselkedését figyelik. SQL injekciós támadások elleni védekezésben az utóbbiak használatosak és a továbbiakban ebbe a kategóriába eső IDS-eket fogunk vizsgálni.

A behatolás-megelőző (IPS) rendszerek proaktívan működnek, a támadás megelőzése a céljuk az IDS-eknél megismert technológiákat használva. AZ IPS rendszereknek is létezik hoszt- és hálózat-alapú változata. Az IDS és IPS rendszerek közötti leglényegesebb különbség a következő. Míg az IDS rendszerek megfigyelik a számítógép folyamatait és csak korlátozott beavatkozásra képesek, addig az IPS rendszerek a támadás kialakulása előtt beavatkoznak. Jelen publikációban a kétfajta rendszert az SQL injekció elleni védekezés szempontjából ugyanannak a kategóriának tekintem.

Két fő technológia létezik az események analizálására, a támadások észlelésére. Az egyik a visszaélést érzékelő modell, a másik a rendellenességet érzékelő modell.

A visszaélést érzékelő modell működése során azáltal elemzi a rendszer folyamatait, hogy események lenyomatait keresi és hasonlítja össze a saját adatbázisában tárolt támadás lenyomatokkal. Mivel ezeket a speciális támadás lenyomatokat angolul „signature”-nek hívják, ezért ezt a modellt „signature based” IDS-nek is nevezik. Ezek a rendszerek a támadás módját előre kell, hogy ismerjék, ismeretlen incidensek ellen nem védenek. Továbbá a lenyomatokat tároló adatbázisukat folyamatosan frissíteniük kell ahhoz, hogy hatékonyan tudjanak működni. SQL injekció kiszűréséhez az adatbázis szervernek küldött SQL utasítást vizsgálja az IDS és abban tipikus injekciós mintákat keres, amik közül néhányat a következő táblázatban mutatok be.

OR 1=1
UNION SELECT
--
EXEC XP_CMDSHELL

A lenyomatokat használó IDS-eket gyakran a web alkalmazások részeként valósítják meg és nem az adatbázis szerver elé helyezik el. Ezen típusú behatolás érzékelők két hátrányát fontos megemlíteni. Az egyik, hogy az injekciós sztringet előre ismernie kell a rendszernek és annak szerepelnie kell a lenyomat adatbázisban, a másik pedig a [9] publikációban részletesen ismertetett módszer, ami segítségével egy ügyes támadó ki tudja játszani a védelmet. A kijátszás alapulhat szóköz karakterek beszúrására, az SQL utasítás részekre vágására, vagy a lenyomattól különböző, de hatásában egyenértékű SQL utasítás használatára. A következőben megnézünk néhány példát a kijátszás módszerére:

A közismert OR 1=1 beszúrására épülő támadást ennek a karaktersorozatnak a szűrésével próbálják védeni. Ha a támadó ennek tudatában van, akkor ugyanezt a típusú támadást végrehajthatja úgy, hogy az 1 helyett tetszőleges sztringet szúr be, például:

OR ’Simple’ = ’Simple’

Ha a védelmi profilon ez is fennakad, akkor lehet próbálkozni, az N karakter beszúrásával a második sztring elé, hisz ez az SQL szerver felé csak azt az információt szolgáltatja, hogy nvarchar típusú érték következik:

OR ’Simple’ = N’Simple’

Ezt a támadást már csak bonyolultabb lenyomat definiálásával lehet kivédeni és eddig már gyakran nem jutnak el az IDS lenyomat adatbázisok. A következőkben felsorolok még néhány injekciós mintát, amivel a kiindulási támadást ki lehet váltani:

OR ’Simple’ = ’Sim’+’ple’

OR ’Simple’ LIKE ’Sim%’

OR ’Simple’ < ’S’

Rendellenességet érzékelő modell a számítógépen bekövetkező nem normális jelenségeket, anomáliákat figyeli (angolul anomaly based model-nek hívják). Az IDS működését két fázisra oszthatjuk. Az elsőben megfigyeli, megtanulja, hogy egy adott környezetben mi számít normál működésnek, majd következik a második fázis, amikor is a normálistól eltérő eseményeket figyeli és észlelésük esetén riaszt. A normális működés megtanulása, meghatározása többféle technikára (például statisztikai számításokra, adatbányászatra, mesterséges intelligenciára) épülve valósulhat meg, ezek jelenleg folyó kutatások tárgyát is képezik [8].

Az SQL injekció védelmére kifejlesztett IDS-ek az adatbázis szerverhez kiadott SQL lekérdezéseket figyelik, és ennek alapján alakítanak ki felhasználói vagy működési profilokat, majd a második, detektáló fázisban azt vizsgálják, hogy érkezik-e az AB szerverhez olyan

lekérdezés, ami nagyon eltér a megállapított profiltól. Ilyen esetben riasztással, vagy visszautasítással válaszolva próbálják a védelmet megvalósítani.

A küszöbérték figyelésén alapuló technika lényege, hogy a rendszer küszöbértékeket rendel a „normális” tevékenységekhez és az IDS ezen értékek figyelésével végzi feladatát. Ezek a számok lehetnek statikus, tapasztalati úton beállított számok, de lehet valamilyen heurisztikán alapuló számítás eredménye is. A küszöbértékek figyelésén alapuló IDS egy könnyen érthető és parametrizálható rendszer. Hátránya viszont az, hogy nehéz feladat a küszöbértékek helyes beállítása. A rosszul beállított rendszer sok hamis pozitív vagy hamis negatív eredményhez vezethet.

Felhasználói profil készítése esetén a rendszer minden egyes felhasználóhoz létrehoz egy felhasználói profilt. Ebben a profilban a felhasználó mindennapi tevékenységei, a felhasználótól „elvárható” események feljegyzései vannak. Ha a felhasználó megváltoztatja a mindennapi ténykedését, akkor a profilja követi a változásokat. Ha a profil és az aktuális cselekmény között jelentős eltérést tapasztal a rendszer, akkor riasztást küld.

A mézesmadzag (angolul honeytoken vagy honeypot) betörés-detektálók az információs rendszerben, például esetünkben, az adatbázisban „csali” elhelyezésével (pl. híres emberek rekordjainak szerepeltetésével) és azok lekérdezésének figyelésével szűri ki a betöréseket, vagy az arra irányuló próbálkozásokat. [10] Az adatbázisban olyan rekordot elhelyezésére kerül sor, amihez nem tartozik valós információ, de jellegéből adódóan kíváncsiság felkeltésére kiválóan alkalmas. Ha egy kíváncsi felhasználó a rekord hozzáférését kezdeményezi és az IDS figyeli a rekord történetét, akkor észlelheti, hogy a rendszerben szabálytalan hozzáférést hajtottak végre. Főleg a szervezeten belülről érkező támadások kiszűrése esetén működhet hatékonyan. Ezen technika előnye, hogy kevés, de releváns adattal dolgozik, alacsony erőforrásigényekkel rendelkezik, nem jelent számára problémát a kódolt csatornák használata. A honeypot rendszereknek természetesen hátrányai is vannak: csak annál a forgalomnál jeleznek behatolást, aminek ők voltak a címzettjei, vagyis önmagában nem alkalmasak teljes körű behatolás detektálásra.

Összegzés

A publikációban bemutattam az adatbázis háttérrel rendelkező alkalmazások biztonságát fenyegető támadások egyik fajtáját, az SQL injekciót. Az SQL injekciós támadás dinamikusan szerkesztett SQL utasításba illeszt káros tevékenységet megvalósító kódot, a behatolás az adatbázisokra épülő alkalmazásoknak, különös tekintettel a webes alkalmazásoknak a sérülékenységét kihasználva valósul meg. A támadó megszerezheti az alkalmazás mögött álló teljes adatbázis tartalmát, megváltoztathatja az adatbázis felépítését, illetve tartalmát, sőt az adatbázis szervert futtató számítógépet is kompromittálhatja. A sérülékenység oka mögött legtöbbször magának az alkalmazásnak a programozási hibája áll, nem pedig az alkalmazás mögött álló adatbázis környezet vagy annak telepítési konfigurációja. A programozási hiba lényege, hogy a kliens oldalról érkező adatokat ellenőrzés nélkül dolgozza fel az alkalmazás, így lehet az eredeti szándéktól eltérő, kártékony kódot végrehajtatni a rendszerrel.

Az SQL injekcióra bemutatott és az irodalomban található példák többségében a támadónak rendelkeznie kell valamennyi előzetes információval az adatbázis felépítéséről, illetve az alkalmazás kódjáról. Abból kell azonban kiindulni, hogy a behatolók különböző forrásokból megszerezhetik a szükséges információkat, tehát nem nyilvános voltuk nem nyújt elégséges védelmet a támadások ellen.

Az SQL injekciós támadás elleni védekezés az architektúra több szintjén is történhet. A hálózati rétegben nincs igazán megfelelő eszköz a prevencióra. Az alkalmazás szintjén a kliens oldali inputokat ellenőrizni kell típus, hossz, formátum és tartomány alapján, valamint célszerű elrejteni az adatbázis szerver hibaüzeneteit a felhasználó elől. Az adatbázis szerver szintjén figyelni kell az alkalmazás számára megvalósított hozzáférés beállítására. Csak a

legszükségesebb jogokkal rendelkező hozzáférést célszerű az alkalmazás számára biztosítani. Kiemelt szerepe van a lekérdezések naplózásának az alkalmazás szintjén, hisz az adatbázis szerveren a hozzáférési logokban csak egyetlen, az alkalmazáshoz kötött felhasználó szerepel. Az alkalmazás, illetve az adatbázis szerver szintjén megvalósítható behatolás-érzékelő (IDS) és behatolás-megelőző rendszerek (IPS) érzékelik a lehetséges támadásokat, majd értesítik a megfelelő személyeket, esetleg válaszlépéseket tesznek. Két fő technológia létezik az események elemzésére, a behatolások kiszűrésére, a visszaélést érzékelő modell és a rendellenességet érzékelő modell.

Összegzésképpen megállapítható, hogy a támadások alapvetően olyan programok kijátszásán alapulnak, amelyek a védelmet, illetve biztonságot figyelmen kívül hagyva születtek. Ezért alapvetően szükséges, hogy a fejlesztők körében is az IT biztonság egy hangsúlyos terület legyen. Programozás során soha nem szabad megbízni semmilyen bejövő adatban, főleg ha az a kliens oldalról érkezik, még akkor sem, ha az egy alkalmazás szerver oldalról megadott süti, vagy rejtett mező (hidden input) értéke esetleg egy legördülő lista eleme.

Felhasznált irodalom

1. A. Shulman: Traditional SQL Injection Protection:. The Wrong Solution for the Right Problem. CTO. Imperva™ Inc. http:webcourse.cs.technion.ac.il/236350/Spring2005/ho/WCFiles/AdvancedSQLInjection.pdf
2. http://szabilinux.hu/php/security.database.html
3. J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Andy Wigley: How To: Protect From SQL Injection in ASP.NET. http://msdn.microsoft.com/en-us/library/ms998271.aspx, 2005
4. A. Agarwal: SQL injection: Developers fight back. http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1179106,00.html, 2006
5. S. Friedl: SQL Injection Attacks by Example. http://www.unixwiz.net/techtips/sql-injection.html, 2007
6. Frank S. Rietta: Application layer intrusion detection for SQL injection. ACM Southeast Regional Conference 2006: 531-536.
7. Ficsor Péter:Incidenskezelő rendszerek (Összehasonlító elemzés). Szakdolgozat 2005 Eötvös Loránd Tudományegyetem Informatikai Kar
8. F. Valeur, D. Mutz, G. Vigna: A learning-based approach to the detection of SQL attacks. In: Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Vienna, Austria, July 2005
9. O. Maor, A. Shulman. Sql injection signatures evasion: An overview of why sql injection signature protection is just not enough. http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html, 2004.
10. L. Spitzner. Honeytokens: The other honeypot. http://www.securityfocus.com/infocus/1713, July 2003.