Szegedi Tudományegyetem

Informatikai Intézet

**SQL injection detektálás természetes nyelvi feldolgozó rendszerek és gépi tanulás által**

**SQL injection detection using natural language processing systems and machine learning**

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Stiller Marianna |  | Dr. Vidács László |  |
|  | programtervező informatika szakos hallgató |  | egyetemi oktató |  |

Szeged

2021

## Feladatkiírás

Az SQL Injection napjaink leggyakoribb és legveszélyesebb sérülékenysége, amely a felhasználói input elégtelen mértékű validációjából ered, következménye pedig SQL lekérdezések futtatása érzékeny adatokon.

A rendszer természetes nyelvi feldolgozó rendszerek és gépi tanulás által képes lesz forrásódban SQL Injection sérülékenységeket keresni. Működésének alapja, hogy valós PHP alapú forráskódokból reprezentál absztrakt szintaxis fákat, melyekből mintákat nyer ki, amit később megtanul.

## Tartalmi összefoglaló

A téma megnevezése:

SQL injection detektálás természetes nyelvi feldolgozó rendszerek és gépi tanulás által.

A megadott feladat megfogalmazása:

Minták által tanított program képes reális becslést adni egy kód SQL injectionbeli sebezhetőségével kapcsolatban.

A megoldási mód:

PHP kódokból generálok absztrakt szintaxis fákat, amelyeket felcímkézek, mint sérülékeny és nem sérülékeny példák. Ezek átesnek egy vektorizáláson, majd egy kifejezett modell segítségével – amely a legkevesebb hibás döntéseket hozza - a program megtanulja felismerni a sérülékenység jeleit. Ennek hatására egy nem tanuló kódról képes kiértékelést készíteni.

Alkalmazott eszközök, módszerek:

A IntelliJ fejlesztői környezetet, emellett a gépi tanulás során Python, a grafikus asztali felület megvalósításához Java programozási nyelvet alkalmaztam.

Elért eredmények:

A program a bemenetként megadott PHP kódra kimenetként egy százalékos értéket ad, amely a kód sérülékenységének mértéke.

Kulcsszavak:

Felügyelt tanulás, train/test készlet, K-Fold, confusion mátrix, recall.

## Tartalomjegyzék

[Feladatkiírás 2](#_Toc70961896)

[Tartalmi összefoglaló 3](#_Toc70961897)

[Tartalomjegyzék 4](#_Toc70961898)

[BEVEZETÉS 5](#_Toc70961899)

[1. SQL INJECTION 6](#_Toc70961900)

[1.1. Típusai 6](#_Toc70961901)

[1.1.1. Sávon belüli (in-band, classic) SQLi 6](#_Toc70961902)

[1.1.2. Vak (blind, inferential) SQLi 7](#_Toc70961903)

[1.1.3. Sávon kívüli (out-of-band) SQLi 8](#_Toc70961904)

[1.2. Megelőzési módjai 8](#_Toc70961905)

[2. GÉPI TANULÁS 11](#_Toc70961906)

[3. PROGRAM FELÉPÍTÉSE 12](#_Toc70961907)

[4. EREDMÉNY ÉS ANALÍZIS 13](#_Toc70961908)

[Irodalomjegyzék 14](#_Toc70961909)

[Nyilatkozat 15](#_Toc70961910)

[Köszönetnyilvánítás 16](#_Toc70961911)

## BEVEZETÉS

A mai világban, az informatika világában, a digitalizálás visszafordíthatatlan, illetve nem okszerű. Az informatika meglehetősen gyors fejlődése miatt, egy generáció már születésétől kezdve természetesnek vélheti az internethasználatot. A társadalom kiélvezi az összes kényelmi funkcióját, lebegnek a felszínen, a boldog tudatlanságban, az adataik pedig másodpercről másodpercre szivárognak le, a felszín alá. Az információk megtartása és biztonsága ezáltal nyert a tudományágában külön területet. Mivel értékesebbek lettek, mint a pénz maga.

Információbiztonságra szakosodó programtervező informatikusként állítom, hogy a szoftverfejlesztés nem minden. A bemeneti mezőket nem elég implementálni, meg is kell védeni őket. Ehhez pedig már több módszer van, annál is több példával. A programom mesterséges intelligencia segítségével ezeknek egy elenyésző részét használja fel a tanuláshoz és annak hasznosításához lehetőséget adva ezzel a fejlesztőknek arra, hogy biztonságos programkódot helyezzenek a „használók” elé.

A témaválasztásom egyszerű célja, hogy egy olyan kezdetleges rendszert készítsek, amely lehetőséget ad megvédi azt, ami a pénznél is drágább. Bár az injektálás csak egy része a biztonsági kockázatoknak, ám a legfenyegetőbb. A dolgozatban ki fogom fejteni többek között ezt a fajta sérülékenységet, a tanulási módszereket és a kettő ötvözetét egy programon keresztül, amely kimutatja a biztonságtechnikai fenyegetettséget az SQL injection irányából.

1. SQL INJECTION

A témamegnevezés során meglehetősen kevés időt vett igénybe a konkrét biztonsági kockázat kiválasztása. Szerettem volna egy olyan sérülékenység köré építeni a programom megvalósítását, amelyhez sok példa és ellenpélda tartozik, valamint a dokumentáltsága jobb, mint kiváló. A választásom ez okból esett az első számú kockázat egy fajtájára.

AZ OWASP (The Open Web Application Security Project) listáján, amely a legkritikusabb biztonsági kockázatokat taglalja, az injection (magyarul: befecskendezés, injekció) áll az első helyen. Ez a támadási típus a felhasználói input elégtelen mértékű validációjából ered, amely lehetővé tesz rosszindulatú SQL utasítások végrehajtását, amelyek közvetlenül az adatbázisból kérhetnek le adatokat, emellett adminisztrációs műveleteket hajthatnak végre az adatbázisban, a DBMS fájlban található adott fájl tartalmát helyreállíthatja a rendszerben, és egyes esetekben parancsokat adhat ki az operációs rendszernek. Például az alábbi sor *' UNION SELECT username, password FROM users--* [1] megadása bemeneti paraméterként az alábbi SQL lekérdezést futtathatja amennyiben a bemenet sérülékeny, létezik *users* tábla, valamint *username* és *password* oszlop: *SELECT name, description FROM products WHERE category = ’Gifts’ UNION SELECT username, password FROM users--* [1]*.* A lekérdezés eredménye az adatbázis összes felhasználójának neve és jelszava.

Ebből adódóan kijelenthetjük, hogy az injection napjaink leggyakoribb és legveszélyesebb sérülékenysége.

## Típusai

* + 1. Sávon belüli (in-band, classic) SQLi

Ahogy a korábbi példa szemléltette, a sikeres SQL injection támadás képes érzékeny adatok kiolvasására az adatbázisból. Egyik legfőbb eszköze az *UNION* operátor, amely lehetővé tesz egy vagy több további *SELECT* lekérdezés végrehajtását és az eredmények hozzáfűzését az eredeti lekérdezéshez. Sikerességének feltétele, hogy a két vagy több *SELECT* oszlopszáma és oszloptípusai megegyezzenek vagy konvertálhatóak legyenek páronként.

Adatok kiolvasásán, lekérdezésén kívül képes továbbá adatbázis béli adatok törlésére, módosítására, beszúrására és frissítésére, ha az alábbi sémát vesszük alapul: *105; DROP TABLE Suppliers*[2]. Így a tényleges lekérdezés *SELECT \* FROM Users WHERE UserId = 105; DROP TABLE Susppliers*[2]*;* lesz. Ez a támadási forma a pontosvessző SQL béli funkcióját használja ki halmozott lekérdezések indításához.

További lehetőség, amikor a támadó SQL lekérdezéseken keresztül az adatbázis-kiszolgáló által dobott hibaüzenetekre támaszkodva gyűjt információkat (error-based), ezzel feltérképezve az adatbázis szerkezetét. Például, ha a *https://www.example.beaglesecurity.com/gallery.php?id=6'* [3] URL megadása esetén a szerver a hibát SQL szintaxisban adja vissza, akkor többlépcsős lekérdezésen keresztül az adatok szintén kinyerhetőek.

Ezek a típusok tehát azért „sávon belüliek”, mert a támadó ugyanazt a kommunikációs csatornát képes használni a támadás elindítására és az eredmények gyűjtésére. Továbbá „klasszikus”, mert a legkedveltebb támadási lehetőséget foglalja magába, eredményeket pedig gyorsan lehet kinyerni általa, ellentétben a következő fajtával.

* + 1. Vak (blind, inferential) SQLi

Az következtetéses befecskendezés esetén - ellentétben a sávon belüli injektálással - hosszabb ideig tarthat, amíg a támadók kihasználják, ugyanakkor ugyanolyan veszélyes, mint az SQL Injection bármely más formája. Ezen fajta esetén a webalkalmazáson keresztül nem kerülnek át tényleges adatok, és a támadó sem láthatja a sávon belüli támadás eredményét. Ehelyett a támadó hasznos terhelések küldésével képes rekonstruálni az adatbázis-struktúrát, figyelemmel kísérve a webalkalmazás válaszát és az adatbázis-kiszolgáló ebből fakadó viselkedését.

Mint altípusa, a logikai alapú (boolean-based) injekció egy következtetéses SQL Injection technika, amely SQL lekérdezéssel arra kényszeríti az alkalmazást, hogy a lekérdezéstől függően más eredményt adjon vissza. Az alábbi URL *https://www.example.beaglesecurity.com/gallery.php?id=1' AND 1=0 --+* [4]egy lekérdezést generál: *SELECT title, description, doby FROM items WHERE id=1’ AND 1=0*[4]. Ha az alkalmazás sebezhető SQL Injekcióval szemben, akkor nem ad vissza semmit, mivel ez egy hamis állítás, 1 nem egyenlő 0-val. Ezek után, ha egy igaz állítást adunk meg: *https://www.example.beaglesecurity.com/gallery.php?id=1' AND 1=1 --+*[4]és az oldal tartalma megváltozik az előző hamis állapothoz képest, akkor a támadó arra következtethet, hogy az injekció működik. A logikai eredménytől (IGAZ vagy HAMIS) függően tehát a HTTP válasz tartalma megváltozik, vagy ugyanaz marad. Ezt használja ki a vak injection.

Az időalapú (time-based) injekció esetén a támadó egy SQL-parancsot küld a kiszolgálónak, hogy késleltesse a lekérdezések végrehajtását. Erre például MySQL-ben a legkézenfekvőbb megoldást a *SLEEP* függvény adhatja: *SELECT \* FROM card WHERE id=1-SLEEP(15)* [5]. A másodpercben megadott szám késlelteti a lekérdezés eredményét ideális esetben. Így tehát a válaszidő jelzi a támadó számára, hogy az injektálás lehetséges és a szerver a MySQL-t használja adatbázisként. Következő lépésként például kinyerhető az adatbázis verziója: *SELECT \* FROM card WHERE id=1-IF(MID(VERSION(),1,1)=’5’, SLEEP(15),0)*[5], amely esetén ha a kiszolgáló válasza legalább 15 másodpercet vesz igénybe, megállapíthatjuk, hogy ez az adatbázis-kiszolgáló a MySQL 5.x verzióját futtatja.

* + 1. Sávon kívüli (out-of-band) SQLi

Sávon kívüli injekció akkor fordul elő, amikor a támadó nem tudja ugyan azt a csatornát használni a támadás elindításához és az eredmények összegyűjtéséhez. A legritkább használt típus a három közül, mivel a webalkalmazás által használt adatbázis-kiszolgálón engedélyezett funkcióktól függ.

Ez a technika az adatbázis-szerver azon képességére támaszkodik, hogy adatok továbbításához DNS vagy HTTP kéréseket használ. A típus sikeres használatakor a felhasználó DNS kérelmét a támadó egy olyan szerverre küldi át például a Microsoft SQL Server *xp\_dirtree* parancsával, amelyet ő irányít. Ilyen támadásra alkalmas URL például: *https://example.com/products.aspx?id=1;EXEC%20master..xp\_dirtree%20'%5c%5ctest.attacker.com%5c'+--+*[6]. Dekódoláskor a *%20* és a *%5c* a szóköznek és a fordított per jelnek felelnek meg, így az URL ténylegesen így néz ki: *https://example.com/products.aspx?id=1;EXEC master..xp\_dirtree '\\test.attacker.com\' –*[6], amely az alábbi lekérdezést eredményezi: *SELECT \* FROM product WHERE id=1;EXEC master..xp\_dirtree ’\\test.attacker.com\’ --*[6]. Ezzel a támadó a *test.master.com* webhelyre kényszeríti a DNS kérést, a támadás sikeres.

## Megelőzési módjai

Az SQL injekciós sebezhetőség elkerülése meglepően egyszerű, szembekerülve azzal a ténnyel, milyen sok hasonló fajta támadás fordul elő. A biztonsági rések megelőzésére is több lehetőség van.

**Elkészített utasítások (prepared statements) használata**

Az elkészített utasítás egy paraméterezett és újrafelhasználható SQL lekérdezés, amely arra kényszeríti a fejlesztőt, hogy külön írja be az SQL parancsot és a felhasználó által megadott adatokat. Ez a kódolási stílus lehetővé teszi az adatbázis számára, hogy különbséget tegyen a kód és az adatok között.

Gyakorlatias példával élve, ha vesszük a boolean-based altípus példáját, akkor a paraméterezett lekérdezést használva az, egy felhasználói azonosítót keresne, amely megegyezik a támadó által megadott karakterlánccal, tehát nem lenne sérülékeny.



1.2.1-es ábra: példák elkészített utasításokra [7]

**Tárolt eljárások (stored procedures) használata**

A tárolt eljárások biztonságosan megvalósítva (azaz a tárolt eljárás nem tartalmaz semmilyen nem biztonságos dinamikus SQL-t) ugyanolyan hatást gyakorolnak, mint a paraméterezett lekérdezések. Különbségük csupán csak annyi, hogy a tárolt eljárás SQL-kódját meghatározzák és magában az adatbázisban tárolják, majd az alkalmazásból meghívják (alábbiakban az *sp\_getAccountBalance*).



1.2.2-es ábra: Java példa tárolt eljárásra [8]

**Engedélyezőlista (allow-list) bemenetének ellenőrzése**

Az SQL lekérdezések különböző részei néha nem alkalmasak olyan változók használatára, amelyeket csak érvényes utasítással vagy értékcímmel lehet helyettesíteni, hogy az utasítás sikeresen végrehajtható legyen („bind variables”). Ilyen helyzetekben a bemenet ellenőrzése vagy a lekérdezés újratervezése a legmegfelelőbb védekezés. Az adatbázis táblázatainak vagy oszlopainak neve esetén például ideális esetben az értékek a kódból származnak, és nem a felhasználói paraméterekből.

Az alábbi példa egy cikluson belül különböző ágakban vizsgálja a *tableName* értékének lehetőségeit, ezzel lekorlátozva az elérhető és lekérhető táblák számát.



1.2.3-as ábra: Java példa adatbázis tábla neveinek ellenőrzésére [8]

**Az összes felhasználó által adott bemenet elkerülése**

Ennek a technikának az egyszerű célja, hogy elkerülje a felhasználói bevitelt, mielőtt lekérdezéshez lehetne használni. Alkalmazása a fenti módszerek megvalósíthatatlansága esetén kézenfekvő lehet, ám nem garancia az SQL injection megakadályozására, ún. végső lehetőséget biztosít.

1. GÉPI TANULÁS
2. PROGRAM FELÉPÍTÉSE
3. EREDMÉNY ÉS ANALÍZIS

## Irodalomjegyzék

## Nyilatkozat

Alulírott Stiller Marianna programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2021.

## Köszönetnyilvánítás