Security: SQL Injection

Academiejaar 2017-2018

Jan Van Overveldt

Johan Ven

Tom De Reys

Karel de Grote Hogeschool

In dit hoofdstuk bekijken we voornamelijk enkele belangrijke aandachtspunten waarop je moet letten als je vanuit een applicatie een databank aanspreekt. De allerbelangrijkste pitfalls overlopen we in dit hoofdstuk.

Security

SQL Injection

- Via een applicatie, SQL statements doorgeven en laten uitvoeren op de DB
  - Via input-schermen in een applicatie query doorgeven
- Mogelijke scenario's
  - Data manipulations
    - Data opvragen waar je normaal niet aan mag
      - » (OR-clausule toevoegen, ...)
    - Inserteren/update/deleten van data waar je dat eigenlijk niet mag
  - Object manipulations
    - Objecten toevoegen, verwijderen of wijzigen
      - » Drop table, create user, ...
    - Of heel de databank droppen
      - » Tot zelfs de 'master'/'sys' DB droppen, dus heel de server 'kapot' maken

2

Een veel voorkomende probleem naar security toe is SQL Injection desondanks het via enkele eenvoudige trucks toch gemakkelijk te vermijden is. Je vindt daarom dit probleem vooral terug bij oudere websites en/of bij websites die door niet-professionele programmeurs ontwikkeld zijn.

SQL Injection wil eigenlijk zeggen dat je een SQL-statement gaat injecteren in de werking van je applicatie. Dat SQL statement zal dat mee doorgestuurd en uitgevoerd worden op de DB. Hoe je dit doet is eigenlijk vrij eenvoudig en kan je bij elk invoerveld van een applicatie toepassen. In plaats van de waarde in dat veld in te vullen, typ je gewoon een SQL-statement erin met als prefix een stukje code om het vorige statement af te sluiten (bv. ";", of ");"). Daarachter zet je dan een eigen geschreven SQL-statement (bv. INSERT INTO .....) en als je geluk hebt, wordt je eigen SQL-statement effectief uitgevoerd op de databank. Om het statement tot slot af te sluiten, gebruik je het commentaarteken (--) om hetgeen achter je geïnjecteerd statement normaal komt, te negeren.
Verderop in de slides is er een demo applicatie uitgewerkt waarin je enkele voorbeelden hiervan kan bekijken.

Het is ook niet zo moeilijk om uit te voeren, je kan dat al met de kennis die je uit Programmeren 1 & 2 en Databanken 1 & 2 hebt. Aan de hand van het type scherm kan je eigenlijk al wel achterhalen welke query er uitgevoerd zal worden (select, insert, ...) en dat kan aan de slag. Je probeert de tabelnaam te gokken door een insert-statement te schrijven (bij websites kan je ook eens naar de HTML en javascript-code kijken), ... je injecteert een enkele statement achterelkaar waardoor je dynamisch de databasenaam kan achterhalen, ... Kortom: je bent vertrokken.

Voor de overzichtelijk delen we SqlInjection op in 3 verschillende soorten: Data manipulation, Object manipulation en Performance Attacks.

Onder data manipulation verstaan we SQL Injections die te maken hebben met het manipuleren van data. Dit gaat over records (of zelfs kolommen) lezen die je eigenlijk niet mag inlezen, records inserteren, wijzigen en/of verwijderen op plaatsen waar dat niet de bedoeling is, ... Je gaat dus als het ware de data integriteit van een databank omzeilen. Uiteraard wil je niet dat gebruikers ineens data gaan manipuleren terwijl dat functioneel niet mag, of data te zien krijgen die ze eigenlijk niet mogen zien. En dan heb je nog het DELETE-statement zonder WHERE-clausule of zelfs TRUNCATE van een gehele tabel

Daarnaast heb je ook Object Manipulation. Dit houdt in dat database objecten zoals tabellen, stored procedures, users, ... gaat manipuleren. Je kan definities gaan wijzigen, toevoegen of zelfs verwijderen. Enkele voorbeelden: een tabel droppen of extra aanmaken, constraints aanpassen, nieuwe user aanmaken - uiteraard met dba-rechten - waarmee je dan later gaat inloggen en alles kijken tot zelfs de database op zich droppen. Als je dan geen goed back-up plan hebt, ben je echt wel gezien...

Security

SQL Injection

- Mogelijke scenario's
  - Data manipulations
  - Object manipulations
  - Performance attacks
    - Records en/of volledige tabbelen gaan locken voor read-operaties zodanig dat niemand anders eraan kan
    - Zware berekeningen laten uitvoeren
    - ...
- Voorbeeld-applicatie
  - Zie BB: SqlInjectionDemo.zip
    - .NET applicatie op SQL Server (2012).

3

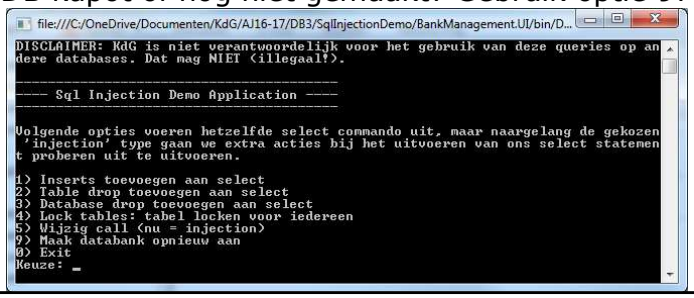
Als derde scenario heb je nog performance attacks, anders gezegd een soort van DoS attacks. Je gaat bepaalde statements laten uitvoeren die veel CPU in beslag nemen, locks leggen op veel records of zelfs gehele tabellen, heel veel connecties openen, ... kortom: je laat de database server zo hard zwoegen dat bijna niemand er nog mee kan werken.

SQL Injection kan dus vergaande gevolgen hebben. Het is daarom héél belangrijk dat je je applicatie goed beschermt daartegen. We geven in deze slides slechts enkele mogelijke scenario's maar eens je op een databank binnen geraakt, zeker met dbo-rechten, kunnen de gevolgen desastreus zijn.

Op BlackBoard is er een voorbeeld applicatie beschikbaar waarmee je enkele scenario's kan uitlokken met betrekking tot SQL Injection. Deze applicatie is een .NET applicatie, geschreven in C# met Visual Studio 2017 (maar werkt ook in VS2015) en werkend op een (lokale) SQLServer2016. Dezelfde configuratie dus als de demo rond het LoadIssue n+1. Download deze zip, pak het bestand uit en open de meegeven solution.
Op zich maakt de technologie weinig uit, want je kan dit met eender welk (relationeel) database management systeem en programmeertaal uitlokken. Dus ook met Java op Oracle bijvoorbeeld.

SQL Injection - Demo applicatie (SqlInjection.zip)

- Doel: lijst van klanten ophalen op basis van 'name'
  - Maar: als 'name' geven we voorgedefinieerde injection statements mee
    - Records inserteren in een tabel (Customers)
    - Een tabel droppen (BankAccount)
    - De databank in zijn geheel droppen
    - Eens wat records locken zodanig er geen enkele andere gebruiker aankan
  - Het uitgevoerde statement wordt telkens getoond
- Open SQL Server Management Studio en bekijk het effect...
- Is de DB kapot of nog niet gemaakt? Gebruik optie 9!



4

De applicatie doet slechts 1 actie naar de databank toe, namelijk het ophalen van een aantal klantenrecords op basis van een naam. In plaats van een echte naam mee te geven, gaan we echter telkens een stuk T-SQL script meegeven waarmee we SQL Injection aantonen.
Wees gerust, er is ook in de applicatie een optie voorzien om de databank opnieuw aan te maken als je één van de SQLInjections doorgevoerd hebt.

De query die telkens verstuurd wordt naar de database is de volgende: "SELECT Id, Name FROM dbo.Customers WHERE Name = '" + name + "'";

Als eerste gaan we eens proberen extra records te inserteren. Dit doen we door als name volgende inhoud aan te leveren:
"Jos'; INSERT INTO dbo.Customers(Name, Password) VALUES('Jos', '1234'), ('Jef', '1234'), ('Charles', '1234'); --"

Het eerste stuke "Jos' hebben we nodig om het originele statement af te sluiten. Je krijgt dan "SELECT Id, Name FROM dbo.Customers WHERE Name = 'Jos'; ...". Dat is op zich een geldige query en hij zal deze ook uitvoeren. Door bijvoorbeeld achter "Jos' het stukje "OR Name=Name" toe te voegen zou je bijvoorbeeld al alle records kunnen opvragen. Dat doen we in dit voorbeeld nu niet.
Als tweede statement gaan we dan onze SQL Injection query toevoegen, in dit geval een INSERT statement die drie records gaat inserteren.
Tot slot maken we nog een nieuw select-statement aan omdat in onze originele

query achter name nog een ' toegevoegd wordt. Dit moeten we ook nog op een propere manier gebruiken anders krijgen we een syntax error tijdens het uitvoeren. In dit voorbeeld kiezen we voor de gemakkelijkheid voor een extra SELECT statement, maar dat kan ook het toewijzen van een waarde aan een variabele zijn, ... Alles kan, zolang we die ' maar gaan gebruiken en qua syntax de juiste T-SQL code krijgen (zie menuoptie 3 voor alternatief).
Er zullen dus door ons SQL Injection statement 2 statements uitgevoerd worden naar de databank i.p.v. één. In concreto:

SELECT Id, Name FROM dbo.Customers WHERE Name = 'Jos';

INSERT INTO dbo.Customers(Name, Password) VALUES('Jos', '1234'), ('Jef', '1234'), ('Charles', '1234');

-- ' (dit was de oorspronkelijk ' die achter de Name stond, deze zetten we dus in commentaar)

Via de menuoptie 1 kunnen we de code laten uitvoeren en kan je nadien eens in de databank bekijken wat de inhoud is van de tabel 'Customers'. Merk op dat onze drie records toegevoegd zijn desondanks de applicatie eigenlijk enkel een lees-operatie gaat uitvoeren op zijn databank.

Menuoptie 2 gaat een tabel droppen in de databank. We geven de parameter 'name' de volgende waarde: "Jos' OR Name = Name; DROP TABLE dbo.BankAccount; --"
Opnieuw moeten we eerste de originele select afsluiten zodanig dat de werking van de code gegarandeerd blijft, alsook iets doen met die laatste '. Maar daartussen zetten we eender welk statement, en nu doen we eens een DROP TABLE. Bekijk goed de databank na het uitvoeren van dit statement en je zal merken dat de tabel weg is....

Bij menuoptie 3 gaan we nog een stap verder. Via een query vragen we de naam van de huidige databank op, we houden die naam bij in een variabele, sluiten alle openstaande connecties naar onze databank en vervolgens droppen we de databank. De parameter 'name' krijgt de waarde "Jos'; DECLARE @name NVARCHAR(40); SET @name = DB_Name(); EXECUTE ('alter database ' + @name + ' set single_user with rollback immediate'); USE [master]; EXECUTE ('DROP DATABASE ' + @name); --".
Als je deze optie uitvoert, mag je nog lang zoeken naar je databank... Je zal ze niet meer vinden ☺
In concreto worden dus volgende statements uitgevoerd:

```
SELECT Id, Name FROM dbo.Customers WHERE Name =
'Jos';
DECLARE @name NVARCHAR(40);
SET @name = DB_Name();
EXECUTE ('alter database ' + @name + ' set
single_user with rollback immediate');
USE [master];
```

```
EXECUTE ('DROP DATABASE ' + @name);
-- '.
```

TIP: Met de menuoptie 9 kan je je databank opnieuw aanmaken, dus geen paniek!

Tot slot hebben we de 4e optie. Hiermee gaan we eigenlijk ervoor zorgen dat we locks gaan leggen op twee tabellen zodanig dat er geen enkele andere gebruiker nog gegevens kan aanpassen of zelfs opvragen uit die tabel. Met andere woorden, alle andere users moeten wachten en/of krijgen time-outs. Daardoor kunnen er ook andere lock-waits ontstaan en deadlocks en probeer dan maar eens te achterhalen waar de oorzaak ligt. Niet eenvoudig!
De waarde die de parameter 'name' krijgt in dit voorbeeld is "Jos'; BEGIN TRANSACTION; SELECT * FROM dbo.Customers WITH(TABLOCKX); WAITFOR DELAY '00:01'; ROLLBACK; --".

In werkelijkheid wordt dus volgende uitgevoerd:

```
SELECT Id, Name FROM dbo.Customers WHERE Name = "Jos';
BEGIN TRANSACTION;
SELECT * FROM dbo.Customers WITH(TABLOCKX);
WAITFOR DELAY '00:01';
ROLLBACK;
--'
```

SQL Injection

Oplossing (1)

- Beperk rechten gebruiker
  - User waarmee applicatie naar DB connecteert is maar al te vaak "database owner / sysadmin / ..."
    - Wordt alleen gebruikt wordt door de applicatie (connectionstring aanpassen!)
    - Heeft net genoeg rechten heeft op de DB-objecten die de applicatie nodig heeft
      - » dus géén dbo / sysadmin / ...
      - » Maak gebruik van systeemrollen zoals "db_Datareader, ..."
  - Let op: dit lost niet alles op! (insert + lock blijft werken)
    - Wel: geen access maar aan DBMS instellen en commando's, bij hacken van deze user "enkel" maar één DB aangetast, ...
  - Hoe testen in Demo-applicatie?
    - Hoofdmenu - keuze 5, dan optie 2 instellen

5

Hoe kunnen we nu SQL Injection vermijden? Voor een volledig sluitende oplossing moet je twee technieken toepassen.

De eerste techniek betreft de credentials waarmee een applicatie naar zijn databank connecteert. Als je geen username en paswoord specifieert maar instelt om 'Integrated Security' (of Windows Authentication) te gebruiken, gaat hij automatisch de credentials gebruiken van het proces waarin je applicatie draait. Bij desktop applicaties is dat de ingelogde gebruiker.

Je kan op de databank ook aparte gebruikers maken en die gebruiken in je connectionstring om te vermijden dat je een hele hoop gebruikers manueel moet aanmaken in je databank. Dit wordt ook zo goed als altijd gedaan. Op zich is dat absoluut niet fout, maar het probleem zit hem vaak in welke rechten die gebruiker heeft. Dikwijls geeft men uit gemakkelijkheid veel te veel rechten, zoals database_owner (db_owner) rechten. Dit houdt in dat de user in kwestie alles mag doen op de bewuste databank, zijnde zowel DML alsook DDL en DBA queries uitvoeren. Dikwijls zijn zulke users niet alleen db_owner, maar ook sys_admin (database server administrator), want dat is toch gemakkelijk om ineens nog extra instellingen aan te passen om server niveau...
Stel dat er dan een gebruiker van de applicatie via SQL Injection zelf commando's kan uitvoeren, heb je een groot probleem want die gebruiker kan dus onbeperkt je databank of zelfs je gehele server gaan manipuleren. En dat wil je niet.

Beperk daarom de toegangsrechten van de gebruiker(s) die naar je databank

connecteren tot het absolute minimum, dus enkel lees- en schrijfrechten op de tabellen die je effectief nodig hebt, alsook enkel execution rechten op de database objecten die hij nodig heeft. Kort samengevat: geef hem enkel DML rechten en geen DDL rechten. De meeste database management systemen hebben hier trouwens ook standaard rollen voor die je kan gebruiken, dus dan hoef je niet alles object per object in te stellen.

Je zal merken dat er een aantal problemen rond SQL Injection nu opgelost zijn: er kunnen geen server instellingen meer aangepast worden, je kan geen database objecten manipuleren, ... maar de data op zich kan nog steeds gemanipuleerd worden, er kunnen nog steeds locks gelegd worden, ... Toch is het sowieso een goede zaak om met zulk een user te werken, omdat je - als men het paswoord van die gebruiker te weten komt - men ook geen server-instellingen of dergelijke kan aanpassen.

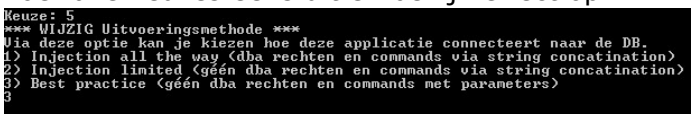
Het effect van het gebruik van zulk een user kan je uittesten met de demo applicatie. Via het menu kan je met behulp van optie 5 kiezen welke connectionstring de applicatie zal gebruiken. Als je voor optie 2 kiest (Injection Limited), zal vanaf dan de applicatie werken met een specifieke (niet-dbo) gebruiker om te connecteren naar de databank.
Voer opnieuw alle vier de tests uit en kijk na elke test wat het effect is op de databank. Je zal merken dat het inserteren (scenario 1) en locken (scenario 4) nog blijft werken, maar dat scenario 2 en 3 (droppen van de tabel respectievelijk de databank) niet meer zal lukken.

SQL Injection

Oplossing (2)

- Werk altijd met parameters
  - GEEN string concatenaties gebruiken, maar waarde van velden of filter-clausules via parameters doorgeven
  - Waarde wordt doorgegeven via de parameter ipv rechtstreeks in de query
    - Geen injection van statements meer mogelijk

- Hoe testen in demo-applicatie?
  - Maak in hoofdmenu de keuze 5, en neem daarna optie 3
  - Voer alle keuzes eens uit en bekijk effect op DB



6

Om een volledige oplossing te krijgen, mag je eigenlijk géén gebruik meer maken van string concatenaties om je queries te schrijven. Het probleem met string concatenaties is dat je wel altijd op één of andere manier een query kan doorgeven die syntaxis correct is maar door gebruik te maken van ; je meerdere statements kan laten uitvoeren.
Met andere woorden: via string concatenatie een query samenstellen waarbij één of meerdere delen door de gebruiker ingevuld kan worden, is altijd uit den boze.

Als je je daar sluitend wil tegen beveiligen, moet je gebruik maken van parameters bij het samenstellen van je queries. In plaats van de string-waarde door te geven aan je query, ga je in je query een parameter definiëren om dan vervolgens aan die parameter een waarde te koppelen. Het DBMS gaat dan de parameterwaarde uitlezen en gebruiken, maar aan de query op zich wordt niets aangepast. Je kan dus eender wat qua inhoud aan die parameter meegeven wat betreft SQL Injection, de query die je gaat uitvoeren kan je niet manipuleren en je "injection" wordt gewoon als een parameterwaarde beschouwd.

Ook dit kan je uittesten in de demo applicatie. Via menuoptie 5 kies je vervolgens de optie 3 (Best Practice) en dan zal de applicatie werken met een specifieke (niet-dbo) gebruiker om te connecteren naar de databank en via queries die gebruik maken van parameters.
Voer opnieuw alle vier de tests uit en kijk na elke test wat het effect is op de databank. Je zal merken dat nu SQL Injection in géén van de vier scenario's nog mogelijk is, ook het inserteren van extra data alsook het leggen van locks zal niet meer lukken.

SQL Injection

Conclusie

- Werk altijd met parameters
- Beperk de rechten van de user waarmee de applicatie connecteert naar de DB tot het absolute minimum
- Gebruik best altijd BEIDE oplossingen samen

Wat met ORM's?

- Werken altijd met parameters
  - (zoniet, smijt die ORM in de vuilbak...)
- Aparte user moet je wel zelf instellen
  - Zowel op databank alsook in de applicatie (connectionstring)

7

Als conclusie kunnen we dus stellen dat om SQLInjection te vermijden, je best beide oplossingen implementeert. Het werken met parameters in je queries die je opstelt vanuit je applicatie is daarbij een absolute must om SQL Injection te vermijden, string concatination zet de deur eigenlijk wagenwijd open.
Daarnaast kan je ook altijd best een aparte gebruiker aanmaken op je databank waarmee je applicatie zal connecteren. Die gebruiker geef je dan net genoeg rechten die de applicatie nodig heeft. Daarmee sluit je niet volledig SQL Injection uit, maar als er dan toch ergens een lek zit, beperk je al wat ermee kan gebeuren.

De moderne applicaties werken met ORM's die de connectie en de commando's beheren tussen de applicatie en een databank. De meeste ORM genereren zelf queries en maken daarbij gebruik van parameters. Controleer of je ORM daadwerkelijk zo werkt en indien niet, gebruik deze dan niet!!
Bij een ORM moet je zelf altijd een connectionstring definiëren en dus bijgevolg ook welke user er vanuit je applicatie naar de databank connecteert. Deze oplossing moet je dus ook bij gebruik van een ORM zelf implementeren.