Strategier för att försvara mot SQL-injektioner

Kristian Nykänen, 41881

Kandidatavhandling i datateknik

Åbo Akademi

Handledare: Dragos Truscan

Abstrakt

När tekniken förändras, blir det allt svårare för företag att upprätthålla säkerheten för kundernas personliga information. Eftersom webbapplikationer blir allt mer populära, blir också säkerhetsproblemen mer aktuella. Alla som arbetar inom webbutveckling bör veta om riskerna som följer med i deras applikationer.

Denna avhandling kommer att inrikta sig på SQL-injektioner, som anses vara ett av de vanligaste säkerhetsproblemen inom webbutveckling. Avhandlingen förklarar hur SQL-injektioner utförs ur angriparens synvinkel, samt visa olika försvarsstrategier som en programmerare måste ta i beaktande under utvecklingen av en webbapplikation.

Innehållsförteckning

1. Inledning

I början av internets utveckling användes webbsidor och webbservrar för att dela ut statiska filer som endast innehöll text. Efter att internet utvecklades vidare, blev det vanligare att webbservern delade ut dynamiska filer istället för statiska filer. Skillnaden mellan dessa statiska och dynamiska filer är att innehållet i statiska filer hålls samma och webbservern delar alltid ut samma fil, medan innehållet i dynamiska filer förändras. Exempel på en dynamisk fil kan vara en webbsida som har en klocka i sig som förändras enligt den aktuella tiden. En webbapplikation bygger på denna utveckling med att det som körs på en webbläsare till största delen är skrivet i ett programmeringsspråk som till exempel JavaScript, vilket gör att kommunikationsmetoden mellan klienten och servern är väldigt olik jämfört med äldre metoder.

Idag är det väldigt svårt att hitta webbsidor som består endast av statiska filer för att representera webbsidan. Exempel på olika webbapplikationer som finns idag kan vara nätbanker och sociala medier, vilka för det mesta innehåller i alla fall en inloggningssida. Inloggningssidan innehåller fält för både ett användarnamn och ett lösenord för kontot. Webbserver tar input från dessa fält och sparar informationen i en relationsdatabas, som i sin tur innehåller personlig information för kontot och endast ska vara tillgänglig för databasunderhållare.

SQL-injektioner är attacker som försöker komma åt informationen i dessa relationsdatabaser med hjälp av att injektera kod som indata till webbservern, vilket skickas vidare till databasen. Den som utvecklar webbapplikationen måste se alla inputfält som farliga och tänka på alla risker som de kan föra med sig med omsorg [1]. Eftersom attackerna involverar obehörig tillgång till personlig information är dessa attacker olagliga. Idag är största delen av webbapplikationerna som existerar osäkra även med användningen av SSL (*Secure Sockets Layer*) teknologin [1]. I en undersökning från år 2007 till 2011 kom man fram till att 32% av undersökta webbapplikationer innehåller risken för

SQL-injektioner [1]. Detta beror på att företag och webbutvecklare inte vet om eller bortser säkra programmeringsmetoder som strider mot dessa risker.

Väsentliga delen av problem i webbapplikationens säkerhet uppstår från situationer där applikationen måste motta data och bearbeta den på ett eller annat sätt. Data som mottas måste anses som opålitligt och kan i värsta fall vara farligt. De mest allvarliga attackerna är de som kan få tillgång till känslig information eller obegränsad tillgång till administrationsdelen av en webbapplikation [1]. På grund av att webbapplikationer blir allt mer populära, måste säkerheten också utvecklas. Tyvärr är förståelsen av säkerhetshot och antalet effektiva sätt att motverka dessa hot underutvecklad inom industrin [1].

Open Web Application Security Project (OWASP) har lagrat en lista på de tio de mest kritiska säkerhetsriskerna i webbapplikationer. OWASP är en icke-vinstdrivande organisation med målet att förbättra säkerheten av mjukvara. Listan på säkerhetsriskerna används för att dra uppmärksamhet till de största riskerna i industrin, så att utvecklare och företag kan minimera möjligheten för de nämnda riskerna [15]. Enligt OWASP är tre av de mest hotande riskerna för tillfället:

- **Injektioner -** Denna sårbarhet tillåter angriparen att skicka opålitliga data till en programtolk i administrationsdelen av en webbapplikation. Angriparens fientliga indata kan få programtolken att utföra olika kommandon på en databas eller ge åtkomst till data utan tillstånd. Det finns flera typer av injektioner, t. ex SQL- och NoSQL-injektioner [15].

- **Bruten autentisering -** Denna kategori omfattar olika brister i applikationens inloggningsfunktion. I många fall är autentiseringshanteringen dåligt byggt, vilket kan tillåta angriparen att gissa åt sig svaga lösenord, komma förbi inloggningen eller få tag på en annan användares identitet för att komma in i applikationen [15].

- **Känslig dataexposition -** Om en webbapplikation eller API (*Application programming interface*) skyddar känsliga data dåligt, klassificeras risken som

känslig dataexposition. Angripare kan stjäla eller modifiera denna data för att utföra brott av olika slag. I största delen av fallen beror känslig dataexposition på att man helt enkelt använder inte kryptering för att skydda känsliga data [15].

Avhandlingen kommer ge en överblick av arkitekturen i en webbapplikation och vilka tekniker som används. Utveckling av webbapplikationer för med sig olika risker som till exempel risken för SQL-injektioner. Denna avhandling kommer förklara vad en SQL-injektion är, hur man utför en injektion, samt gå igenom olika strategier för att försvara mot och minska antalet injektionsattacker.

2. Tekniker i webbapplikationer

En webbapplikation är en applikation som genererar anpassat innehåll för olika användare. Webbapplikationen tar hand om logiken i applikationen för att generera dynamiska filer enligt till exempel inmatningsdata. En webbklient kan kommunicera med webbservern över ett bestämt protokoll som till exempel HTTP-protokoll *(Hypertext Transfer Protocol)*, vilket i sin tur är ansluten till applikationsservern, som tar hand om logikskitet. HTTP-protokollet innehåller bestämda regler så att webbklienten och webbservern skall kunna kommunicera med varandra. Den aktiva kommunikationen mellan webbservern och webbklienten är det som skiljer en webbapplikation från en vanlig webbsida. I denna avsnitt förklaras de tekniker som behövs för en webbapplikation med kommunikation till en databas.

2.1 HTTP

Hypertext Transfer Protocol (HTTP)[2] är webbens viktigaste del för att överföra filer. HTTP-protokollet är ett klient/server-protokoll, vilket betyder att det alltid är webbklienten (oftast webbläsaren) som gör förfrågan till en webbserver. När en

webbklient vill hämta en webbsida, gör webbklienten en GET-förfrågan till webbservern för att få en fil i HTML-format (*Hypertext Markup Language*) som svar till förfrågningen, vilket i sin tur ska föreställa webbsidan.

```
GET / HTTP/1.1
Host: www.test.com
```

Figur 2.1 HTTP-förfrågan

```
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Mon, 23 Mar 2020 16:30:16 GMT
Content-Type: text/html
...

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
...
```

Figur 2.2 HTTP-svar

I figurerna 2.1 och 2.2 illustreras hur en vanlig HTTP-förfrågning fungerar om man öppnar en webbsida på adressen *http://www.test.com/,* som gör att webbklienten skickar en GET-förfrågan (figur 2.1) till webbservern och väntar på ett svar (figur 2.2). I förfrågningens första rad definieras vilken typ av förfrågning man gör med ett specifikt verb på engelska. De vanligaste förfrågningarna är GET, som hämtar en specifik resurs (till exempel en HTML-fil) från webbservern, och POST, vilket i sin tur används för att skapa eller updatera nya resurser (t. ex sätta till ett objekt in en databas) [10]. Efter förfrågningsverbet specificeras adressen eller namnet på resursen som man vill hämta. I figur 2.1 används ’/’ (snedstreck) för adressen, som används i Unix system för att definiera rotkatalogen, vilket i detta fall används för att hämta HTML-filen för startsidan

(roten definieras på rad två med ”Host: *www.test.com”)*. I slutet av första raden definierar man också kort vilken HTTP-protokollversion som används.

I HTTP-svaret får man veta om förfrågningen lyckades eller inte. Det finns HTTP-responskoder för alla möjliga fall, av vilka de vanligaste är statuskoden 200 och 404. Statuskoden 200 returneras ifall förfrågningen lyckades och ifall sidan inte hittas, returneras koden 404. I figur 2.2 ser man på första raden att svaret returnerade HTTP-statuskoden 200, som ger ett lyckat svar. I detta fall får man i slutet av HTTP-svaret HTML-data som förfrågningen frågade efter [10].

Efter att webbklienten har mottagit HTML-filen (ifall den existerar), analyserar webbklienten filen för att se till att den har tillgång till alla nödvändiga filer för att representera sidan. Oftast brukar en webbsida behöva exekveringsskript, layoutinformation i form av en CSS-fil (*Cascading Style Sheets*), samt andra filer som t.ex. bilder och videoklipp. Ifall filer fattas eller om webbklienten utför ett exekveringsskript, kan webbklienten fortsättningsvis göra nya förfrågningar till webbservern för att webbsidan ska representeras som avsett [2].

2.2 HTML

Hypertext Markup Language (HTML)[3] är ett sidbeskrivningsspråk som används för att konstruera de webbsidor som bildar World Wide Web. Med hjälp av HTML-dokument kan man göra webbapplikationer som användaren kan interagera med på ett eller annat sätt. En HTML-dokument är en grundläggande textfil som innehåller olika HTML-taggar, som tillsammans bygger upp en layout för en webbsida [3].

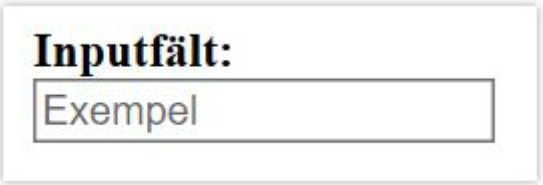
```
<html>
<header><title>Webbsidans namn</title></header>
<body>
HTML exempel
</body>
</html>
```

Figur 2.3 Enkel HTML-dokument

I figur 2.3 illustreras en väldigt enkel HTML-dokument som skriver ut ”HTML exempel” på webbsidan med hjälp av HTML-taggar. För att få input från användaren kan man använda en <*form*> tagg inne i <*body*> delen, som bygger en layout för ett inputfält. In i form-taggen kan man använda <*input*> taggar med olika egenskaper, som kan bygga upp till exempel ett textfält där användaren själv kan skriva något valfritt. För att få ett inputfält i användargränssittet, kan man använda <*form*> och <*input*> taggarna enligt figur 2.4, vilket föreställs i användargränssnittet enligt figur 2.5.

```
<html>
<head>
    <meta charset="utf-8">
    <title>Webbsidans namn</title>
</head>
<body>
    <form>
        <label><b>Inputfält:</b></label><br/>
        <input type="text" placeholder="Exempel"><br/>
    </form>
</body>
</html>
```

Figur 2.4 HTML-dokument för ett inputfält



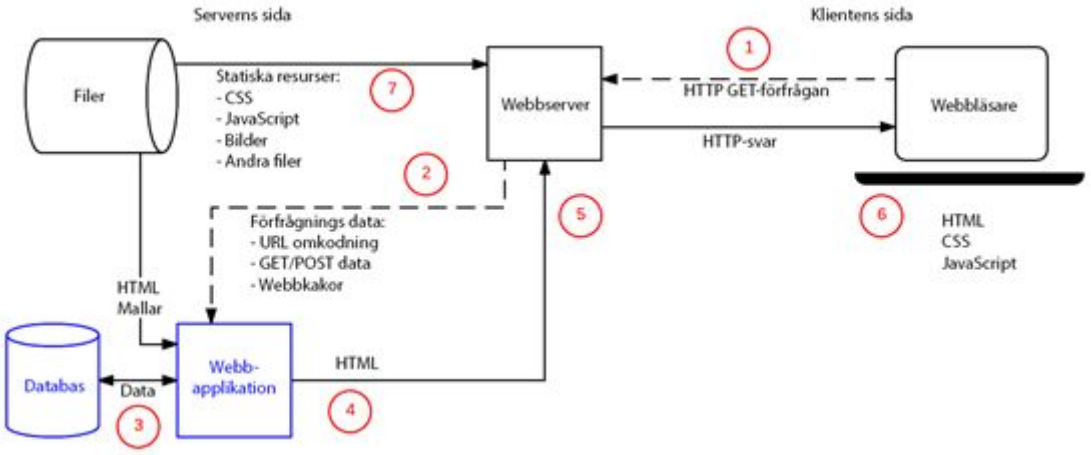
Figur 2.5 Enkel inputfält

Indatan man skriver in i ett textfält används i många fall för att köra förfrågningar till en databas, vilket gör att webbapplikationsutvecklaren måste se till att användaren inte kan mata in ogiltiga inmatningar med tanken att göra ett anfall för att få information från databasen. All indata som man får från användaren ska anses som farliga och man ska se till att eliminera bort alla möjliga risker med indatat innan man använder indatat för att till exempel söka i en databas.

2.3 JavaScript

JavaScript är ett skriptspråk eller ett programmeringsspråk som tillåter utvecklaren att göra dynamiskt uppdaterande innehåll för webbapplikationen. Om man laddar en webbsida med komplexa funktioner, d.v.s. icke-statiska webbsidor, är det JavaScript som sköter om dessa funktioner i många fall. Exempel på funktioner som JavaScript möjliggör kan vara animerade 2D/3D grafiker, animerade bilder, uppdatering av data och ändring av innehåll på webbsidan [4]. JavaScript ger webbutvecklaren verktyg för att göra webbapplikationen mer interaktiv och nya designmöjligheter för att ge användaren en bättre upplevelse av webbapplikationen.

2.4 Klient-server arkitektur av en webbapplikation



Figur 2.6 Arkitekturen för en dynamisk HTTP-förfrågan [10].

På grund av att webbapplikationen är den som bearbetar indata som användaren skriver in i till exempel ett inputfält, måste man skydda alla operationer som använder användarens indata, för att minska risker för olika slag av anfall. I figur 2.6 illustreras arkitekturen för en dynamisk HTTP-förfrågan och hur data förflyttar sig mellan de olika skikten. En dynamisk webbsida är genererad och returnerad baserat på specifika data som skickas till webbservern, d.v.s. webbsidan använder inte en statisk fil för att visa upp webbsidan. Istället för statiska filer använder man mallar som kan lätt förändra data enligt vilken användare använder applikationen [10], d.v.s. en dynamisk fil. Ifall användaren har syftet att anfalla en sårbar del av applikationen som till exempel databasen, kan angriparen skriva in olika kommandon in i textfälten som kommunicerar direkt med en databas, vilket i sin tur betyder att angriparen har tillgång till så gott som alla funktioner och information som databasen innehåller.

För att webbapplikationen ska kunna kommunicera med en databas, kan man använda olika typer av mjukvara som webbservrar, vilket tar hand om logikskiktet i en webbapplikation. Exempel på kända webbservrar som används i industrin är Nginx och Apache [6]. Webbservern tar emot data från webbklienten, vilket också anses som presentationsskiktet, med ett HTTP-förfrågan och exekverar ett skript

som behandlar förfrågningen, vilket i sin tur kommer att påbörja kommunikationen med databashanteraren som hör till lagringsskiktet. Presentationsskiktet, logikskiktet och lagringsskiktet bildar treskiktsarkitekturen som används i största delen av databasdrivna webbapplikationer.

2.5 Databaser

En stor del av webbapplikationerna använder sig stora mängder data för att applikationens funktioner ska fungera som de ska. Informationen som webbapplikationen lagrar måste sparas på ett eller annat sätt, som förverkligas med databaser. Databasen ger möjligheten att spara information i organiserade samlingar i ett datorsystem. Största delen av de mest populära databaser använder relationsdatabasmodellen, som anses vara mest effektiv och flexibel att använda [11]. Relationsdatabasen har tabeller som innehåller rader och kolumner av data, som kan till exempel hämtas, modifieras, uppdateras eller tas bort med hjälp av olika databashanteringssystem [13]. Populära databashanteringssystem som används idag är till exempel Oracle Database, MySQL och Microsoft SQL server [12]. Nästan alla av de mest populära relationsdatabaser använder SQL som programmeringsspråk, vilket står för *Structured Query Language* [13], men det finns också andra programmeringsspråk som håller på att dyka upp [12]. SQL programmeringsspråket ger möjligheten att kommunicera med databashanteringssystemet, vilket i sin tur genomför operationer enligt vad som förfrågas.

2.5.1 SQL syntax

För att klargöra hur en SQL-injektion utförs, är det viktigt att veta hur syntaxen i SQL programmeringsspråket används. SQL är överraskande lätt att förstå även för människor som har ingen programmeringserfarenhet från förut. Orsaken som gör språket lätt att förstå är på grund av att syntaxen är väldigt lik vardaglig engelska.

Man använder olika engelska ord i en specifik ordning för att definiera vilken förfrågning man vill utföra, som bildar en mening vilket liknar vardaglig engelska ifall man läser ut operationerna högt. Exempel på de vanligaste operationer som används i SQL syntaxen är:

- **SELECT FROM** – Läser in kolumner som specificeras efter SELECT-operationen från tabellen som specificeras efter FROM-operationen. Ifall man vill läsa in alla kolumner som hittas i tabellen, kan man använda symbolen ’*’ (asterisk) istället för att specificera namnet på kolumnen [13][14].
- **CREATE** – Används för att skapa till exempel en tabell med kommandon CREATE TABLE. Databasen behöver tabeller för att man ska kunna spara data [13][14].
- **DELETE** – Används ifall man vill ta bort en eller flera rader i en tabell, men tabell strukturen behålls. Vid användning av DELETE-operationen kan man återladda data ifall man vill få tillbaka borttagna data [13][14].
- **DROP** – Till skillnad från DELETE-operationen, används DROP TABLE för att ta bort en tabell från databasen, samt informationen som den innehöll [13][14].
- **UPDATE** – Operationen tillåter uppdatering av en eller flera kolumner av existerande data [13][14].
- **INSERT INTO** – Tillåter insättning av data in i en tabell [13][14].
- **WHERE** – Används i samband med operationer som till exempel SELECT, UPDATE och DELETE, för att specificera vilka rader som ska väljas. [13][14].

Med hjälp av att binda ihop dessa operationer kan man konstruera olika förfrågningar till databashanteringssystemet. Ett exempel på en SQL-förfrågning som returnerar allt information om alla kunder som finns i en databas illustreras i figur 2.5 och 2.6, vilket visar syntaxen och vad som returneras ifall man utför förfrågningen.

```
SELECT * FROM kunder;
```

Figur 2.5 SQL-förfrågan

id	namn	ålder
1	John Doe	32
2	Jane Smith	45
3	Dan Williams	22
4	James Davis	25
5	Peter Tyson	28

Figur 2.6 Tabell av alla kunder

I figur 2.5 används SELECT-operationen med en asterisk som parameter, vilket betyder att förfrågningen vill hämta all information som finns från tabellen *kunder*. I SQL syntaxen används ';' (semikolon) i slutet av förfrågningen som satsavskiljare. Om man vill filtrera resultatet att endast innehålla kunder som är äldre än 30 år, skulle förfrågningen se ut enligt figur 2.7:

```
SELECT * FROM kunder
WHERE ålder > 30; -- Kommentarer påverkar inte förfrågningen
```

Figur 2.7 SQL-förfrågan med filtrering och kommentar

Illustrationen i figur 2.7 använder WHERE-operationen för att filtrera bort rader som förfrågningen är inte intresserad i. I detta fall, när man endast är intresserad av kunder äldre än 30 år, kommer raderna med namnet ”John Doe” och ”Jane Smith” att returneras som svar till SQL-förfrågningen medan resten ignoreras. Illustrationen visar också hur kommentarer används i SQL syntaxen. Syntaxen använder sig av två bindestreck ('--') för att påbörja en kommentar. Allt text som skrivs mellan två bindestreck och slutet av raden ignoreras i förfrågningen och exekveras inte. Kommentarer kan spela en stor roll i risken för många fall av SQL-injektioner. Angriparen kan manipulera och omformulera förfrågningen till

databasen med hjälp av kommentarer för att göra en giltig förfrågning, vilket i sin tur tillåter angriparen att utföra vilket SQL-kommando som helst. Om databasen skyddas inte mot anfall av dessa slag, ligger databasens information i fara.

3. SQL-injektion

Detta kapitel kommer ta en närmare titt på vad en SQL-injektion är och hur den utförs. Kapitlet kommer också ge en helhetssyn på webbapplikationssäkerhet och dens viktighet. Det är viktigt för en webbapplikationsutvecklare att förstå vad som möjliggör SQL-injektionsattacker och hur man ska hantera känslig information.

SQL-injektion är en typ av attack mot en webbserver, vilket i sin tur kommunicerar med en databas som innehåller data. Attacken klassificeras som en typ av sårbarhet i kategorin ”injektioner” enligt OWASP, och är en av de vanligaste attackerna som finns inom webbapplikationssäkerhet [15].

För att förhindra SQL-injektioner, måste man se till att behandla indata och utdata med omsorg. Det som möjliggör SQL-injektioner är att World Wide Web utvecklades inte med säkerhet i åtanke, vilket man måste kompensera med applikationer som använder bra kodningspraxis och kontrollerade inmatningsfält i användargränssnittet [16]. Om applikationen drabbas av en SQL-injektion, har utvecklaren misslyckats med att sanera applikationens indata och utdata. Ur angriparens synvinkel, är det lätt att hitta en webbapplikation som innehåller dålig kod och är känslig för SQL-injektioner. I värsta fall kan angriparen få reda på om webbapplikationen är känslig för en attack med att skriva in endast ett specifikt tecken in i inmatningsfältet.

3.1 Hur SQL-injektioner utförs

I detta avsnitt demonstreras ett fåtal tillvägagångssätt för att utföra en SQL-injektion. SQL-injektioner är olagliga att utföra utan tillstånd och rekommenderas inte om man inte vet vad man håller på med.

Figur 3.1 Exempel på inloggningsfält

I figur 3.1 ser man ett väldigt enkelt inloggningsfält. I vanliga fall när användaren vill logga in på en webbapplikation, skulle användaren skriva in till exempel ”john.doe” i fältet för användarnamn och ”lösenord” i fältet för lös enord. Enligt figur 2.4 i avsnitt 2.4, kommer indatat att skickas till webbservern, där indata bearbetas, vilket i sin tur kommer att göra en förfrågan till databasen som webbservern kommunicerar med.

```
app.post('/submit', function (req, res) {
    var sql_förfrågan = "SELECT * FROM Användare \
    WHERE användarnamn = '" + req.body.användarnamn + "' AND lösenord = '" + req.body.lösenord + "'";

    mysqlConnection.query(sql_förfrågan, function (err) {
        //...
```

Figur 3.2 Logiken för en förfrågan (Node.js)

I figur 3.2 demonstreras logikskiktet för en förfrågning, som i detta fall körs på en JavaScript exekveringsmiljö som heter Node.js. Man använder ett fördefinierat format på förfrågningen, som bestäms av utvecklaren. När användaren trycker på ”Logga in” knappen i figur 3.1, skickas ett POST-kommando från presentationskiktet till logikskiktet, vilket mottas i figur 3.2. I detta fall kan man få indata från POST-kommandon genom att använda ”*req.body.användarnamn/-lösenord”*, som färdigställer formatet för förfrågningen. Efter att strängen för förfrågningen är klar, skickar man

förfrågningen till MySQL-databasen med funktionen ”*query*”. Implementationen i figur 3.2 är dock sårbar och väldigt lätt att attackera.

Om angriparen skulle mata in ett citationstecken, kan angriparen få reda på om inmatningsfälten är sårbara för SQL-injektion. Ifall man matar in ett citationstecken i inmatningsfälten och webbservern returnerar ett felmeddelande som syftar på att syntaxen är fel, får angriparen en aning om hur syntaxen för förfrågningen kan se ut och att det är möjligt att manipulera inmatningen för att utföra andra SQL-kommandon till databasen [17].

Orsaken till hur angriparen kan få reda på om applikationen är sårbar bara med att mata in ett citationstecken beror på att strängen som innehåller formatet för förfrågningen blir ogiltig, på grund av att det finns ett citationstecken för mycket [17]. En webbserver som har försäkrat sig mot SQL-injektioner skulle returnera ett meddelande som syftar på att det finns inga användare i databasen som har användarnamnet ” (citationstecken) eller ogiltig kombination av användarnamn och lösenord.

När angriparen vet att det finns en sårbarhet i inmatningsfälten, kan angriparen börja injektera kod till databasen. Istället för att skriva ett vanligt användarnamn in i inmatningsfältet, kan angriparen skriva in strängen **’ OR ’1’=’1** och lämna lösenordet tomt. När strängen sätts in i formatet för förfrågningen i logikskiktet, kommer förfrågningen se ut enligt figur 3.3:

```
SELECT * FROM Användare WHERE användarnamn = '' OR '1'='1' AND lösenord = ''
```

Figur 3.3 Injekterad SQL-förfrågning

```
SELECT * FROM Användare WHERE användarnamn = ''
```

Figur 3.4 Ekvivalent och förenklad förfrågning av figur 3.3

Förfrågningen i figur 3.3 kommer dock inte ge alla rader i databasen. På grund av att AND-operationen har högre prioritet än OR-operationen, kommer förfrågningen endast returnera en användare med ett blankt användarnamn ifall den finns [17]. Man kan se förfrågningen i figur 3.3 och figur 3.4 som ekvivalenta på grund av att lösenordet har ingen betydelse i förfrågningen. Ekvivalensen beror på att 1=1 är alltid är sann och vi antar att det inte finns ett blankt lösenord i databasen, som kommer att returnera falskt. Om man utför en AND-operation på ett sant och ett falskt påstående, kommer svaret bli falskt enligt logikens regler. På grund av att **'1'='1' AND lösenord = ''** returnerar ett falskt påstående, måste man på ett eller annat sätt få användarnamnet att bli sann för att förfrågningen skall returnera något. Detta beror på att OR-operationen returnerar alltid sant om man har minst ett påstående som är sann. Med denna SQL-injektion kan angriparen i värsta fall logga in i applikationen med vilket konto som helst bara angriparen känner till användarnamnet.

Ifall angriparen vill få tag på alla användare i databasen, kan angriparen modifiera på SQL-injektionen i figur 3.3. Om angriparen skriver in **' OR 1=1;--** i fältet för användarnamn, kommer SQL-förfrågan se ut enligt figur 3.4:

```
SELECT * FROM Användare WHERE användarnamn = '' OR 1=1;-- ' AND lösenord = ''
```

Figur 3.4 Injekterad SQL-förfrågning

```
SELECT * FROM Användare
```

Figur 3.5 Ekvivalent och förenklad förfrågning av figur 3.4

Med hjälp av kommentarer, kan man avsluta en förfrågning tidigare än förväntat, vilket illustreras i figur 3.4. Kommentaren ignorerar AND operationen helt och kör endast OR-operationen. På grund av att 1=1 är alltid sann, kommer alla rader i tabellen returnera sant, vilket i sin tur kommer returneras till angriparen.

Angriparen kan göra ännu mer skada till databasen med att köra egna förfrågning till databasen. Med hjälp av att skära av förfrågningen tidigare, kan angriparen också köra en till förfrågning efter semikolonet i figur 3.4. Om angriparen till exempel kör **DROP TABLE Användare;** efter semikolonet, kommer tabellen *Användare* tas bort helt ur databasen efter att alla användare har returnerats till användaren. Med dessa exempel kan man se hur angriparen kan manipulera förfrågningen enligt tycke, som visar hur mycket skada en SQL-injektion kan göra i praktiken.

4. Försvarsstrategier mot SQL-injektioner

I detta kapitel kommer ett antal strategier för att försvara och minska på SQL-injektioner. Strategierna som tas upp anses vara de primära strategier som varje utvecklare borde känna igen enligt OWASP [18], om applikationen som utvecklas kommunicerar med en databas och använder indata från okända användare.

Många webbsidor som ger ut handledning och exempelkod på vanliga programmeringsrelaterade problem, lär oftast ut osäkra kodningspraxis, vilket i sin tur leder till osäkra applikationer [17]. Situationen har dock förbättrats med tiden, på grund av att man kommer väldigt lätt åt information om hur man försvarar applikationer mot SQL-injektioner genom internet [17].

Enligt OWASP är förhindring av SQL-injektioner väldigt enkelt. Utvecklare måste upphöra användningen av dynamiska förfrågningar eller förhindra användaren från att manipulera logiken i förfrågningen med att mata in osäkra data [18].

TODO

Referenser

[1] D. Stuttard och M. Pinto, The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws, Andra upplagan, *s. 7-8, 9, 14*

[2] MDN Web Docs, “An overview of HTTP" [Online] [Hämtad 13.02.2020]

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[3] MDN Web Docs, “HTML: Hypertext Markup Language" [Online] [Hämtad 13.02.2020]

https://developer.mozilla.org/en-US/docs/Web/HTML

[4] MDN Web Docs, “JavaScript" [Online] [Hämtad 14.02.2020]

https://developer.mozilla.org/en-US/docs/Web/JavaScript

[5] MDN Web Docs, “Server-Side web Frameworks" [Online] [Hämtad 17.03.2020]

https://developer.mozilla.org/en-US/docs/Web/JavaScript

[6] W3Techs, “Usage statistics of web servers" [Online] [Hämtad 17.03.2020]

https://w3techs.com/technologies/overview/web_server

[10] MDN Web Docs, “Client-Server Overview" [Online] [Hämtad 22.03.2020]

https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

[11] DB-Engines, “DB-Engines Ranking” [Online] [Hämtad 24.03.2020]

https://db-engines.com/en/ranking

[12] Oracle Corporation, “What Is a Database?” [Online] [Hämtad 24.03.2020]

https://www.oracle.com/database/what-is-database.html

[13] A. Silbershatz, H. Korth och S. Sudarshan, “Database System Concepts”, 6:e upplagan, *s. 9, 12-15, 63-66, 98-103*

[14] D. Darmawikarta, “SQL for MySQL: A Beginner’s Tutorial”, Första upplagan, *s. 6, 7, 8, 10, 13, 81, 85*

[15] Open Web Application Security Project (OWASP), ”OWASP Top 10 – 2017” [Online] [Hämtad 28.03.2020]

https://owasp.org/www-project-top-ten/

[16] J. Fonseca, M. Vieira, H. Madeira, “Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks”, [Online] [Hämtad 28.03.2020]

https://ieeexplore.ieee.org/document/4459684

[17] J. Clarke, “SQL Injection Attacks and Defense”, Andra upplagan, *s. 15*