Engineer

SQL-injektion

en attack som gör indata till en del av en databasfråga

Vad det är

SQL-injektion, på engelska SQL injection och ofta förkortat SQLi, är angreppet där en applikation bygger ett SQL-kommando av text som någon annan har lämnat, och databasen till slut läser en del av texten som SQL i stället för som ett värde. Common Weakness Enumeration för upp svagheten som CWE-89 och beskriver den som ett kommando som helt eller delvis byggs av indata utifrån, utan att de tecken som har särskild betydelse i SQL först har oskadliggjorts. OWASP:s fusklapp för förebyggande (cheat sheet) säger samma sak enklare: angreppet fungerar när data som inte går att lita på blir en del av syntaxen i en databasfråga.

Roten till problemet är att SQL inte drar någon fast gräns mellan instruktionerna i ett kommando och datan inuti det. Ett enkelt citattecken avslutar en strängliteral, och allt som angriparen skriver efter det tolkas som mer SQL. Två bindestreck inleder en kommentar, vilket låter angriparen kasta bort resten av utvecklarens sats. Ett semikolon avslutar, där databasen och drivrutinen tillåter det, en sats och inleder nästa. Inget av tecknen är exotiskt: samma tecken dyker upp i efternamn och fritext.

OWASP:s Web Security Testing Guide delar in angreppen i tre klasser efter hur resultaten når angriparen. Angrepp i samma kanal (in-band) läser svaret genom samma kanal som användes för att skicka det, oftast på applikationens egen sida. Angrepp utanför kanalen (out-of-band) får databasen att leverera resultaten någon annanstans, till exempel över en separat HTTP-anslutning. Slutledande eller blinda angrepp (inferential eller blind) får inga data alls, utan bygger upp dem en fråga i taget utifrån hur applikationen beter sig. Guiden nämner fem vanliga tekniker i de här klasserna: en `UNION` -operator som hänger på en andra `SELECT` efter den första, booleska villkor, fel som framkallas med flit, anrop utanför kanalen och avsiktliga fördröjningar.

SQL-injektion hör till en större familj. I CWE står den under den allmänna injektionssvagheten, CWE-74, tillsammans med cross-site scripting, och OWASP Top Ten för 2025 placerar båda i kategorin A05 Injection. Den gemensamma nämnaren är data som korsar en gräns och hamnar på ett ställe där den läses som instruktioner. Här är tolken databasens SQL-parser, och botemedlet är att hålla varje värde helt utanför SQL-texten.

Använd begreppet i ett beslut

Använd ”Vad är SQL-injektion?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Skadan beror på vad applikationens databaskonto får göra, och det är ofta mycket. CWE-89 räknar upp att läsa data som angriparen aldrig skulle ha sett, att ta sig förbi autentiseringen utan att känna till ett lösenord, att ändra de data som avgör vem som får göra vad, att ändra eller radera poster och, i vissa uppsättningar, att köra kommandon i operativsystemet. OWASP:s sida om angreppet lägger till administrativa åtgärder som att stänga av databasen, läsa innehåll från databasserverns disk och ta över databasservern som administratör.

Båda källorna bedömer svagheten som allvarlig. CWE anger sannolikheten för utnyttjande som hög, och OWASP:s sida om angreppet beskriver felet som både lätt att hitta och lätt att utnyttja. Svagheten finns med i varje utgåva av CWE Top 25 Most Dangerous Software Weaknesses från 2019 till 2025. CWE-posten nämner också en våg 2008, då en och samma angreppssträng, som fungerade mot många olika program, komprometterade ett stort antal webbservrar och fick dem att sprida skadlig kod.

Försvaret är däremot välkänt och billigt. OWASP:s fusklapp gör förberedda satser (prepared statements) med bundna parametrar till det första försvaret, och CWE rekommenderar dem också: SQL-koden och värdena färdas var för sig, så databasen kan inte förväxla det ena med det andra. Allt annat, från minsta möjliga behörighet till noggrann felhantering, är ett extra lager som begränsar skadan när ett misstag slinker igenom.

Så ser det ut i praktiken

Ta en webbutik vars sida med orderhistorik slår upp beställningar efter den e-postadress som skrivs in i ett formulär. Utvecklaren bygger SQL-koden med en f-sträng i Python:

```
cur.execute(f"SELECT id, total FROM orders WHERE email = '{email}'")
```

En besökare skriver `x' OR '1'='1` i fältet. Programmet klistrar in det som det är, och databasen tar emot det här:

```
SELECT id, total FROM orders WHERE email = 'x' OR '1'='1'
```

Citattecknet efter `x` stänger strängen som utvecklaren öppnade, och `OR`-villkoret efter det är alltid sant, så databasen returnerar varenda beställning i tabellen. CWE-89 och testguiden använder samma alltid sanna trick, som också kan få ett inloggningsformulär att släppa in en besökare som varken känner till användarnamnet eller lösenordet.

Värre indata avslutar satsen och inleder en ny. Exemplet i CWE-posten matar in `name'; DELETE FROM items; --` som artikelnamn, så att en andra sats följer på den första och bindestrecken gör resten till en kommentar. Om det körs beror på databasen och drivrutinen, och flera av dagens gränssnitt vägrar. PostgreSQL:s `PQexecParams` tar högst ett SQL-kommando per sträng, vilket dokumentationen räknar som ett visst skydd mot SQL-injektion. Python-modulen `sqlite3` ger ett fel så snart `execute()` får mer än en sats, och psycopg vägrar köra flera satser i ett anrop när parametrar skickas med. Inget av det stoppar det alltid sanna villkoret ovan, som bara behöver en sats.

Lösningen är att skicka värdet separat. PEP 249, standarden för databasgränssnitt i Python, låter varje drivrutin ange sin platshållarstil i en modulkonstant som heter `paramstyle`, och räknar upp fem stilar: `qmark` (`?`), `numeric` (`:1`), `named` (`:name`), `format` (`%s`) och `pyformat` (`%(name)s`). Modulen `sqlite3` är låst till `qmark` och godtar även `named`. Psycopg, drivrutinen för PostgreSQL, tar `%s` och `%(name)s`:

```
cur.execute(
    "SELECT id, total FROM orders WHERE email = %s",
    (email,),
)
```

`%s` ser ut som strängformatering i Python, men ingenting formateras: SQL-koden och tupeln med värden når drivrutinen som två separata argument. Psycopg 3 håller isär dem hela vägen och skickar kommandot och dess parametrar till servern var för sig, i stället för att foga ihop dem hos klienten. OWASP:s fusklapp visar poängen med sitt eget exempel på indata, `tom' or '1'='1` : bundet som parameter är det ett märkligt användarnamn som slås upp bokstavligt, och det matchar ingen.

PostgreSQL erbjuder samma uppdelning i ren SQL. `PREPARE` låser satsen med numrerade parametrar, och `EXECUTE` levererar värdena senare:

```
PREPARE order_lookup (text) AS
    SELECT id, total FROM orders WHERE email = $1;
EXECUTE order_lookup('x'' OR ''1''=''1');
```

Den fientliga indatan, här skriven som en SQL-literal, hamnar i `$1` som en enda underlig e-postadress och hittar inga beställningar.

På en plattform med yrkes- och organisationsdata valideras indata innan något når databasen, och sedan ansluter API:et till databasen som en roll med minsta möjliga behörighet, som bara får anropa appens funktioner, och allt är parametriserat. Bindningen hindrar data från att bli SQL, och den smala rollen begränsar vad ett misstag kan nå.

Var det går fel

Strängen byggs ändå. En f-sträng, `%` -operatorn, `+` eller `.format()` fogar in värdet i SQL-koden innan drivrutinen ser det, och det gör även en platshållare som skrivs inom citattecken, som `'%s'` . OWASP påpekar att den som sätter ihop värden som inte går att lita på innan satsen förbereds, gör hela skyddet verkningslöst, och dokumentationen för psycopg säger att utvecklare aldrig ska använda `%` eller `+` för att foga in värden i databasfrågor. Skicka varje värde som det andra argumentet till `execute()` , med platshållaren utan citattecken.

Tecken som maskeras för hand. Att dubblera citattecken eller lägga till omvända snedstreck (escaping) ser likvärdigt ut, men reglerna beror på SQL-dialekten och dess inställningar. OWASP:s exempel är MySQL-läget `NO_BACKSLASH_ESCAPES`, som stänger av maskering med omvänt snedstreck, och fusklappen avråder starkt från sådan maskering. Ersätt strängar som maskerats för hand med bundna parametrar.

Validering eller en WAF som lösningen. Validering fångar misstag, men fritext måste tillåta de tecken som en angripare behöver: exemplet i CWE-posten är efternamnet O'Reilly, som sannolikt skulle klara valideringen och ändå innehåller ett citattecken. Listor över farliga ord (denylists) har kryphål, och CWE bedömer effekten av en brandvägg för applikationer som bara måttlig. OWASP säger det rakt ut: att ett värde är validerat gör det inte säkert att foga in i SQL. Fortsätt validera för korrekthetens skull, och bind för säkerhetens.

Tabellnamn, kolumnnamn eller sorteringsordning från indata. Platshållare står bara för värden, så en utvecklare som behöver låta en användare välja en kolumn eller riktningen för `ORDER BY` frestas att klistra in texten. OWASP:s svar är att koppla varje val till en liten uppsättning namn som applikationen själv äger och avvisa allt annat, till exempel genom att göra om en begärd riktning till literalen `ASC` eller `DESC`. När ett namn verkligen måste vara dynamiskt kan modulen `psycopg.sql` sätta ihop identifierare, men OWASP påpekar att ett namn inom citattecken inte begränsar vilka tabeller applikationen kan styras mot.

Lagrade procedurer som antas vara säkra. Att anropa en procedur med bundna argument skyddar anropet, inte den SQL-kod som proceduren bygger inuti sig själv. CWE påpekar att lagrade procedurer förhindrar vissa typer av SQL-injektion men inte många andra. Använd statisk SQL inne i databasen, eller bind värdena även i dynamisk SQL; i PostgreSQL innebär det `EXECUTE ... USING` för värden och `format('%I', ...)` för identifierare.

En ORM med strängar insydda. En objektrelationell mappare (object-relational mapper, ORM) binder värden när den används som det är tänkt, och CWE tillskriver persistenslager som Hibernate ett betydande skydd när de används rätt. OWASP tillägger att databasfrågor i Hibernate fortfarande kan angripas när utvecklare sätter ihop indata med frågetexten. Använd ORM:ens parametrar, inte strängbygge.

Ett applikationskonto med administratörsrättigheter. Om kontot kan göra allt kan angriparen det också. OWASP vill se skrivskyddade konton där det bara behövs läsning, en egen databasanvändare för varje webbapplikation, vyer (views) som bara visar de kolumner ett konto behöver, och inga administratörsrättigheter för applikationskonton. När en applikation bara arbetar genom lagrade procedurer ska den få köra dem och ingenting annat. Minsta möjliga behörighet begränsar skadan, men förhindrar inte själva injektionen.

Detaljerade databasfel som visas för användarna. Ett felmeddelande som avslöjar satsens struktur hjälper en angripare att skräddarsy nästa försök, och den felbaserade tekniken i testguiden bygger just på det. CWE rekommenderar minimala felmeddelanden. Visa användaren ett allmänt felmeddelande och behåll detaljerna på servern för teamet.

Tillit till data som redan finns i databasen. Ett värde som sparades säkert kan ändå bli farligt när en senare kodbit klistrar in det i SQL. OWASP säger att dess åtgärder gäller oavsett om data kommer från ett HTTP-anrop, en annan tjänst eller sparades tidigare, och CWE varnar för att senare processer kan glömma att maskera det som sparades. Bind värden som läses från dina egna tabeller precis som du binder det en besökare skriver in.

Läs mer

- CWE-89 in the Common Weakness Enumeration — själva posten om svagheten: konsekvenser, exempel och åtgärderna med deras begränsningar.
- OWASP SQL Injection Prevention Cheat Sheet — parametriserade databasfrågor, lagrade procedurer, dynamiska identifierare och minsta möjliga behörighet.
- OWASP Web Security Testing Guide: Testing for SQL Injection — angreppsklasserna och teknikerna så som en testare möter dem.
- PEP 249: Python Database API Specification v2.0 — gränssnittet som Pythons drivrutiner delar, med de fem platshållarstilarna.
- Psycopg 3: Passing parameters to SQL queries — reglerna för platshållare och rätt sätt att bygga dynamisk SQL.
- PostgreSQL: Command Execution Functions — hur libpq skickar parametrar åtskilda från kommandosträngen.



Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är SQL-injektion?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/sql-injection/