Engineer

SQL-injeksjon

🪡 et angrep som gjør inndata til en del av en databasespørring

Hva det er

SQL-injeksjon, på engelsk *SQL injection* og ofte forkortet SQLi, er angrepet der en applikasjon bygger en SQL-kommando av tekst som noen andre har levert, og databasen ender opp med å lese en del av teksten som SQL i stedet for som en verdi. Common Weakness Enumeration fører svakheten som CWE-89 og beskriver den som en kommando bygd helt eller delvis av inndata utenfra, uten at tegnene som har en særlig betydning i SQL, først er nøytralisert. Forebyggingsarket (*cheat sheet*) fra OWASP sier det samme enklere: angrepet virker når data man ikke kan stole på, blir en del av syntaksen i en spørring.

Roten til problemet er at SQL ikke trekker noen fast grense mellom instruksjonene i en kommando og dataene inni den. Et enkelt anførselstegn avslutter en strengliteral, og alt angriperen skriver etter det, tolkes som mer SQL. To bindestreker starter en kommentar, og dermed kan angriperen kaste resten av setningen utvikleren skrev. Et semikolon avslutter, der databasen og driveren tillater det, én setning og starter en ny. Ingen av dem er eksotiske: de samme tegnene dukker opp i etternavn og fritekst.

OWASPs *Web Security Testing Guide* deler angrepene inn i tre klasser etter hvordan resultatene når angriperen. Angrep i samme kanal (*in-band*) leser svaret gjennom den samme kanalen som ble brukt til å sende det, som regel på applikasjonens egen side. Angrep utenfor kanalen (*out-of-band*) får databasen til å levere resultatene et annet sted, for eksempel over en egen HTTP-forbindelse. Slutningsbaserte eller blinde angrep (*inferential* eller *blind*) får ingen data i det hele tatt, og bygger dem opp igjen ett spørsmål om gangen ut fra hvordan applikasjonen oppfører seg. Veiledningen nevner fem vanlige teknikker på tvers av disse klassene: en `UNION` -operator som henger en ny `SELECT` på den første, boolske betingelser, feil som fremprovoseres med vilje, forespørsler utenfor kanalen, og bevisste tidsforsinkelser.

SQL-injeksjon hører til en større familie. I CWE står den under den generelle injeksjonssvakheten, CWE-74, sammen med krysskripting, og OWASP Top Ten for 2025 plasserer begge i kategorien A05 Injection. Fellestrekket er data som krysser en grense og havner et sted der de leses som instruksjoner. Her er tolkeren databasens SQL-parser, og kuren er å holde hver eneste verdi helt utenfor SQL-teksten.

Bruk begrepet i en beslutning

Bruk «Hva er SQL-injeksjon?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

Skaden avhenger av hva applikasjonens databasekonto har lov til, og det er ofte mye. CWE-89 lister opp å lese data angriperen aldri skulle sett, å komme forbi autentiseringen uten å kjenne et passord, å endre dataene som avgjør hvem som får gjøre hva, å endre eller slette poster, og i noen oppsett å kjøre kommandoer i operativsystemet. OWASPs side om angrepet legger til administrative operasjoner som å slå av databasen, lese innhold fra disken på databaseserveren og ta over databaseserveren som administrator.

Begge kildene regner svakheten som alvorlig. CWE angir sannsynligheten for utnyttelse som høy, og OWASPs side om angrepet beskriver feilen som både lett å finne og lett å utnytte. Svakheten står i hver utgave av CWE Top 25 Most Dangerous Software Weaknesses fra 2019 til 2025. CWE-oppføringen nevner også en bølge i 2008, der én og samme angrepsstreng, som virket mot mange forskjellige programmer, kompromitterte et stort antall webservere og fikk dem til å levere ondsinnet kode.

Forsvaret er derimot godt forstått og billig. Forebyggingsarket fra OWASP gjør forberedte setninger (*prepared statements*) med bundne parametere til det første forsvaret, og CWE anbefaler dem også: SQL-en og verdiene reiser hver for seg, så databasen kan ikke forveksle det ene med det andre. Alt annet, fra minst mulig rettigheter til nøye feilhåndtering, er et ekstra lag som begrenser skaden når en feil slipper gjennom.

Slik ser det ut i praksis

Ta en nettbutikk der siden med ordrehistorikk slår opp bestillinger etter e-postadressen som tastes inn i et felt. Utvikleren bygger SQL-en med en *f-string* i Python:

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

En besøkende skriver `x' OR '1'='1` i feltet. Programmet limer det rett inn, og databasen får dette:

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

Anførselstegnet etter `x` lukker strengen utvikleren åpnet, og `OR`-leddet etter det er alltid sant, så databasen returnerer hver eneste bestilling i tabellen. CWE-89 og testveiledningen bruker det samme alltid sanne trikset, som også kan få en innloggingsside til å slippe inn en besøkende som verken kjenner brukernavnet eller passordet.

Verre inndata avslutter setningen og starter en ny. Eksempelet i CWE-oppføringen skriver inn `name'; DELETE FROM items; --` som varenavn, slik at en ny setning følger etter den første og bindestrekene gjør resten til en kommentar. Om dette kjører, avhenger av databasen og driveren, og flere av dagens grensesnitt avviser det. PostgreSQLs `PQexecParams` tar høyst én SQL-kommando per streng, noe dokumentasjonen regner som en viss beskyttelse mot SQL-injeksjon. Python-modulen `sqlite3` gir en feil hver gang `execute()` får mer enn én setning, og psycopg avviser flere setninger i ett kall når parametere sendes med. Ingenting av dette stopper den alltid sanne betingelsen ovenfor, som bare trenger én setning.

Løsningen er å sende verdien separat. PEP 249, standarden for databasegrensesnitt i Python, lar hver driver oppgi plassholderstilen sin i en modulkonstant som heter `paramstyle`, og lister opp fem stiler: `qmark` (`?`), `numeric` (`:1`), `named` (`:name`), `format` (`%s`) og `pyformat` (`%(name)s`). Modulen `sqlite3` er låst til `qmark` og godtar også `named`. Psycopg, driveren for PostgreSQL, tar `%s` og `%(name)s`:

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

`%s` ser ut som strengformatering i Python, men ingenting formateres: SQL-en og tuppelen med verdier når driveren som to separate argumenter. Psycopg 3 holder dem atskilt hele veien og sender kommandoen og parameterne til serveren hver for seg, i stedet for å flette dem sammen på klienten. Forebyggingsarket fra OWASP viser poenget med sitt eget eksempel på inndata, `tom' or '1'='1` : bundet som parameter er det et merkelig brukernavn som slås opp bokstavelig, og det treffer ingen.

PostgreSQL tilbyr det samme skillet i ren SQL. `PREPARE` låser setningen med nummererte parametere, og `EXECUTE` leverer verdiene senere:

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

De fiendtlige inndataene, her skrevet som en SQL-literal, kommer inn i `$1` som én underlig e-postadresse og finner ingen bestillinger.

På en plattform med faglige og organisatoriske data blir inndata validert før noe rører databasen, og deretter kobler API-et seg til databasen som en rolle med minst mulig rettigheter, som bare kan kjøre appfunksjonene, og alt er parameterisert. Bindingen hindrer at data blir til SQL, og den smale rollen begrenser hva en feil kan nå.

Hvor det går galt

Strengen bygges likevel. En *f-string*, `%` -operatoren, `+` eller `.format()` fletter verdien inn i SQL-en før driveren ser den, og det samme gjør en plassholder skrevet inne i anførselstegn, som `'%s'` . OWASP påpeker at det å sette sammen verdier man ikke kan stole på, før setningen forberedes, ødelegger hele beskyttelsen, og dokumentasjonen til psycopg sier at utviklere aldri skal bruke `%` eller `+` til å flette verdier inn i spørringer. Send hver verdi som det andre argumentet til `execute()` , med plassholderen uten anførselstegn.

Tegn maskert for hånd. Å doble anførselstegn eller legge til omvendte skråstreker (*escaping*) ser likeverdig ut, men reglene avhenger av SQL-dialekten og innstillingene dens. OWASPs eksempel er MySQL-modusen `NO_BACKSLASH_ESCAPES`, som slår av maskering med omvendt skråstrek, og forebyggingsarket fraråder slik maskering på det sterkeste. Erstatt strenger som er maskert for hånd, med bundne parametere.

Validering eller en WAF som løsningen. Validering fanger feil, men fritekst må tillate tegnene en angriper trenger: eksempelet i CWE-oppføringen er etternavnet O'Reilly, som trolig ville passert valideringen og likevel inneholder et anførselstegn. Lister over farlige ord (*denylists*) har smutthull, og CWE vurderer effekten av en brannmur for applikasjoner som bare moderat. OWASP sier det rett ut: at en verdi er validert, gjør den ikke trygg å sette sammen med SQL. Fortsett å validere for korrekthetens skyld, og bind for sikkerhetens.

Tabellnavn, kolonnenavn eller sorteringsrekkefølge fra inndata. Plassholdere står bare for verdier, så en utvikler som må la en bruker velge en kolonne eller retningen på `ORDER BY`, fristes til å lime inn teksten. OWASPs svar er å koble hvert valg til et lite sett med navn som applikasjonen selv eier, og avvise alt annet, for eksempel ved å gjøre en ønsket retning om til literalen `ASC` eller `DESC`. Når et navn virkelig må være dynamisk, kan modulen `psycopg.sql` sette sammen identifikatorer, men OWASP påpeker at et navn i anførselstegn ikke begrenser hvilke tabeller applikasjonen kan styres mot.

Lagrede prosedyrer som antas å være trygge. Å kalle en prosedyre med bundne argumenter beskytter kallet, ikke SQL-en prosedyren bygger inne i seg selv. CWE påpeker at lagrede prosedyrer hindrer noen typer SQL-injeksjon, men ikke mange andre. Bruk statisk SQL inne i databasen, eller bind verdiene også i dynamisk SQL; i PostgreSQL betyr det `EXECUTE ... USING` for verdier og `format('%I', ...)` for identifikatorer.

En ORM med strenger sydd inn. En objektrelasjonell mapper (*object-relational mapper*, ORM) binder verdier når den brukes slik den er tenkt, og CWE tilskriver persistenslag som Hibernate betydelig beskyttelse når de brukes riktig. OWASP legger til at spørringer i Hibernate fortsatt kan angripes når utviklere setter inndata sammen med spørringsteksten. Bruk parameterne i ORM-en, ikke strengbygging.

En applikasjonskonto med administratorrettigheter. Hvis kontoen kan gjøre alt, kan angriperen det også. OWASP ber om skrivebeskyttede kontoer der det bare trengs lesing, en egen databasebruker for hver webapplikasjon, visninger (*views*) som bare viser kolonnene en konto trenger, og ingen administratorrettigheter for applikasjonskontoer. Når en applikasjon bare arbeider gjennom lagrede prosedyrer, bør den få lov til å kjøre dem og ingenting annet. Minst mulig rettigheter begrenser skaden, men hindrer ikke selve injeksjonen.

Detaljerte databasefeil vist til brukerne. En feilmelding som avslører strukturen i setningen, hjelper en angriper med å skreddersy neste forsøk, og den feilbaserte teknikken i testveiledningen bygger nettopp på det. CWE anbefaler minimale feilmeldinger. Vis brukeren en generell feilmelding, og behold detaljene på serveren for teamet.

Tillit til data som allerede ligger i databasen. En verdi som ble lagret trygt, kan likevel bli farlig når en senere kodebit limer den inn i SQL. OWASP sier at tiltakene gjelder enten dataene kommer fra en HTTP-forespørsel, en annen tjeneste eller ble lagret tidligere, og CWE advarer om at senere prosesser kan glemme å maskere det som ble lagret. Bind verdier lest fra dine egne tabeller på samme måte som du binder det en besøkende taster inn.

Les mer

- CWE-89 in the Common Weakness Enumeration — selve oppføringen om svakheten: konsekvenser, eksempler og tiltakene med begrensningene sine.
- OWASP SQL Injection Prevention Cheat Sheet — parameteriserte spørringer, lagrede prosedyrer, dynamiske identifikatorer og minst mulig rettigheter.
- OWASP Web Security Testing Guide: Testing for SQL Injection — angrepsklassene og teknikkene slik en tester møter dem.
- PEP 249: Python Database API Specification v2.0 — grensesnittet Python-drivere deler, med de fem plassholderstilene.
- Psycopg 3: Passing parameters to SQL queries — reglene for plassholdere og riktig måte å bygge dynamisk SQL på.
- PostgreSQL: Command Execution Functions — hvordan libpq sender parametere atskilt fra kommandostrengen.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er SQL-injeksjon?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/sql-injection/