Napadači konstantno pretražuju internet kako bi pronašli SQLi propuste. Došlo je dotle da koriste alate koji automatiziraju potragu za manama u SQLi zaštiti i pokušavaju to iskoristiti za vlastitu finansijsku dobit.

S obzirom da je većina modernih aplikacija danas vođeno podacima (*eng. data driven*), i dostupno na internetu, SQLi propusti su veoma česti i rasprostranjeni.

Kada se jednom iskoristi SQLi propust, napadač može:

* ukrasti, promijeniti i uništiit osjetljive podatke kao što su nečiji podaci korišteni za identifikaciju (kao JMBG, šifre i usernames)
* koristiti privilegije na nivou aplikacije, baze ili čak operativnog sistema u nekim ekstremnijim slučajevima
* koristiti kompromizovani server na kojem se nalazi baza da napada druge sisteme koji se nalaze na istoj mreži

**Paramtetrizovani upiti**

Programski jezici komuniciraju sa SQL bazama podataka pomoću database drivera. Driver omogućava aplikaciji da kreira i pokreće SQL upite na bazi, i da pritom dobija i manipuliše podacima, shodno potrebama. *Parametrizovani* upiti osiguravaju da se parametri odnosno ulazi u SQL upit tretiraju sigurno. Ovo je ubjedljivo najbolji i najjači način zaštite.

**ORM (Object Relational Mapping)**

ORM frameworks mapiraju SQL upite u objekte, tako da developer koji manipuliše nad podacima u bazi efektivno radi samo nad kreiranim objektom, a ne nad samim upitom. Ispod haube, ovi objekti koriste parametrizovane upite nad bazom. Dakle, ORM je suštinski alternativa parametrizovanim upitima, jer radi istu stvar, samo što programer nema obavezu da sam piše parametrizovane upite.

(sequelize je znaci vid zastite, sacuvaj obavezno kod koji napises s njim i onda ga later uvali kao ss)

**Escaping Inputs**

Ukoliko iz nekog razloga nismo u mogućnosti koristiti parametrizovane upite niti ORM framework (iako se to preporučuje), idući način zaštite je escaping inputs. Injection napadi se oslanjaju na to da će napadač prerano zatvoriti string koji se pojavljuje kao argument u SQL upitu. Iz tog razloga često vidimo ‘ ili “ karaktere u SQL napadima.

Ideja mi nije jasna tbh.

**Sanitizing Inputs**

Ovo je veoma dobra praksa za sve tipove aplikacija. Ideja je u tome da proispitujemo polja za šifru ili username na formi. Naime, kada napadač pokušava probiti zaštitu, kao što smo vidjeli u primjeru iznad [dodaj neki primjer bypassa zaštite sa onim fazonom or 1=1--], koristi stringove poprilično sumnjivog izgleda.

Stoga na leđa programera spada obaveza da zabrani sve ulaze u input polja koja izgledaju sumnjivo, recimo pomoću regularnih izraza, osiguravanja da alphanumerička polja ne sadrže simbole i brisanje whitespacea.

**Dodatna zaštita**

Postoje i neki sekundarni vidovi zaštite koji nisu esencijalni kao dosad navedeni, ali se smatraju dobrom praksom i trebalo bi se uložiti vrijeme i trud da se i oni ispoštuju.

* održavati sve softverske komponente (biblioteke, plug-ins, frameworks i slične) ažurnim sa najnovijim prepravkama sigurnosnih detalja koje nam proizvođači nude
* koristiti minimalni broj privilegija - ukoliko je našem web siteu za neometan rad dovoljan samo SELECT izraz da dobija podatke iz baze, ne bismo joj trebali davati privilegije koje će omogućiti INSERT, UPDATE ili DELETE operacije. U mnogo slučajeva, ovo je moguće postići korištenjem rola, pri čemu se mora voditi računa da aplikaciji *nikad* ne dozvolimo da se poveže na bazu podataka kao administrator.
* ne bismo trebali koristiti dijeljenu bazu podataka za više web aplikacija ili stranica.
* osmisliti dobar mehanizam za prijavljivanja server-side grešaka. Ove greške se *nikada* ne bi trebale prikazivati korisnicima, iz razloga što su one pune tehničkih detalja koje bi napadač (koji je također korisnik) mogao iskoristiti da podesi svoje upite i probije zaštitu.

NAPOMENA

Validacija pomoću JavaScripta (odnosno client-side validacija) je dobra stvar kada želimo korisnika aplikacije odmah obavijestiti o tome da je nešto pogriješio/pogrešno popunio polje, međutim ona *ne* predstavlja adekvatnu zaštitu od ozbiljnog hakera. Većina hakera *ne* koristi browser za svoje napade, nego maliciozne skripte.

References

<https://www.hacksplaining.com/prevention/sql-injection>

<https://security.berkeley.edu/resources/best-practices-how-articles/system-application-security/how-protect-against-sql-injection>