SQL INJECTION

Vladimir SIRBU

Departamentul Ingineria Software și Automatică, grupa TI-201FR,Facultatea Calculatoare Informatică și Microelectronică, Universitatea Tehnică a Moldovei, Chișinău, Republica Moldova

Autorul corespondent: Vladimir SIRBU, e-mail: vladimir.sirbu@isa.utm.md

Îndrumătorul/coordonatorul științific **Dorian SARANCIUC**, asistent universitar

Abstract: *În lucrarea dată este descris ce este SQL injection, care sunt cele mai des întâlnite metode de spargere a accounutilor folosite de hackers pentru a ataca contul utilizatorului. SQL injection este descries ca fiind o vulnerabilitate de securitate care apare atunci când un atacator reușește să manipuleze o interogare SQL într-un mod neintenționat de către dezvoltatorul aplicației. Acest lucru poate duce la acces neautorizat, manipularea datelor sau chiar ștergerea acestora într-o bază de date. Atacurile de SQL injection vizează în mod tipic aplicații web care interacționează cu o bază de date. SQL injection poate fi utilizat în orice situație în care o aplicație web interacționează cu o bază de date și nu implementează suficiente măsuri de securitate pentru a proteja împotriva acestui tip de atac. În final, sunt indicate măsurile de protecție împotriva atacurilor de tip SQL Injection.*

Cuvinte cheie: *Securitate, SQL injection, Error-Based, Blind SQL, WAF*

Introducere

Injecția SQL este un tip de atac prin injecție. Atacurile prin injecție apar atunci când intrările create cu răutate sunt trimise de către un atacator, determinând o aplicație să efectueze o acțiune neintenționată. Datorită ubicuității bazelor de date SQL, injecția SQL este unul dintre cele mai comune tipuri de atac pe internet.

În această lucrare, voi arăta și exemple pas cu pas de atacuri comune. Vom începe cu un atac de bază SQL Injection îndreptat către o aplicație web și care duce la escaladarea privilegiilor la rădăcina sistemului de operare.

Dacă intrarea unui utilizator este trecută nevalidată și neigienizată ca parte a unei interogări SQL, utilizatorul poate manipula interogarea în sine și o poate forța să returneze date diferite de ceea ce trebuia să returneze. În acest articol, vedem cum și de ce atacurile SQLi au un impact atât de mare asupra securității aplicațiilor.

Capitolul 1 SQL Injection

În continuare vom vedea cum funcționează SQL injection și câteva măsuri preventive:

Cum funcționează SQL Injection:

1. **Input de la Utilizator:** Aplicațiile web iau adesea input de la utilizatori, cum ar fi prin formulare web sau parametri URL, și folosesc acele inputuri pentru a construi interogări SQL.
2. **Punctul de Injectare:** Dacă aplicația nu validează sau nu curăță corespunzător inputul de la utilizator, un atacator poate injecta cod SQL malițios în câmpurile de input.
3. **Manipularea Interogărilor:** Codul SQL injectat devine parte a interogării executate de baza de date, ducând la consecințe nedorite.

Exemplu:

Să luăm un formular simplu de autentificare:

sqlCopy code

```
SELECT * FROM users WHERE username = 'input_username' AND password = 'input_password';
```

Dacă un atacator introduce ' **OR '1'='1'**; -- ca nume de utilizator și lasă câmpul parolei necompletat, interogarea devine:

sqlCopy code

SELECT * FROM users WHERE username = '' OR '1'='1'; --' AND password = '';

Aceasta trece efectiv de mecanismul de autentificare, deoarece condiția '1'='1' este întotdeauna adevărată.

Ce se întamplă când interacționăm cu o aplicație vulnerabilă, o să explicăm print-un exemplu.

1: Introducem datele noastre

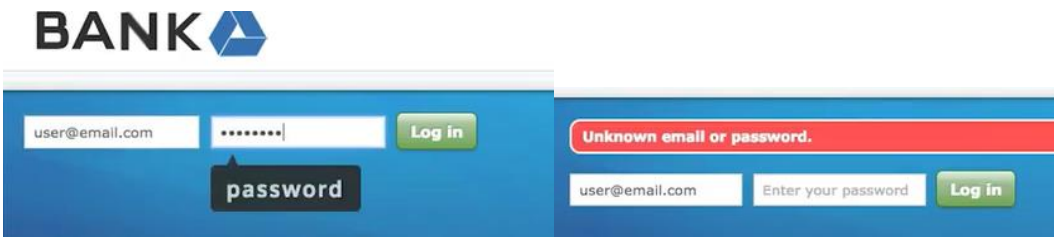


Figura 1. Exemplu de SQL Injection la aplicație vulnerabilă

Mai jos avem codul cum aplicația arată în spatele acestei interfețe. Parola este introdusă direct în șirul SQL, iar terminalul reacționează. Alături vedem eroarea care apare la introducerea caracterelor, ceea ce indică faptul că aplicația este vulnerabilă la SQL injection.

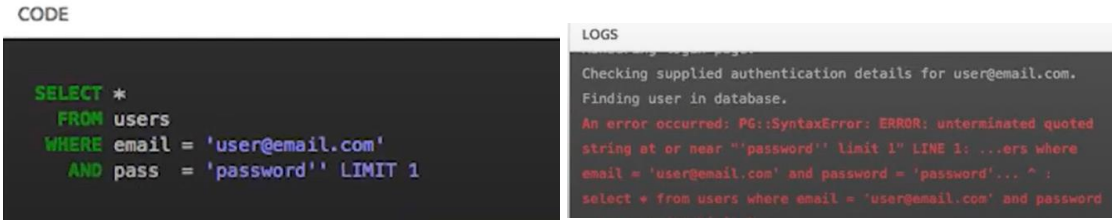


Figura 2. Exemplu de SQL Injection la aplicatie vulnerabile

Mai departe, noi mai încercăm o dată să intrăm și chiar am reușit, dar atrageți atenția la cod – dublă liniuță (--) – care înseamnă că baza de date va ignora restul instrucțiunii SQL. Acest lucru ne permite să ne identificăm fără a introduce parola reală.

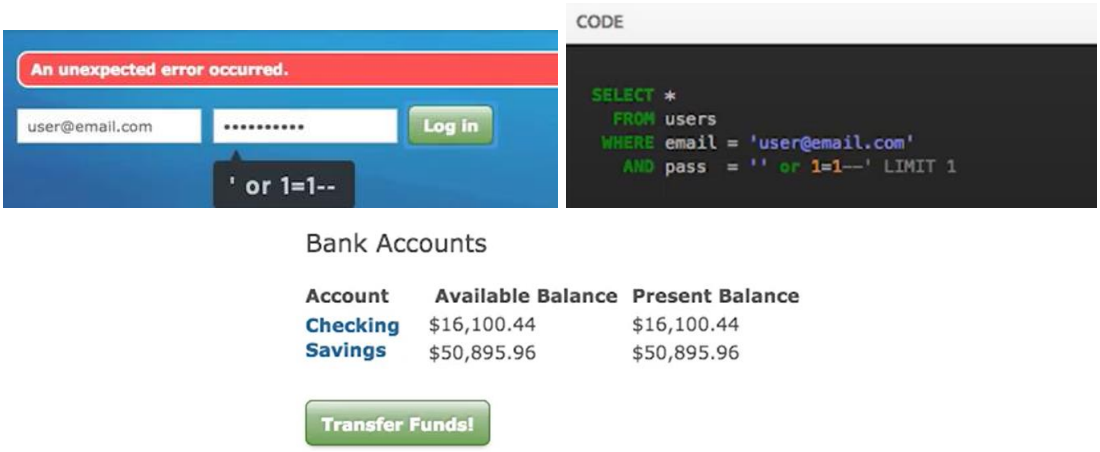


Figura 3. Exemplu de SQL Injection la aplicație vulnerabilă

Care este cel mai rău lucru care s-ar putea întâmpla atunci când suferiți un atac de injecție SQL?

Exemplul nostru de hack v-a arătat cum să ocoliți pagina de autentificare: o defecțiune uriașă de securitate pentru un site bancar. Atacurile mai complexe vor permite unui atacator să execute declarații arbitrare în baza de date. În trecut, hackerii au folosit atacuri prin injecție pentru:

- Extrageți informații sensibile, cum ar fi numerele de securitate socială sau detaliile cardului de credit.
- Enumerați detaliile de autentificare ale utilizatorilor înregistrați pe un site web, astfel încât aceste autentificări să poată fi folosite în atacuri asupra altor site-uri.
- Ștergeți date sau aruncați tabele, corupând baza de date și făcând site-ul inutilizabil.
- Injectați un alt cod rău intenționat pentru a fi executat atunci când utilizatorii vizitează site-ul.

Atacurile de injectare SQL sunt uimitor de frecvente. Companii majore precum Yahoo și Sony și-au compromis aplicațiile. În alte cazuri, grupurile de hackeri au vizat aplicații specifice sau au scris scripturi menite să colecteze detalii de autentificare. Nici măcar firmele de securitate nu sunt imune!

Dacă aveți timp să vă protejați doar de o vulnerabilitate, ar trebui să verificați vulnerabilitățile de injectare SQL în baza de cod!

(Vezi Anexe).

Capitolul 2. Cum funcționează SQL injection și măsuri preventive

Măsuri preventive:

1. **Instrucțiuni parametrizate/Prepared Statements:** Folosiți interogări parametrizate sau instrucțiuni pregătite oferite de limbajul de programare sau biblioteca de bază de date. Aceste metode asigură că inputul de la utilizator este tratat ca date și nu ca cod executabil.
2. **Validare și Curățare a Inputului:** Validați și curățați inputul de la utilizator pentru a vă asigura că respectă formatele așteptate și nu conține cod malițios.
3. **Principiul Celei Mai Mici Privilegii**: Asigurați-vă că conturile de bază de date utilizate de aplicații web au privilegiile minime necesare. Evitați utilizarea conturilor cu drepturi administrative complete.
4. **Firewall-uri de Bază de Date:** Implementați firewall-uri de bază de date sau sisteme de detecție a intruziunilor pentru a monitoriza și bloca activitățile suspecte.
5. **Audituri de Securitate Regulate:** Efectuați audituri de securitate regulate și teste de penetrare pentru a identifica și aborda vulnerabilitățile din aplicație.
6. **Educați Dezvoltatorii:** Instruiți dezvoltatorii să urmeze practici sigure de codificare și să conștientizeze riscurile SQL injection.

Deci injectarea SQL este un risc serios. Cum te poți proteja?

Instrucțiuni parametrizate

Limbajele de programare vorbesc cu bazele de date SQL folosind drivere de baze de date. Un driver permite unei aplicații să construiască și să ruleze instrucțiuni SQL pe o bază de date, extrăgând și manipulând datele după cum este necesar. Instrucțiunile parametrizate se asigură că parametrii (adică intrările) trecuți în instrucțiunile SQL sunt tratați într-un mod sigur.

De exemplu, o modalitate sigură de a rula o interogare SQL în JDBC folosind o instrucțiune parametrizată ar fi:

```java
// Connect to the database.
Connection conn = DriverManager.getConnection(URL, USER, PASS);

// Construct the SQL statement we want to run, specifying the parameter.
String sql = "SELECT * FROM users WHERE email = ?";

// Generate a prepared statement with the placeholder parameter.
PreparedStatement stmt = conn.prepareStatement(sql);

// Bind email value into the statement at parameter index 1.
stmt.setString(1, email);

// Run the query...
ResultSet results = stmt.executeQuery(sql);

while (results.next())
{
    // ...do something with the data returned.
}
```

Figura 4. O interogare SQL în JDBC

Comparați acest lucru cu construcția explicită a șirului SQL, care este foarte, foarte periculoasă:

```java
// The user we want to find.
String email = "user@email.com";

// Connect to the database.
Connection conn = DriverManager.getConnection(URL, USER, PASS);
Statement stmt = conn.createStatement();

// Bad, bad news! Don't construct the query with string concatenation.
String sql = "SELECT * FROM users WHERE email = '" + email + "'";

// I have a bad feeling about this...
ResultSet results = stmt.executeQuery(sql);

while (results.next()) {
  // ...oh look, we got hacked.
}
```

Figura 5. O interogare SQL în JDBC periculoasă

Diferența cheie este că datele sunt transmise metodei executeQuery (...). În primul caz, șirul parametrizat și parametrii sunt transmise în baza de date separat, ceea ce permite șoferului să le interpreteze corect. În al doilea caz, instrucțiunea SQL completă este construită înainte ca driverul să fie invocat, ceea ce înseamnă că suntem vulnerabili la parametrii creați în mod rău intenționat.

Ar trebui să utilizați întotdeauna instrucțiunile parametrizate acolo unde sunt disponibile, acestea sunt protecția dvs. numărul unu împotriva injectării SQL.

Puteți vedea mai multe exemple de instrucțiuni parametrizate în diferite limbi în exemplele de cod de mai jos.

Maparea relațională a obiectelor

Multe echipe de dezvoltare preferă să folosească cadrele ORM (Object Relational Mapping) pentru a face traducerea seturilor de rezultate SQL în obiecte de cod mai simplă. Instrumentele ORM înseamnă adesea că dezvoltatorii vor trebui rar să scrie instrucțiuni SQL în codul lor – iar aceste instrumente folosesc, din fericire, instrucțiuni parametrizate sub capotă.

Cel mai cunoscut ORM este probabil cadrul Ruby on Rails Active Record. Preluarea datelor din baza de date folosind Active Record arată astfel:

```ruby
def current_user(email)
  # The 'User' object is an Active Record object, that has find methods
  # auto-magically generated by Rails.
  User.find_by_email(email)
end
```

Figura 6. Preluarea datelor din baza de date

Un astfel de cod este protejat de atacurile SQL Injection!!!

Cu toate acestea, utilizarea unui ORM nu vă face imun automat la injecția SQL. Multe cadre ORM vă permit să construiți instrucțiuni SQL, sau fragmente de instrucțiuni SQL, atunci când trebuie efectuate operațiuni mai complexe pe baza de date. De exemplu, următorul cod Ruby este vulnerabil la atacurile prin injecție:

```ruby
def current_user(email)
  # This code would be vulnerable to a maliciously crafted email parameter.
  User.where("email = '" + email + "'")
end
```

Figura 7. Preluarea datelor din baza de date vulnerabil la atacurile prin injecție

Ca regulă generală: dacă vă treziți să scrieți instrucțiuni SQL prin concatenarea șirurilor, gândiți-vă foarte atent la ceea ce faceți.

Cum Funcționează SQL Injection:

1. **Tipuri de SQL Injection:**
   - **Classic SQL Injection:** Implică inserarea de instrucțiuni SQL în inputurile aplicației, cum ar fi formularele web sau URL-urile.
   - **Blind SQL Injection:** Atunci când rezultatul interogării nu este vizibil în răspunsul aplicației, dar atacatorul poate deduce informații prin testarea condițiilor „true” sau „false”.
   - **Time-Based Blind SQL Injection:** Similar cu blind SQL injection, dar atacatorul induce întârzieri în răspunsul aplicației pentru a deduce informații.
   - **Error-Based SQL Injection:** Se bazează pe exploatarea erorilor generate de baza de date pentru a obține informații despre structura sau conținutul acesteia.
2. **Exemplu Mai Detaliat:**
   - Considerăm un formular de căutare unde utilizatorii pot introduce un nume de utilizator pentru a obține informații dintr-o bază de date.
   - Inputul utilizatorului nu este validat sau curățat adecvat.
   - Dacă un atacator introduce admin' **OR '1'='1';** --, interogarea rezultantă ar putea deveni: **SELECT * FROM users WHERE username = 'admin' OR '1'='1'; --'**.
   - În acest caz, condiția **'1'='1'** este întotdeauna adevărată, iar atacatorul ar putea obține acces la toate înregistrările din tabelul utilizatorilor.

Măsuri Preventive:

1. **Parametrizarea Interogărilor/Instrucțiuni Pregătite:**
   - Folosiți interogări parametrizate sau instrucțiuni pregătite pentru a separa datele de instrucțiunile SQL.
2. **Validare Riguroasă a Inputului:**
   - Implementați validarea și curățarea adecvată a inputului de la utilizator pentru a preveni inserarea de caractere speciale sau comenzi SQL.
3. **Utilizarea a Drepturilor Minime:**
   - Acordați conturilor de bază de date privilegii minime necesare pentru a executa operațiile dorite, reducând astfel impactul unui atac.
4. **Monitorizare și Jurnalizare:**
   - Implementați sisteme de monitorizare și jurnalizare pentru a detecta activități neobișnuite și pentru a urmări eventuale tentative de atac.
5. **Actualizări și Patch-uri:**
   - Asigurați-vă că sistemul de gestiune a bazelor de date și aplicația sunt actualizate cu cele mai recente patch-uri și actualizări de securitate.
6. **Teste de Securitate Periodice:**
   - Realizați teste de securitate și audituri periodice pentru a identifica și remedia eventualele vulnerabilități.
7. **Educație și Conștientizare:**
   - Educați dezvoltatorii, administratorii de bază de date și utilizatorii cu privire la riscurile SQL injection și la practicile de securitate.

Implementarea acestor măsuri preventive poate contribui la reducerea riscului de a fi afectat de atacuri de SQL injection și la asigurarea securității aplicației și a bazelor de date asociate.

Intrări de evacuare

Dacă nu puteți utiliza instrucțiuni parametrizate sau o bibliotecă care scrie SQL pentru dvs., următoarea abordare cea mai bună este să vă asigurați că evadarea corectă a caracterelor șiruri speciale din parametrii de intrare.

Atacurile prin injecție se bazează adesea pe faptul că atacatorul poate crea o intrare care va închide prematur șirul de argument în care apar în instrucțiunea SQL. (De aceea, veți vedea adesea caracterele „ sau „ în încercările de atacuri cu injecție SQL.)

Limbajele de programare au moduri standard de a descrie șiruri de caractere care conțin ghilimele în ele – SQL nu este diferit în acest sens. În mod obișnuit, dublarea caracterului ghilimele – înlocuirea „cu „ – înseamnă „tratați acest ghilimele ca parte a șirului, nu sfârșitul șirului”.

Evadarea caracterelor simbol este o modalitate simplă de a vă proteja împotriva celor mai multe atacuri de injecție SQL și multe limbi au funcții standard pentru a realiza acest lucru. Există însă câteva dezavantaje ale acestei abordări:

- Trebuie să fii foarte atent să scapi de caractere peste tot în baza de cod unde este construită o instrucțiune SQL.
- Nu toate atacurile prin injectare se bazează pe abuzul de caractere de ghilimele. De exemplu, când este așteptat un ID numeric într-o instrucțiune SQL, ghilimelele nu sunt necesare. Următorul cod este încă vulnerabil la atacurile prin injecție, indiferent cât de mult vă jucați cu ghilimele:

```
def current_user(id)
  User.where("id = " + id)
end
```

Figura 8. Cod cu caractere de ghilimele

Intrări de igienizare

Dezinfectarea intrărilor este o practică bună pentru toate aplicațiile. În exemplul nostru de hack, utilizatorul a furnizat o parolă ca „ sau 1=1--, ceea ce pare destul de suspect ca alegere a parolei.

Dezvoltatorii ar trebui să depună întotdeauna eforturi pentru a respinge intrările care par suspecte din mână, având grijă să nu pedepsească accidental utilizatorii legitimi. De exemplu, aplicația dvs. poate curăța parametrii furnizați în solicitările GET și POST în următoarele moduri:

- Verificați dacă câmpurile furnizate, cum ar fi adresele de e-mail, corespund unei expresii regulate.
- Asigurați-vă că câmpurile numerice sau alfanumerice nu conțin caractere simbol.
- Respingeți (sau eliminați) spațiile albe și caracterele de linie nouă acolo unde acestea nu sunt adecvate.

Validarea la nivelul clientului (adică în JavaScript) este utilă pentru a oferi utilizatorului feedback imediat atunci când completează un formular, dar nu reprezintă o apărare împotriva unui hacker serios. Cele mai multe încercări de hack sunt efectuate folosind scripturi, mai degrabă decât browserul în sine.

Principiul celui mai mic privilegiu

Aplicațiile ar trebui să se asigure că fiecare proces sau componentă software poate accesa și afecta doar resursele de care are nevoie. Aplicați „niveluri de autorizare” după caz, în același mod în care doar anumiți angajați ai băncii au acces la seif. Aplicarea privilegiilor restricționate poate ajuta la atenuarea multor riscuri din jurul atacurilor de injecție.

Rareori este necesar ca aplicațiile să schimbe structura bazei de date în timpul rulării – de obicei, tabelele sunt create, abandonate și modificate în timpul ferestrelor de lansare, cu permisiuni temporar ridicate. Prin urmare, este o bună practică să reduceți permisiunile aplicației în timpul execuției, astfel încât să poată edita cel mult date, dar să nu modifice structurile tabelelor. Într-o bază de date SQL, aceasta înseamnă să vă asigurați că conturile dvs. de producție pot executa numai instrucțiuni DML, nu instrucțiuni DDL.

Cu bazele de date complexe, poate merita să faceți aceste permisiuni și mai precise. Multe procese pot fi autorizate să efectueze editări de date numai prin proceduri stocate sau să se execute cu permisiuni numai pentru citire.

Proiectarea rațională a managementului accesului în acest fel poate oferi o a doua linie de apărare vitală. Indiferent de modul în care atacatorul obține acces la sistemul dvs., acesta poate atenua tipul de daune pe care le pot face.

Hashing parole

Exemplul nostru de hack s-a bazat pe faptul că parola a fost stocată ca text simplu în baza de date. De fapt, stocarea parolelor necriptate este un defect major de securitate în sine. Aplicațiile ar trebui să stocheze parolele utilizatorului ca hashuri puternice, unidirecționale, de preferință sărate. Acest lucru atenuează riscul ca utilizatorii rău intenționați să fure acreditări sau să uzurpare identitatea altor utilizatori.

Autentificare terță parte

Ca o notă finală, deseori merită să luați în considerare externalizarea fluxului de lucru de autentificare al aplicației dvs. Facebook, Twitter și Google oferă toate API-uri OAuth mature, care pot fi folosite pentru a permite utilizatorilor să se conecteze la site-ul dvs. folosind conturile existente pe acele sisteme. Acest lucru vă scutește, ca dezvoltator de aplicații, de a vă rula propria autentificare și vă asigură utilizatorilor că parolele lor sunt stocate doar într-o singură locație.

Capitolul 3: Unde putem folosi?

SQL injection poate fi utilizat în orice situație în care o aplicație web interacționează cu o bază de date și nu implementează suficiente măsuri de securitate pentru a proteja împotriva acestui tip de atac. Aici sunt câteva exemple de locuri în care SQL injection poate fi aplicat:

1. **Formulare de Autentificare:**
   - Atacatorii pot încerca să compromită sistemul de autentificare prin injectarea de comenzi SQL în câmpurile de utilizator și parolă.
2. **Formulare de Căutare:**
   - Dacă o aplicație permite utilizatorilor să efectueze căutări în baza de date, inputul de căutare poate fi folosit pentru a injecta comenzi SQL.
3. **Formulare de Înregistrare:**
   - În cazul în care o aplicație permite utilizatorilor să-și înregistreze informațiile, un atacator poate încerca să injecteze comenzi SQL în câmpurile de înregistrare.
4. **Parametri URL:**
   - Aplicațiile care transmit parametri prin URL ar trebui să fie atent validate pentru a preveni SQL injection.
5. **Comenzi de Filtrare sau Sortare:**
   - Aplicațiile care permit utilizatorilor să filtreze sau să sorteze rezultatele afișate pot fi vulnerabile la SQL injection dacă inputul nu este gestionat corespunzător.
6. **Sisteme de Raportare și Afisare a Datelor:**
   - Aplicațiile care generează rapoarte sau afișează date din baza de date pot fi expuse la SQL injection dacă nu se aplică măsuri adecvate de securitate.
7. **Feedback și Comentarii:**
   - Sistemele care permit utilizatorilor să trimită feedback sau să adauge comentarii pot fi vulnerabile dacă nu se filtrează sau validează corespunzător datele introduse.
8. **Sisteme de Gestiune a Conținutului (CMS):**
   - Platformele CMS care interacționează cu o bază de date pot fi ținte pentru SQL injection, în special dacă permit input din partea utilizatorilor pentru gestionarea conținutului.
9. **Platforme de Comerț Electronic:**
   - Sistemele de comerț electronic care utilizează baze de date pentru a stoca informații despre produse, clienți și tranzacții pot fi vulnerabile la SQL injection.

Concluzii

În concluzie, SQL injection reprezintă o amenințare semnificativă la adresa securității sistemelor de gestionare a bazelor de date și a aplicațiilor web. Atacurile de SQL injection pot avea consecințe grave. Abordarea acestor riscuri necesită eforturi considerabile la nivelul dezvoltării software și al managementului bazelor de date.

În ceea ce privește dezvoltarea aplicațiilor, este crucial să se adopte practici securitare precum parametrizarea interogărilor SQL, validarea riguroasă a inputurilor și implementarea de mecanisme adecvate de autentificare și autorizare. Educația continuă a dezvoltatorilor în ceea ce privește amenințările de securitate și implementarea unor coduri sigure sunt esențiale pentru prevenirea și detectarea timpurie a vulnerabilităților.

Pe de altă parte, administrarea eficientă a bazelor de date presupune implementarea unor politici stricte de securitate, monitorizarea constantă a activităților suspecte și actualizări regulate ale sistemelor de gestionare a bazelor de date pentru a remediat posibilele vulnerabilități. O atenție deosebită trebuie acordată gestionării accesului la bazele de date, asigurându-se că doar utilizatorii autorizați au acces la informații sensibile.

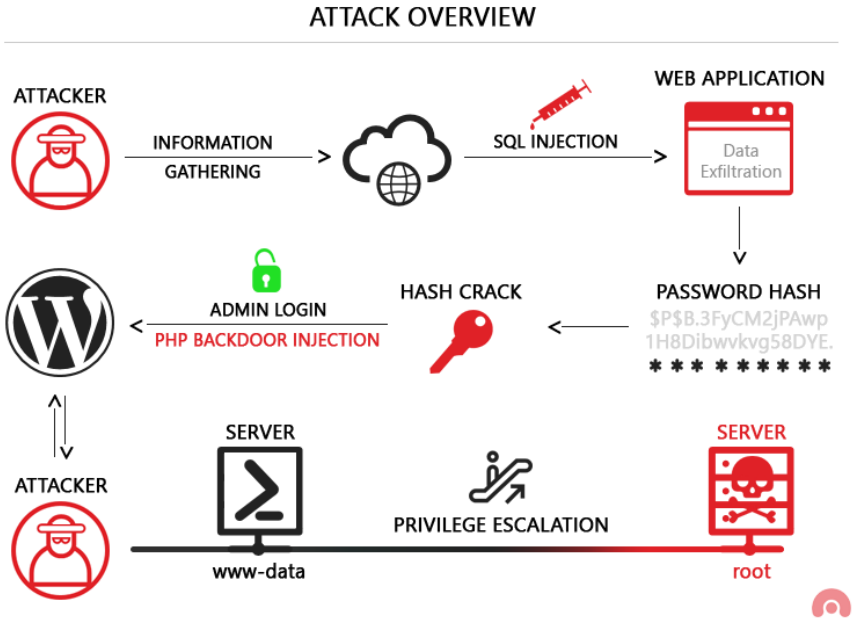
Bibliografie

[1] „Hacksplaining”, [Online]. Available: https://www.hacksplaining.com/prevention/sql-injection

[2] „Exploiting SQL Injection: a Hands-on Example”, [Online]. Available: https://www.acunetix.com/blog/articles/exploiting-sql-injection-example/?utm_source=hacksplaining&utm_medium=post&utm_campaign=articlelink

[3] „SQL injection cheat sheet”, [Online]. Available: https://www.invicti.com/blog/web-security/sql-injection-cheat-sheet/?utm_source=hacksplaining&utm_medium=post&utm_campaign=articlelink
[4] „A webcomic of romance,sarcasm, math, and language”, [Online]. Available: https://xkcd.com/327/

Anexe



Vulnerability Classification and Severity Table

Classification	ID / Severity
PCI v3.1	6.5.1
PCI v3.2	6.5.1
OWASP 2013	A1
CWE	89
CAPEC	66
WASC	19
HIPAA	164.306(a), 164.308(a)

CVSS 3.0 Score	
Base	10 (Critical)
Temporal	10 (Critical)
Environmental	10 (Critical)
CVSS Vector String	
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H	