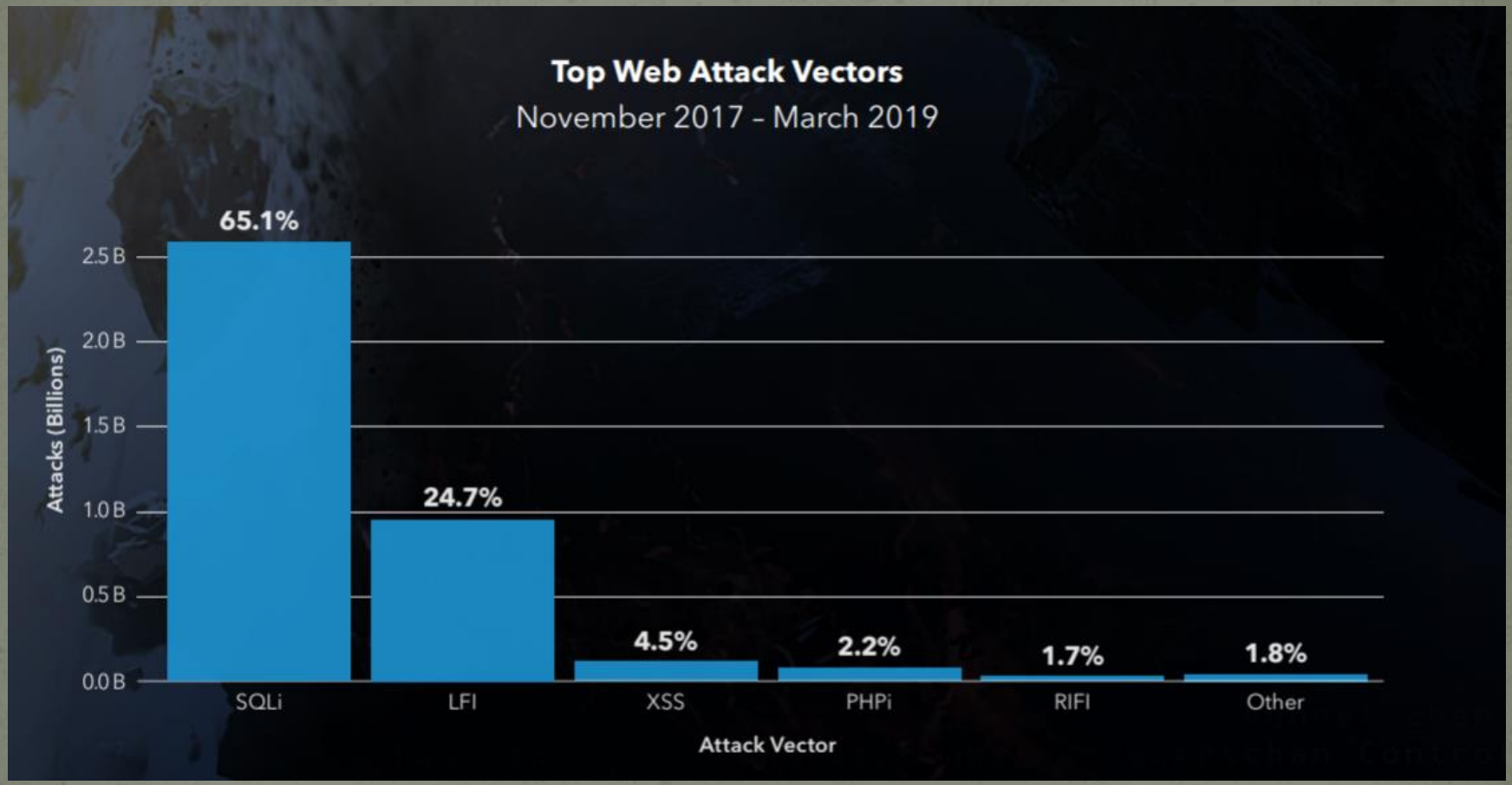
Curs 7

SQL Injection

Privire de ansamblu

„Știința se răzbună ca o femeie,
nu când o ataci, ci când o negliezi.”
Grigore C. Moisil

Introducere



SQLi (SQL injection); LFI (Local File Inclusion); XSS (Cross Site Scripting); RFI (Remote File Inclusion)
https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-web-attacks-and-gaming-abuse-report-2019.pdf

Introducere

- Majoritatea dezvoltatorilor subestimează riscul atacurilor cu SQL Injection ce afectează aplicațiile cu baze de date Oracle în spate.
- Se pare că mulți dezvoltatori de aplicații nu înțeleg în întregime riscul atacurilor cu SQL Injection și nici tehnicile folosite pentru prevenirea unor astfel de atacuri.
- Scopul este sublinierea riscurilor atacurilor cu SQL Injection și demonstrarea premiselor de vulnerabilitate ale aplicațiilor web.
- Nu se vrea a fi un tutorial pentru realizarea de atacuri SQL și nici nu oferă nici instrucțiuni în această privință.

Privire de ansamblu asupra SQL Injection

- SQL Injection este un atac de bază folosit fie pentru obținerea accesului neautorizat la o bază de date, fie pentru extragerea informațiilor direct din baza de date.
- Orice program sau aplicație pot fi vulnerabile la SQL Injection, inclusiv procedurile stocate executate în mod direct printr-o conexiune la o bază de date, aplicații în Oracle Forms, aplicații web, etc.
- Numeroase vulnerabilități au fost descoperite în pachetele standard Oracle Database cum ar fi DBMS_DATAPUMP, DBMS_REGISTRY și DBMS_METADATA (vezi update-ul de importanță critică din ianuarie 2006).
- Aplicațiile web prezintă cel mai mare risc al acestor atacuri de vreme ce un atacator poate exploata vulnerabilitățile la SQL Injection de la distanță fără autentificare la nivel de bază de date sau la nivel de aplicație.

Privire de ansamblu asupra SQL Injection

- Aplicațiile web care au în spate o bază de date Oracle sunt mai vulnerabile la atacurile cu SQL Injection decât ar crede majoritatea dezvoltatorilor. În urma unor auditări de aplicații au fost descoperite multe aplicații web vulnerabile la SQL Injection.
- Atacurile SQL Injection pe baza funcțiilor reprezintă cea mai mare preocupare, de vreme ce aceste atacuri nu necesită informații despre aplicație și pot fi ușor automatizate.
- Atacurile SQL Injection sunt ușor de combătut prin anumite procedee de codare.
- Orice parametru transmis oricărui enunț SQL dinamic trebuie validat, sau trebuie folosite variabile legate.

SQL Injection: Oracle versus alte sisteme

- Oracle se descurcă bine în general împotriva atacurilor SQL Injection de vreme ce nu există suport pentru cereri SQL multiple (SQL Server și PostgreSQL), nu există cererea EXECUTE (SQL Server), și nu există funcția INTO OUTFILE (MySQL) – toate acestea fiind metode de exploatare a vulnerabilităților la SQL Injection.
- Utilizarea variabilelor legate în mediile Oracle din motive de performanță oferă cea mai eficientă protecție împotriva atacurilor SQL Injection.

SQL Injection

„Răutatea calculată este
cea mai ascuțită dintre toate răutățile.”
Honore de Balzac

Introducere

- Atacurile prin SQL Injection sunt în esență simple – un atacator strecoară un șir de caractere în aplicație în speranța că va reuși să manipuleze cererea SQL în avantajul său.
- Complexitatea atacului implică exploatarea unei cereri SQL care poate fi necunoscută atacatorului.
- Aplicațiile open-source și cele comerciale livrate alături codul sursă sunt mai vulnerabile de vreme ce atacatorul poate descoperi potențiale vulnerabilități înaintea atacului.

Categoriile de atacuri cu SQL Injection

- Sunt patru categorii principale pentru atacurile SQL Injection împotriva bazelor de date Oracle:

  - Manipularea SQL
  - Injectarea codului
  - Injectarea apelurilor de funcții
  - Supraîncărcarea bufferelor.

Ce este și ce nu este vulnerabil

- O aplicație este vulnerabilă la SQL Injection dintr-un singur motiv – inputul de stringuri nu este validat corespunzător și este trecut către o cerere SQL dinamică fără vreo astfel de validare.
- Din cauza lipsei de „memorie” a multor aplicații web, este frecventă scrierea datelor în baza de date sau stocarea acestora prin alte mijloace între paginile web.
- Cererile SQL care folosesc variabile legate sunt în general protejate împotriva SQL Injection de vreme ce baza de date folosește valoarea variabilei legate exclusiv și nu interpretează variabila în niciun fel. PL/SQL și JDBC permit variabile legate. Variabilele legate ar trebui să fie utilizate pe scară largă din rațiuni de securitate și performanță.

Tipuri de SQL Injection

„Răutatea este neîmblânzită,
chiar dacă îi faci cel mai mare bine.”
Aesop

Manipularea codului SQL

- Manipularea codului SQL reprezintă cea mai raspândită metodă de atac de tip SQL Injection. Cel mai adesea, această metodă presupune modificarea instrucţiunii SQL prin adăugarea de elemente clauzei WHERE sau extinderea instrucţiunii cu ajutorul operatorilor UNION, INTERSECT, sau MINUS
- O aplicație web ar putea să facă autentificarea unui utilizator prin executarea query-ului de mai jos și apoi verificarea daca acesta a întors vreun rezultat:

```
SELECT * FROM users
WHERE username = 'bob' and PASSWORD = 'mypassword'
```

Manipularea codului SQL

- Manipulând instrucțiunea de mai sus, ca în exemplul de mai jos, bazându-se pe precendența operatorilor, clauza WHERE poate deveni adevarată pentru orice linie, astfel atacatorul primind acces la datele aplicației.

```
SELECT * FROM users
WHERE username = 'bob' and PASSWORD =
'mypassword'
        or 'a' = 'a'
```

- Operatorul pentru mulțimi, UNION, este folosit adesea în atacurile de tip SQL Injection, scopul fiind acela de a manipula o instrucțiune SQL astfel încât să returneze rânduri dintr-un alt tabel.

```
SELECT product_name FROM all_products
WHERE product_name like '%Chairs'
UNION
SELECT username FROM dba_users
WHERE username like '%'
```

Code Injection

- Atacurile de tip code injection încearcă adăugarea unor instrucțiuni sau comenzi SQL adiționale. Acest tip de atac este utilizat adesea împotriva aplicațiilor de tip Microsoft SQL Server, dar funcționează rar în cazul aplicațiilor ce folosesc o bază de date Oracle. Instrucțiunea *EXECUTE* folosită de SQL Server reprezintă o țintă frecventă a atacurilor de tip SQL Injection, Oracle neavând însă o instrucțiune corespunzătoare acesteia.
- atacul de mai jos nu va funcționa într-o aplicație PL/SQL sau Java ce are în spate o bază de date Oracle, și va returna o eroare:

```
SELECT * FROM users
WHERE username = 'bob' and PASSWORD = 'mypassword';
DELETE FROM users
WHERE username = 'admin';
```

- Însă există limbaje de programare sau diferite API-uri care permit executarea instrucțiunilor SQL multiple.
- Un atacator ar putea să încerce manipularea blocului PL/SQL astfel:

```
BEGIN ENCRYPT PASSWORD('bob', 'mypassword');
DELETE FROM users
WHERE upper(username) = upper('admin'); END;
```

Injectare cu apel funcție

- Orice instrucțiune SQL dinamică este vulnerabilă, astfel încât chiar și cele mai simple instrucțiuni SQL pot fi exploatate, așa cum arată exemplul de mai jos.

```
SELECT TRANSLATE('user input',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789') FROM dual;
```

- . Atacatorul poate manipula instrucțiunea SQL astfel încât să se execute ca mai jos, ceea ce va duce la executarea cererii de afișare a unei pagini de pe un server web. De asemenea atacatorul ar putea manipula URL-ul pentru a include și alte funcții ce ar putea returna informații din baza de date ce vor fi apoi trimise serverului web din cadrul URL-ului.

```
SELECT TRANSLATE('' ||
UTL_HTTP.REQUEST('http://192.168.1.1/') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789')
FROM dual;
```

Injectare cu apel funcție

- Întrucât este permisă executarea funcțiilor custom și funcțiilor din pachete custom, acest lucru face ca anumite instrucțiuni SQL să fie vulnerabile. Un exemplu este reprezentat de o aplicație ce are definită o funcție ADDUSER în pachetul custom MYAPPADMIN. Funcția a fost marcată "*PRAGMA TRANSACTION*", pentru a putea fi executată în orice situație specială ce ar putea apărea. Datorită acestui lucru, această funcție poate face scrieri în baza de date chiar și din cadrul unei instrucțiuni *SELECT*, cum se observă în exemplul de mai jos în care atacatorul poate crea noi utilizatori ai aplicației.

```
SELECT TRANSLATE('
      ' || myappadmin.adduser('admin', 'newpass') ||
'',
     '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789')
FROM dual;
```

Supraîncărcarea bufferului

- Atacurile de forma supraîncărcarea bufferului sunt folosite pentru a corupe execuţia unei aplicaţii. Prin trimiterea unor date de intrare atent construite, un atacator poate face ca aplicaţia să execute cod arbitrar, şi poate duce de asemenea la oprirea funcţionării aplicaţiei.
- Anumite funcţii standard Oracle sunt vulnerabile la supraîncărcarea bufferului, lucru ce poate fi exploatat de către un atac SQL Injection. Printre funcţiile standard Oracle vulnerabile se numără: BFILENAME (Oracle8i, Oracle9i), FROM_TZ (Oracle9i), NUMTODSINTERVAL (Oracle8i, Oracle9i), NUMTOYMINTERVAL (Oracle8i, Oracle9i), TO_TIMESTAMP_TZ (Oracle9i), TZ_OFFSET (Oracle9i).
- Pentru a împiedica atacurile de acest gen, este recomandat ca pachetele neesenţiale lucrului cu baze de ate (DBMS_*, UTL_*) să nu fie accesibile utilizatorilor aplicaţiei.

PL/SQL

„Binele trebuie făcut așa cum încercăm să facem răul: pe ascuns.”

John Petit

Instrucțiunea *execute immediate*

- Un exemplu de instrucțiune ***execute immediate*** ce este vulnerabilă atacurilor SQL injection poate arăta astfel:

```
CREATE OR REPLACE PROCEDURE demo(name IN VARCHAR2) AS
   sqlstr VARCHAR2(1000);
   code VARCHAR2(100);
BEGIN
...
sqlstr := 'SELECT postal-code FROM states WHERE state-name = ''' ||
   name || '''';
EXECUTE IMMEDIATE sqlstr INTO code;
IF code = 'IL' THEN ...
...
END;
```

Instrucțiunea *execute immediate*

- Pentru a preveni atacurile SQL Injection și pentru a îmbunătăți performanța aplicației, ar trebui folosite întotdeauna cereri parametrizate.

```
CREATE OR REPLACE PROCEDURE demo(name IN VARCHAR2)
  AS
  sqlstr VARCHAR2(1000);
  code VARCHAR2(100);
BEGIN
...
sqlstr := 'SELECT postal-code FROM states WHERE
  state-name = :name';
EXECUTE IMMEDIATE sqlstr USING name INTO code;
IF code = 'IL' THEN ...
...
END;
```

Instrucțiunea *execute immediate*

- Instrucțiunea ***execute immediate*** poate utiliza și blocuri PL/SQL anonime, ceea ce o face vulnerabilă la atacuri SQL Injection întrucât un atacator ar putea încerca inserarea mai multe instrucțiuni SQL.

```
CREATE OR REPLACE PROCEDURE demo(value IN
  VARCHAR2) AS
BEGIN
...
-- vulnerable
EXECUTE IMMEDIATE 'BEGIN updatepass(''' || value
  || '''); END;'; -- not vulnerable
cmd := 'BEGIN updatepass(:1); END;';
EXECUTE IMMEDIATE cmd USING value;
...
END;
```

Pachetul DBMS_SQL

- Procedura prezentată mai jos folosește pachetul DBMS_SQL și este vulnerabilă la atacuri:

```
CREATE OR REPLACE PROCEDURE demo(name IN VARCHAR2) AS
  cursor_name INTEGER;
  rows_processed INTEGER;
  sqlstr VARCHAR2(150);
  code VARCHAR2(2);
BEGIN
...
sqlstr := 'SELECT postal-code FROM states WHERE state-name
  = ''' || name || '''';
cursor_name := dbms_sql.open_cursor;
DBMS_SQL.PARSE(cursor_name, sqlstr, DBMS_SQL.NATIVE);
  DBMS_SQL.DEFINE_COLUMN(cursor_name, 1, code, 10);
  rows_processed :=
        DBMS_SQL.EXECUTE(cursor_name);
  DBMS_SQL.CLOSE_CURSOR(cursor_name);
...
END;
```

Pachetul DBMS_SQL

- În schimb, aceeași procedură care folosește cereri parametrizate nu mai este susceptibilă atacurilor, cum se vede în exemplul de mai jos:

```
CREATE OR REPLACE PROCEDURE demo(name IN VARCHAR2) AS
cursor_name INTEGER;
rows_processed INTEGER;
sqlstr VARCHAR2;
code VARCHAR2;
BEGIN
...
sqlstr := 'SELECT postal-code FROM states WHERE state-name
  = :name'; cursor_name := dbms_sql.open_cursor;
DBMS_SQL.PARSE(cursor_name, sqlstr, DBMS_SQL.NATIVE);
  DBMS_SQL.DEFINE_COLUMN(cursor_name, 1, code, 10);
  DBMS_SQL.BIND_VARIABLE(cursor_name, ':name', name);
  rows_processed :=     DBMS_SQL.EXECUTE(cursor_name);
  DBMS_SQL.CLOSE_CURSOR(cursor_name);
...
END;
```

Cursoare dinamice

- PL/SQL permite folosirea cursoarelor statice și a celor dinamice. La fel ca și instrucțiunea ***execute immediate*** sau instrucțiunile DBMS_SQL, instrucțiunile cursor pot fi vulnerabile la atacuri cum se vede în exemplul de mai jos. Însă ele pot fi generate în mod dinamic prin folosirea cererilor parametrizate, ceea ce le protejează de atacurile SQL Injection.

```
CREATE OR REPLACE PROCEDURE demo(name IN VARCHAR2) AS
sqlstr VARCHAR2;
...
BEGIN
...
sqlstr := 'SELECT * FROM states WHERE state-name = ''' || name ||
  '''';
OPEN cursor_states FOR sqlstr;
LOOP
  FETCH cursor_states INTO rec_state
  EXIT WHEN cursor_states%NOTFOUND;
  ...
 END LOOP;
CLOSE cursor_status;
...
END;
```

JDBC

„Lenevia, ca și rugina,
roade viața mai mult decât munca.”
Benjamin Franklin

PreparedStatement

- O instrucțiune *PreparedStatement* care e vulnerabilă la SQL Injection este prezentată în exemplul de mai jos:

```
String name = request.getParameter("name");
PreparedStatement pstmt =
conn.prepareStatement("insert into EMP (ENAME) values ('" +
  name + "')"); pstmt.execute();
 pstmt.close();
```

- Pentru a preveni SQL Injection, o variabilă de legătură trebuie să fie folosită:

```
PreparedStatement pstmt =
conn.prepareStatement ("insert into EMP (ENAME) values (?)");
String name = request.getParameter("name");
pstmt.setString (1, name);
pstmt.execute();
pstmt.close();
```

CallableStatement

- Procedura stocată

```
prepareCall( "{call proc (?,?)}" );
```

Bloc PL/SQL anonim

```
prepareCall("begin proc1(?,?); ? := func1(?);
                                ...; end;");
```

- Mai jos este prezentat un bloc PL/SQL anonim, care este vulnerabil la atacuri:

```
String name = request.getParameter("name");
String sql = "begin ? := GetPostalCode('" + name +
  "'); end;";
CallableStatement cs = conn.prepareCall(sql);
cs.registerOutParameter(1, Types.CHAR);
cs.executeUpdate();
String result = cs.getString(1);
cs.close();
```

CallableStatement

- În cadrul unui atac, inputul ar putea fi cu mult diferit față de cel așteptat de către dezvoltator:

```
begin ? := GetPostalCode('Illinois'); end;
begin ? := GetPostalCode(''); delete from users; commit;
  dummy(''); end;
begin ? :=
  GetPostalCode(''||UTL_HTTP.REQUEST('http://192.168.1.1/'
  )||''); end;
```

- Cea mai simplă metodă de a preveni acest lucru este de a folosi cereri parametrizate:

```
String name = request.getParameter("name");
CallableStatement cs = conn.prepareCall ("begin ? :=
                    GetStatePostalCode(?); end;");
  cs.registerOutParameter(1,Types.CHAR);
cs.setString(2, name);
cs.executeUpdate();
String result = cs.getString(1);
cs.close();
```

Prevenirea *SQL Injection* în Oracle

„Anticiparea nenorocirilor viitoare atenuează sosirea lor, căci le vedem cu mult înainte de a veni.”

Marcus Tullius Cicero

Prevenirea *SQL Injection* în Oracle

- Cereri parametrizate
  - Interogările parametrizate tratează datele introduse ca pe niște valori făcând parte din comenzile SQL, în acest fel făcând imposibil ca serverul să trateze interogările parametrizate ca și cod executabil. Chiar dacă se utilizează procedurile stocate, parametrizarea inputului este necesară, pentru că procedurile stocate nu oferă protecție împotriva *SQL Injection*.
  - Cererile parametrizate trebuie folosite pentru fiecare comandă SQL indiferent de când sau unde este executată această comandă.
- **Validarea datelor de intrare**
  - Orice date de intrare introduse de către utilizatori, fie direct în aplicația web sau deja stocate, trebuie să fie validate după tipul serverului, lungime sau format, înainte de a fi trimise mai departe.
  - În cazul bazelor de date Oracle, caracterul ce ar putea pune în pericol securitatea bazei de date, este apostroful. Oracle interpretează ghilimelele simple consecutive ca un literal SQL, iar cea mai simplă metodă de a rezolva această problemă este de a le elimina.

Prevenirea *SQL Injection* în Oracle

- **Securitatea funcțiilor apelate**
  - Oracle are la bază sute de funcții standard, care implicit, pot avea acordate drepturi *PUBLIC*. Aplicația trebuie să conțină funcții suplimentare care să execute operații precum schimbarea parolelor sau crearea unor useri care ar putea să fie exploatați printr-un eventual atac.
  - Toate funcțiile care nu sunt absolut necesare aplicației ar trebui limitate.
- Mesajele de eroare
  - Acesta este punctul de plecare pentru cele mai populare atacuri *SQL Injection*.
  - Însă, ascunderea mesajelor de eroare nu are ca efect oprirea atacului. În cazul în care un atac eșuează, atacatorul poate utiliza informația din mesajele de eroare furnizate de server sau aplicație, pentru a lansa un alt atac.
  - Un atac ce utilizează vulnerabilitatea *SQL Injection* ar putea dezvălui structura unui tabel, a bazei de date, sau să expună logica de interogare, și posibil chiar parole sau informații sensibile utilizate în interogare.

Prevenirea *SQL Injection* în Oracle

- Pentru a evita un atac *SQL Injection* bazat pe mesajele de eroare primite de la serverul aplicației, programatorul trebuie să aibă în vedere:
  - mesajele de eroare să conțină cât mai puține de detalii cu privire la baza de date, care pot fi utilizare pentru a compromite întreg sistemul.
  - mesajele de eroare nu trebuie să conțină numele metodelor, funcțiilor în care a avut loc eroarea.
  - mesajele de eroare trebuie păstrate într-un fișier log, însă trebuie restricționat accesul la ele pentru a evita un posibil nou atac.
  - în fișierele log nu trebuie reținute informații precum parolele userilor.
  - mesajele de eroare nu trebuie să conțină informații privind evenimente interne ale sistemului, ca de exemplu, în caz de logare nereușită, utilizatorul nu trebuie să fie informații dacă există sau nu userul respectiv, ci doar că autentificarea nu a fost posibilă.

Excepții

„Nu există norme. Toți oamenii
sunt excepții ale unei legi care nu există.”
Fernando Pessoa

Excepții

- **Nume de tabele dinamice**
  - Orice tabel dinamic sau nume de coloană trebuie să fie validate, iar toate caracterele invalide ar trebui eliminate, în special apostrofurile. Pentru aceasta, în PL/SQL, poate fi utilizată funcția *TRANSLATE*:

```
translate ( upper ( <input string> ),
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890_#$@.
  `~!%^*()-=+{}[];":''?/><,|\',
'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890_#$@.');
```

Excepții

- **Clauza LIKE**
  - Următoarea concatenare nu ar trebui utilizată:

```
String name = request.getParameter("name");
conn.prepareStatement("SELECT id
    FROM users
    WHERE name LIKE '%" + name + "%'");
```

  - Mai degrabă se folosesc comenzi multiple și o cerere parametrizată pentru a crea în mod corespunzător o instrucțiune SQL

```
String name = request.getParameter("name");
name = query.append("%").append(name).append("%");
pstmt = conn.prepareStatement("SELECT id
FROM users
WHERE name LIKE ?");
pstmt.setString (1, name);
```

Funcții Oracle

„Nu este îndeajuns să ai o minte bună.
Scopul principal e să o folosești bine.”
Rene Descartes

Funcții Oracle

- **Determinarea privilegiilor funcțiilor**
  - Toate funcțiile disponibile PUBLIC pot fi găsite prin următoarea cerere:

```
SELECT *
FROM dba_tab_privs p, all_arguments a
WHERE grantee = 'PUBLIC'
AND privilege = 'EXECUTE'
AND p.table_name = a.package_name
AND p.owner = a.owner
AND a.position = 0
AND a.in_out = 'OUT'
ORDER BY p.owner, p.table_name, p.grantee;
```

- **Restricționarea accesului la funcții**
  - Accesul la o anumită funcție dintr-un pachet nu poate fi restricționat, ci doar accesul la întreg pachetul. Pentru a restricționa accesul PUBLIC la un pachet se folosește următoarea comandă:

```
REVOKE EXECUTE ON <package_name> FROM public;
```

Funcții Oracle

- **Funcții standard**
  - Supraîncărcarea bufferelor a fost descoperită în câteva funcții standard ORACLE: BFILENAME, TZ_OFFSET, TO_TIMESTAMP_TZ, FROM_TZ, NUMTOYMINTERVAL, și NUMTODSINTERVAL.
  - Aceste funcții aparțin pachetului STANDARD și nu există nici o modalitate de a restricționa accesul la ele.
- **Funcții oferite de ORACLE**
  - Oracle oferă sute de funcții în pachetele bazei de date standard. Majoritatea acestor pachete sunt prefixate de DBMS_ și UTL_. Dacă un anumit pachet nu este folosit de aplicație, accesul la el ar trebui limitat.

Exemple reale - SQL Injection

„Lumea e atât de tare ocupată cu aparența,
încât prea puțin îi pasă de realitate.”
Oxenstierna

Exemple reale - SQL Injection

- **1 noiembrie 2005** : un elev de liceu a folosit o inserție SQL pentru a pătrunde în site-ul unei reviste de securitate taiwaneze și a fura informațiile clienților.
- **13 ianuare 2006** : un grup de infractori ruși au pătruns într-un site al guvernului Rhode Island, și se presupune că a furat datele cărților de credit de la persoane care au făcut afaceri online cu agențiile de stat.
- **2 martie 2007** : Sebastian Bauer a descoperit un defect de inserție SQL în pagina de login knorr.de.
- **29 iunie 2007** : un infractor a neutralizat site-ul Microsoft din Marea Britanie folosind SQL Injection. Un purtător de cuvânt al Microsoft a recunoscut problema site-ului *The Register* din U.K..

Exemple reale - SQL Injection

- **ianuarie 2008** : zeci de mii de PC-uri au fost infectate de un atac *SQL Injection* automat, care a exploatat o vulnerabilitate în codul aplicațiilor ce utilizează Microsoft SQL Server pentru stocarea bazei de date.
- **17 august 2009** : Departamentul de Justiție al Statelor Unite au acuzat un cetățean american, Albert Gonzalez și doi ruși anonimi de furtul a 130 milioane de numere de cărți de credit folosind un atac *SQL Injection*. În relatările presei apare că "cel mai mare caz de furt de identitate din istoria Statelor Unite", omul a furat carduri de la un număr de victime corporative după cercetarea sistemelor de prelucrare a plății. Printre companiile lovite se numără: procesorul de sisteme de plată *Heartland Payment Systems*, lanțul de magazine economice *7-Eleven* și lanțul de supermarketuri *Hannaford Brothers*.
- **decembrie 2009** : un atacator a pătruns într-o bază de date RockYou! prin utilizarea unui atac *SQL Injection,* care conținea în format text numele de utilizator și parolele necriptate pentru aproximativ 32 milioane de utilizatori.
- ...

Exemple reale - SQL Injection

- **25 octombrie 2018 :**
- The Wordfence Web Application Firewall has blocked 134 attacks over the last 10 minutes. Below is a sample of these recent attacks:

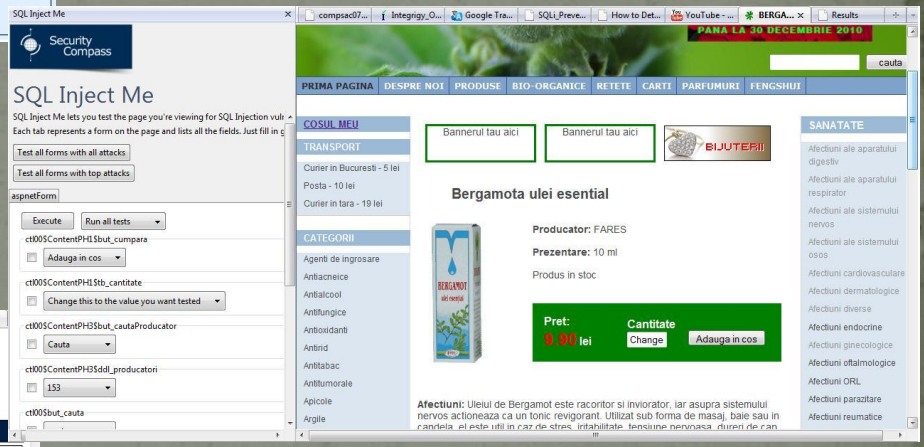
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL,NULL#
- octombrie 25, 2018 5:07pm 5.101.40.234 (Russian Federation) Blocked for SQL Injection in query string: full=1%' UNION ALL SELECT NULL#

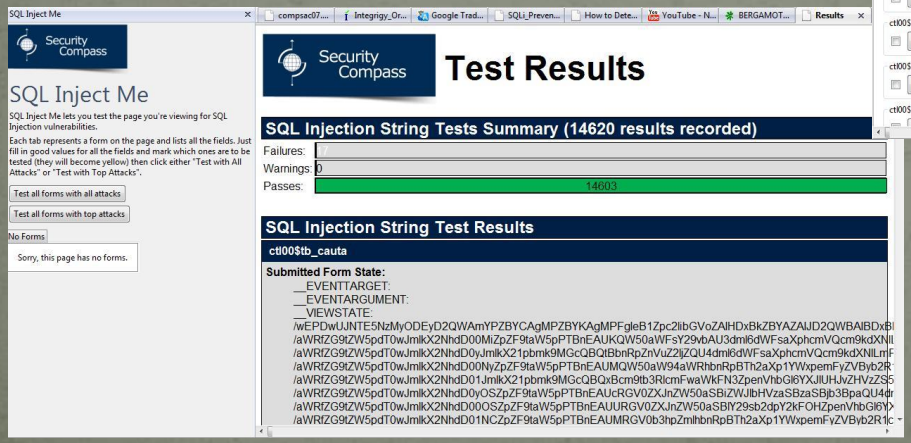
Aplicatii si tool-uri pentru detectarea SQL Injection

SQL Injection me (Firefox Add-on)

- In browser in meniul Tools -> SQL Inject Me -> Open SQL Inject Me Sidebar
- se selecteaza testele care urmeaza a fi rulate (Run all test, Run top 9 tests)
- de asemenea, exista posibilitatea de a testa toate formularele din site-ul web cu toate tipurile de atac sau toate formularele cu cele mai frecvente 9 tipuri de atac
- dupa executarea testelor se va deschide o noua pagina web care va contine un raport al executiei testelor


Add-ons for Firefox > Extensions > SQL Inject Me
SQL Inject Me 0.4.5
by Security Compass
Security Compass
SQL Injection vulnerabilites can cause a lot of damage to a web application. A malicious user can possibly view records, delete records, drop tables or gain access to your server. SQL Inject-Me is Firefox Extension used to test for SQL Injection vulnerabilities.
Continue to Download
Add to collection
Share this Add-on
Updated May 5, 2010
Website http://labs.securitycompass.com/index.php/exploit-me/
Works with Firefox 2.0.0.8 - 3.7a1pre
Rating 12 reviews
Downloads 162,223


SQL Inject Me
Security Compass
SQL Inject Me
Test all forms with all attacks
Test all forms with top attacks
Execute
Run all tests
PANA LA 30 DECEMBRIE 2010
PRIMA PAGINA
DESPRE NOI
PRODUSE
BIO-ORGANICE
RETETE
CARTI
PARFUMURI
FENGSHUI
COSUL MEU
TRANSPORT
CATEGORII
Bannerul tau aici
Bergamota ulei esential
Producator: FARES
Prezentare: 10 ml
Produs in stoc
Pret:
9.90 lei
Cantitate
Change
Adauga in cos
SANATATE


SQL Inject Me
Security Compass
SQL Inject Me
Test all forms with all attacks
Test all forms with top attacks
No Forms
Sorry, this page has no forms.
Test Results
SQL Injection String Tests Summary (14620 results recorded)
Failures: 0
Warnings: 0
Passes: 14603
SQL Injection String Test Results
ctl00$tb_cauta
Submitted Form State:
__EVENTTARGET:
__EVENTARGUMENT:
__VIEWSTATE:

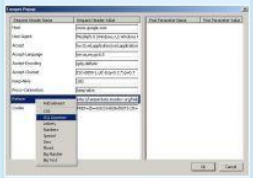
Tamper Data (Firefox Add-on)

- Pentru a utiliza acest add-on trebuie urmati pasii de mai jos:
  - se downloadeaza si se instaleaza add-on-ul Tamper Data
  - se porneste browserul Firefox
  - se selecteaza meniul Tools -> Tamper Data
  - se apasa Start Tamper
  - din browser se fac request-uri catre un site web
  - requesturile sunt inregistrate in Tamper Data – Ongoing requests (imaginea de mai jos)
  - intr-un textbox se introduce o valoare si se apasa butonul submit
  - o fereastra de tip pop-up este deschisa din Tamper Data, unde se specifica actiunea dorita: se trimite requestul mai departe catre server cu datale introduse de utilizator (Submit), se opreste cererea catre server (Abort request) sau se pot schimba parametrii cererii (Tamper)
  - daca se apasa butonul Tamper se pot vedea si schimba parametrii requestului respectiv; daca totul este in regula si nu a fost detectat un input malitios se apasa butonul Ok (imaginea de mai jos)

Add-ons for Firefox > Extensions > Tamper Data

Tamper Data 11.0.1

by Adam Judson



Use tamperdata to view and modify HTTP/HTTPS headers and post parameters...

Continue to Download ›

Updated	February 11, 2010
Website	http://tamperdata.mozde
Works with	Firefox 3.5 - 3.6.*
Rating	79 reviews
Downloads	3,432,430

Add to collection

Share this Add-on

Tamper Data - Ongoing requests

Start Tamper | Stop Tamper | Clear | Options | Help

Filter | Show All

Time	Duration	Total Duration	Size	Method	Status	Content Type	URL	Load Flags
17:17:03.5...	426 ms	426 ms	40702	GET	200	text/html	http://gr...	LOAD_DOCUM...
17:17:03.9...	0 ms	0 ms	unknown	GET	pending	unknown	http://gr...	LOAD_NORMAL
17:17:04.0...	0 ms	0 ms	unknown	GET	pending	unknown	http://gr...	LOAD_NORMAL
17:17:06.8...	n/a ms	n/a ms	unknown	GET	Cancelled	unknown	http://gr...	LOAD_NORMAL
17:17:07.7...	n/a ms	n/a ms	unknown	GET	Cancelled	unknown	http://gr...	LOAD_NORMAL
17:17:18.8...	n/a ms	n/a ms	unknown	GET	Cancelled	unknown	http://s....	LOAD_NORMAL

Request Header Name	Request Header Value

Response Header Name	Response Header Value

Tamper Popup

http://greenmedica.ro/search.aspx?cuvant=1+and+1%3d1

Request Header Name	Request Header Value
Host	greenmedica.ro
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 6
Accept	text/html,application/xhtml+xml,applic
Accept-Language	en-gb,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Referer	http://greenmedica.ro/search.aspx?cuv

Post Parameter Name	Post Parameter Value
__VIEWSTATE	%2FwEPDwUINDM3MjI0MjUPZ
__EVENTTARGET	
__EVENTARGUMENT	
__EVENTVALIDATION	%2FwEWuAEC%2B4Su6QkCkf
ctl00%24tb_cauta	1+and+1%3D1
ctl00%24but_cauta	cauta
ctl00%24ContentPH3%24ddl_producatori	153

OK | Cancel

AppScan – Application Vulnerability Scanner

- Parcurge o aplicatie web pentru a gasi punctele vulnerabile ale acesteia. Scannerul va cauta vulnerabilitati des intalnite, cum ar fi cross-site scripting, sql injection sau session hijacking.