Studiul atacurilor de tip SQL injection asupra unei baze de date

Daniel Gabriel CHIRUTA, Adina Luminița BĂRÎLĂ, Mirela DANUBIANU, Radu ȚIGĂNESCUL AMARIȚII, Sebastian COȚOVANU

***Abstract*—Securing the database may be the biggest action an organization, institution, government or simply a person can take in proactively defending itself against the myriad of intruders.**

Attacks on database can cause serious damage such as social impact or losses of incomes.

This paper presents a security perspective Database, by discussing the SQL Injection problem and its position in a compromised database and makes proposals to improve security against these attacks.

***Index Terms*—database, SQL, SQL injection, vulnerability**

I. INTRODUCERE

Încă din primele zile ale rețelelor de calculatoare bazele de date au avut întotdeauna un rol important .Rolul acestora este de a stoca, gestiona, şi de a oferi acces la cunoştințele colective ale oricărei organizații. Tranzacțiile financiare, extrase de cont, dosarele medicale, informațiile cărții de credit, polițele de asigurare, secrete comerciale, etc sunt unele din numeroasele lucruri care sunt memorate în orice baza de date.

Înainte de apariția internetului, bazele de date au fost ca nişte insule cu nici o conexiune sau foarte slabă între ele. Singurii oameni capabili să acceseze bazelor de date erau angajații calificați. Nivelul de amenințare la care aceste sisteme erau expuse era foarte redus.

În ultimile trei decenii numărul de baze de date computerizate a fost într-o rapidă şi continuă ascensiune. Apariția WEB-ului precum şi a capabilităților rețelelor au făcut ca accesul la date şi informații să fie mult mai usor.Un efect al acestei ascensiuni a fost acela că utilizatorii pot accesa acum cantități din ce în ce mai mari de informații într-un scurt timp .

Datorită faptului că tot mai multe instrumente şi tehnologii sunt în curs de dezvoltare pentru a, accesa şi utiliza datele, există, de asemenea o urgentă nevoie de a proteja aceste date.

Conform organizației Privacy Rights Clearinghouse[13] care menține o cronologie a încălcărilor datelor electronice, din februarie 2005 până în noiembrie 2010 au existat mai mult de 51 milioane de conturi personale compromise ca urmare a securității slabe; iar acestea sunt doar cazurile care au devenit publice. Informațiile furate includeau: dosare medicale, carduri de credit, informații personale, etc. Un clasament al acestor tipuri de informatii este prezentat in Fig 1.

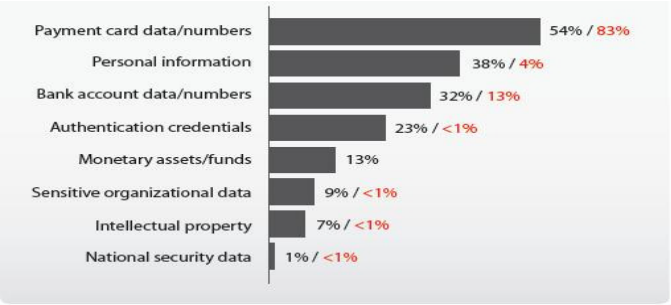

Fig. 1. Tipuri de date compromise (preluat Verizon Business RISK Team[17]).

II. SECURITATE VS HACKING

Aşa cum am mentionat mai devreme, securitatea în bazele de date este critică şi absolut esențială. Cu toate acestea, este de asemenea extrem de provocatoare. Securitatea în lumea digitala este înrudită cu securitatea în lumea reală. Ultimii ani ne-ar sugera, ca protejarea infrastructurii fizice s-a transformat într-un coşmar pentru specialiştii în securitate. Motivul este că adversarii rău intenționați pot locui oriunde şi totul este o posibilă țintă.

Acțiunile din categoria hacking cuprind toate încercările de a accesa sau de a deteriora în mod intenționat diferite informații fară a fi autorizat de mecanismele de securitate.

Hacking-ul oferă utilizatorilor mai multe avantaje decât alte categorii; acesta poate fi realizat de la distanță şi anonim, el nu are nevoie de interacțiune directă sau proximitate fizică, şi există mai multe instrumente disponibile pentru a automatiza şi a accelera atacurile.

Comform Verizon Business RISK Team[17] care au făcut un studiu asupra tipurilor de atacuri asupra bazelor de date pe locul al treilea se află SQL Injection .Acest lucru ne arată că ¼ din totalul de încălcări şi 89% din inregistrări se face cu ajutorul modificărilor asupra limbajului SQL.[Fig 2]

III. STRUCTURA SQL

Structura unei instrucțiuni SQL(Structured Query Language) este formată din cuvinte rezervate şi cuvinte definite de utilizator.

- *Cuvintele rezervate* : au un înteles fix, trebuie scrise exact cum este necesar şi nu pot fi împărțite în mai multe rânduri.(Ex: SELECT , INSERT , DELETE , UPDATE, etc.)
- *Cuvintele definite de utilizator* : sunt formate de către acesta, conform unor anumite reguli de sintaxă şi reprezintă denumirile diverselor obiecte din baza de date, cum ar fi relațiile, coloanele , vederile, etc.

Daniel Gabriel CHIRUȚĂ – Universitatea "Ştefan Cel Mare" Suceava, str.Universității, nr.13, RO-720229, Suceava (e-mail: danchiruta@stud.usv.ro)

Mirela DANUBIANU - Universitatea "Ştefan Cel Mare" Suceava, str.Universității, nr.13, RO-720229, Suceava (e-mail: mdanub@eed.usv.ro)

Radu ȚIGĂNESCUL AMARIȚII

Sebastian COȚOVANU

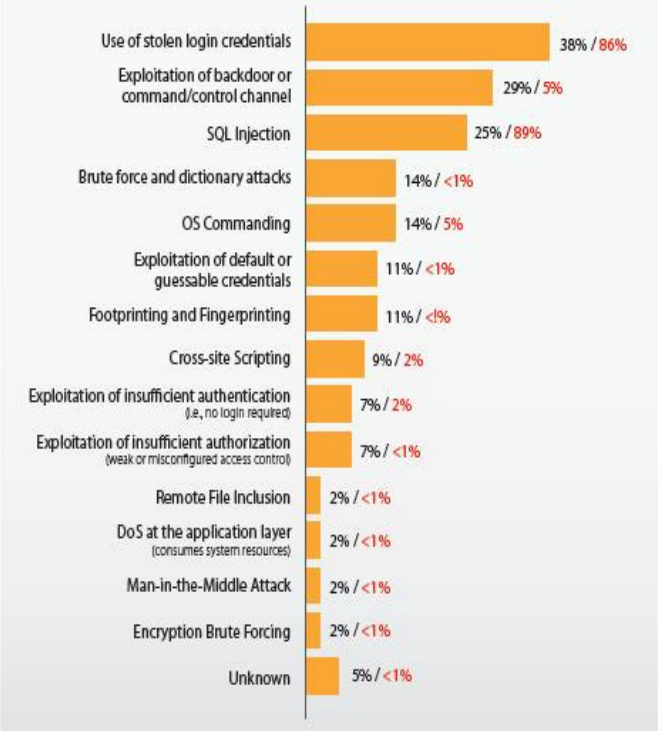

Fig. 2. Tipuri de atacuri (preluat Verizon Business RISK Team[17]).

Una dintre cele mai importante operații în SQL folosite pentru manipularea datelor este consultarea.Acest lucru se realizează cu fraza SELECT.

Forma generală a unei instrucțiuni SELECT este următoare:

```
SELECT [ALL | DISTINCT] [TOP nExpr [PERCENT]]
  [Alias.] Select_Item [Column_Name]
[, [Alias.] Select_Item [Column_Name] ...]
FROM [FORCE]
[DatabaseName!]Table [ Local_Alias]
[[INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL
[OUTER] JOIN
DatabaseName!]Table [ Local_Alias]
[ON JoinCondition …]
[[INTO Destination]
| [TO FILE FileName [ADDITIVE] | TO PRINTER
[PROMPT]
| TO SCREEN]]
[PREFERENCE PreferenceName]
[NOCONSOLE]
[PLAIN]
[NOWAIT]
[WHERE JoinCondition [AND JoinCondition ...]
[AND | OR FilterCondition [AND | OR
FilterCondition ...]]]
[GROUP BY GroupColumn [, GroupColumn ...]]
[HAVING FilterCondition]
[UNION [ALL] SELECTCommand]
[ORDER BY Order_Item [ASC | DESC] [,
Order_Item [ASC | DESC] ...]]
```

În exemplu de mai sus putem observa că avem două cuvinte definite de utilizator respectiv *JoinCondition* şi *FilterCondition* care pot fi întrebuințate de utilizatori în scopuri diferite.

În mod normal scopul ar fi unul 'BUN' dar în multe cazuri întâlnim şi utilizatori care folosesc aceste proprietăți ale instrucțiunii SQL într-un scop benefic doar pentru ei sau pentru organizațiile cărora sunt afiliați.

Un astfel de exemplu este descris în continuare.

IV. SQL Injection

SQL injection este una dintre cele mai devastatoare vulnerabilități, deoarece poate avea un impact sever şi poate duce la pierderea confidențialității /integrității, autentificări rupte şi în unele cazuri preluare de comenzi. În cazul cel mai grav, ea poate duce la compromiterea completă a bazei de date.

La o simplă întrebare ce este cu adevarat SQL injection? răspunsul deşi este unul "complicat" el este paradoxal "simplu". SQL injection este o vulnerabilitate care rezultă atunci cand i se dă unui atacator capacitatea de a influența interogările SQL şi ceea ce trece spre baza de date.

În cazul în care datele furnizate de utilizator nu sunt corect curațate şi validate, un utilizator rău intenționat poate injecta date artizanale pentru a schimba semantica interogările sau a comanda, ceea ce poate conduce la consecințe nedorite.

SQL injection nu este o vulnerabilitate care afectează exclusiv aplicațiile WEB; orice cod care acceptă ca intrare declarațiile SQL poate fi vulnerabil.

V. Tipuri de atacuri –SQL Injection

Există patru categorii principale de atacuri SQL injection împotriva bazelor de date :

- Manipulare SQL,
- Injecție de cod,
- Injecție de funcții,
- Umplerea bufferului.

a) Manipularea SQL

Manipulare SQL implică modificarea declarației SQL prin intermediul operatorilor pe mulțimi (de exemplu, uniunea, intersectia, etc), sau modificarea clauzei WHERE pentru a returna un rezultat diferit. Multe atacuri SQL injection demonstrate sunt de acest tip.

Atacul cel mai bine cunoscut este de a modifica clauza WHERE în situația autentificării unui utilizator, astfel încât această clauza WHERE să fie întotdeauna adevarată.

Ex:

O aplicație poate verifica autentificarea unui utilizator prin executarea urmatoarei interogari şi verificarea dacă exista raspuns de la server.

```
SELECT * FROM tabel_utilizatori
       WHERE nume = 'admin'
              and
              parola = 'parola'
```

Atacatorul încearcă să manipuleze instrucțiunea SQL astfel incat sa execute urmatoarea sintaxa:

```
SELECT * FROM tabel_utilizatori
```

```
      WHERE nume = 'admin'
               and
              parola = 'parola' or 'a' = 'a'
```

Datorită prioritații operatorilor clauza WHERE devine adevarată şi atacatorul dobândeşte în acest fel accesul.

Operatorul UNIUN este de asemenea utilizat în atacuri SQL injection. Scopul este de a manipula o declarație SQL astfel încât să returneze şi coloane din alte tabele. Un formular care execută următoarea interogare pentru a returna o listă de carți disponibile este:

```
SELECT nume_carte FROM carti
                WHERE nume_carte  like'%BAZE%'
```

Atacatorul încearcă să manipuleze instrucțiunea SQL astfel incât să execute urmatorea sintaxă:

```
SELECT nume_carte FROM carti
      WHERE nume_carte  like '%BAZE%'
             UNION
                    SELECT nume FROM dba_users
                      WHERE nume like '%'
```

Lista returnată de server va include toate cărțile selectate, dar, de asemenea şi toți utilizatorii bazei de date.

b) Injectie de cod

Atacurile prin injecție de cod încearcă să adauge declarații suplimentare sau alte comenzi SQL la comanda SQL trimisă către server.

Acest tip de atac este frecvent utilizat împotriva aplicațiilor Microsoft SQL Server, dar unerori se întalnește şi asupra bazelor de date Oracle. Declarația EXECUTE din SQL Server este o țintă frecventă a atacurilor de tip SQL injection.

Oracle nu suporta mai multe declarații SQL trimise pe aceeaşi cerere către baza de date de aceea următorul atac nu va avea nici un rezultat asupra bazelor de date Oracle. Următoarea declarație va duce la o eroare:

```
SELECT * FROM tabel_utilizatori
      WHERE nume = 'alice'
             and
             parola = 'parola';
 DELETE FROM tabel_utilizatori
             WHERE nume = 'admin';
```

Cu toate acestea, unele limbaje de programare sau API-uri permit situațiile în care să fie executate mai multe instrucțiuni. PL/SQL şi aplicațiile Java pot executa dinamic mai multe blocuri PL/SQL, blocuri care sunt vulnerabile la injecție de cod.

Ex:

```
BEGIN ENCRYPT_PASSWORD ('alice', 'parola');
END;
```

Exemplul de mai sus de bloc PL/SQL execută o procedură care criptează şi salvează parola şi utilizatorul. Un atacator va încerca să manipuleze acest bloc PL/SQL astfel încât să execute urmatorul cod:

```
BEGIN ENCRYPT_PASSWORD ('alice', 'parola');
   DELETE FROM tabel_utilizatori
       WHERE upper(nume) = upper('admin');
END;
```

c) Injectie de funcții

Baza de date Oracle permite funcții sau funcții în pachete să fie executate ca parte a unei declarații SQL. Deşi Oracle deține peste 1.000 de funcții în aproximativ 175 de pachete de date standard, doar câteva dintre aceste funcții pot fi utilizate

într-un atac SQL injection. Orice funcție personalizată sau funcția într-un pachet personalizat poate fi, de asemenea, executată într-o declarație SQL.

Funcțiile executate ca parte a unei declarații SQL SELECT nu pot întotdeauna modifica baza de date, excepție făcând cazul în care funcția este marcat ca "TRANSACTION PRAGMA". Nici una dintre funcțiile standard Oracle nu sunt executate ca tranzacții autonome; doar funcțiile ca INSERT, UPDATE, DELETE sunt în măsură să modifice datele dintr-o baza de date.

De asemenea multe pachete Oracle sau pachete personalizate pot fi exploatate de un atacator. Aceste pachete personalizate pot include funcții pentru a modifica parolele sau alte operațiuni.

Problema legata de injectia de funcții este că orice generare dinamica de, declarații SQL este vulnerabilă. Chiar cele mai simple declarațiile SQL pot fi exploatate în mod eficient.

Următorul exemplu demonstrează, că şi cele mai simple declarații SQL pot fi vulnerabile. Problema este că toti dezvoltatori de aplicatii trebuie să folosească functii predefinite ale bazelor de date pentru a efectua diferite task-uri în loc de cod nativ.

In urmatorul caz nu există nici o funcție care să facă translația, deci programatorul este decis să folosească o declarație SQL:

```
SELECT TRANSLATE(
             'intrare',
             '02468ABCDEFG',
             '02468')
  FROM dual;
```

Această declarație SQL nu este vulnerabilă la alte tipuri de atacuri SQL injection, dar este uşor de manipulat printr-un atac de injectare de functie. Atacatorul încearcă să manipuleze declaratia SQL în urmatorul mod:

```
SELECT TRANSLATE(
''||UTL_HTTP.REQUEST('http://192.168.1.1/') || '',
'02468ABCDEFG',
'02468')
FROM dual;
```

Schimbând declarația SQL se va solicita o pagina de la un server WEB. Atacatorul ar putea manipula şirul de caractere şi URL-ul pentru a include alte funcții în scopul de a obține informații utile de pe serverul de baze de date. Având în vedere că serverul de baze de date este cel mai probabil în spatele unui firewall, ar putea fi de asemenea folosit pentru a

ataca şi alte servere din rețeaua internă.

Un alt exmplu ar fi o aplicație particularizată care are functia SET_UTIL în pachetul MYAPPADMIN.Dezvoltatorul marchează funcția ca “PRAGMA TRANSACTION”, astfel încât să poată fi executată în orice circumstanțe speciale pe care aplicația le-ar putea întâmpina. Din moment ce este marcată” PRAGMA TRANSACTION”, atacatorul poate scrie în baza de date chiar într-o declaratie de tip SELECT.

```
SELECT TRANSLATE(
'' || myappadmin.set_util('admin', 'parola') || '',
'02468ABCDEFG',
'02468')
FROM dual;
```

Executând declarația SQL de mai sus, atacatorul este capabil sa creeze noi utilizatori.

d) Umplerea bufferului.

Umplerea bufferului a fost indentificată printre funcțiile standard ale mai multor baze de date. Cateva funcții ale bazei de date Oracle sunt suscetibile la umplerea bufferului, de aceea ele pot fi exloatate în scopul de a permite un atac SQL injection asupra bazei de date.

Unele dintre aceste functii :

- tz_offset,
- to_timestamp_tz,
- bfilename.

Cele mai multe servere de aplicații web nu se ocupă grațios de pierderea unei conexiuni la baza de date din cauza unei umplere a bufferului de memorie. De obicei în timpul unei cereri web conexiunea la client va fi menținută până în momentul rezilierii; ceea ce face un bun moment de atac.

În continuare pentru a putea executa instrucțiiunile SQL se folosesc tehnicile descrise mai sus.

VI. Protecții împotriva atacurilor SQL Injection

Există mai multe tehnici pentru a preveni atacurile SQL injection.

Validarea intrărilor

Una dintre aceste tehnici este validarea intrările astfel încât să conțină numai date alfanumerice(fara spații, semne de punctuatie, etc).

În mod similar, trebuie verificat dacă valoarea introdusă se încadrează între un minim şi un maxim definit pentru respectivul câmp. Multe platforme oferă o facilitate cunoscută sub numele de “SQL preparat”. Acestă facilitate utilizează ca tehnică împotriva atacurilor SQL injection verificări de tip precompilate. Din păcate nu există nici un astfel de produs comercial. Aceeasta înseamnă că noi trebuie să creiem manual facilități pentru a fi utilizate în analiza şi execuția comenzilor SQL.

SQL injection este doar un exemplu dintr-o clasa largă de defecte software care pot compromite securitatea unei aplicații.

Variabile locale

Cea mai puternică protecție împotriva atacurilor SQL injection este utilizarea de variabile locale. Utilizând variabile locale se va îmbunătăți de asemenea şi perfomanța aplicațiilor. Aplicațiile de codificare standard ar trebui să solicite utilizarea de variabile locale în toate declaratiile SQL. Nici o declarație SQL nu ar trebui sa fie creată prin concatenarea împreună a şirurilor de caractere şi a parametrilor.

Variabilile locale ar trebui utilizate pentru fiecare declarație SQL indiferent de momentul sau locul unde declarația SQL este executată. Un complex atac SQL injection ar putea exploata o aplicație prin stocarea unui şir de caractere în baza de date, şir care ulterior să fie executat de o instrucțiune SQL dinamică.

Funcții de securitate

Atât funcțiile standard cât şi cele definite pot fi exploatate în atacuri SQL injection. Multe dintre aceste funcții pot fi utilizate în mod eficient într-un atac. De exemplu Oracle are sute de funcții standard şi implicit toate sunt setate cu dreptul de a fi accesate de orice utilizator(PUBLIC). În general aplicațiile au funcții suplimentare care efectuează operații precum schimbarea de parole sau crearea de utilizatori care ar putea fi exploatate în mod negativ.

Toate funcțiile care nu sunt absolut necesare pentru o aplicație ar trebui restricționate.

Mesaje de eroare

Dacă un atacator nu poate obține codul sursă pentru o aplicație, mesajele de eroare pot deveni extrem de importante pentru un atac de succes. Cele mai multe aplicații Java nu returneze mesaje de eroare detaliate.

Testarea şi analiza pentru o aplicație trebuie să fie efectuate foarte amănunțit pentru a observa dacă aplicația întoarce mesaje de eroare detaliate sau nu.

VII. Concluzii

Lucrarea de față este o sinteză a problematicii pe care le ridică atacurilor SQL injection asupra bazelor de date. Acestea sunt de departe cea mai gravă amenințare pentru operatorii de baze de date.

Pentru a ataca cu succes o bază de date, atacatorul trebuie să fie capabil să modifice codul SQL într-o manieră potrivită. Această sarcină nu este una simplă pentru că acest lucru necesită cunoştințe temeinice ale codului, chestiunilor tehnice specifice bazelor de date şi ale implementării particulare a limbajului SQL.

Pe parcursul acestui articol se analizează modul în care sunt afectate bazele de date, de atacurile SQL injection, efectele pe care le are un astfel de atac si tehnicile folosite pentru protejarea bazele de date împotriva acestor atacuri.

Deşi limbajul SQL are o vârstă, subiectul este şi va rămâne de actualitate pentru că încă nu se ştie dacă noile tehnologii vor asigura securitate ‘PERFECTĂ’.

REFERINȚE

[1] C. Anley, Advanced SQL Injection in SQL Server Applications, www.ngssoftware.com/papers/advanced_sql_injection.pdf

[2] A. Belapurkar, A. Chakrabarti, H. Ponnapalli(2009), Distributed System Security/Issues, Processes and Solutions, ISBN 978-0-470-51988-2

[3] M. Bravenboer , E. Dolstra, E. Visser, Preventing injection attacks with syntax embeddings , http://www.sciencedirect.com/

[4] J. Clarke(2009), SQL Injection Attacks and Defense, ISBN 13: 978-1-59749-424-3

[5] J. B. Ullrich,Defacing websites via SQL injection http://www.sciencedirect.com/

[6] D.Litchfield(2005),The Database Hacker's Handbook: Defending Database Servers, ISBN:0764578014

[7] U. Mattsson, Defending the database, http://www.sciencedirect.com/

[8] D. Morgan, Web application security – SQL injection attacks, http://www.sciencedirect.com/

[9] R. B. Natan(2005) Implementing Database Security and Auditing, ISBN: 1-55558-334-2

[10] NIST, National Vulnerability Database, http://nvd.nist.gov/

[11] Oracle Magazine Jul-Aug(2009), Developing Secure Applications http://www.oracle.com/technetwork/oramag/magazine/home/index.html

[12] Oracle , How to write SQL injection proof PL/SQL http://www.oracle.com/technetwork/database/features/plsql/overview/how-to-write-injection-proof-plsql-1-129572.pdf

[13] Privacy Rights Clearinghouse, http://www.privacyrights.org/

[14] S. K. Rahimi, F. S. Haug (2010), Distributed Database Management Systems/ A Practical Approach, ISBN 978-0-470-40745-5

[15] S.Thomas , L. Williams, T. Xie, On automated prepared statement generation to remove SQL injection vulnerabilities. http://www.sciencedirect.com/

[16] B.Thuraisingham(2005),Database and Applications Security Integrating Information Security and Data Management, ISBN 978-0-8493-2224-2

[17] Verizon- 2010 Data Breach Investigations Report http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf.

[18] Z.Zhang, Q. Zheng, X. Guan, Q. Wang, T. Wang, A method for detecting code security vulnerability based on variables tracking with validated-tree http://www.springerlink.com/