Secure ABAP Custom Code Project

How-To Guide

Potential injection of harmful SQL statements of clauses in execution of DML statements in ADBC

(Message number 1130)

CS	STATUS
internal	to be approved
DATE	VERSION
14.11.2014	0.1

Allocator	Name	Company
	Angel Pérez	Accenture

Date	Name	Alteration Reason	Version
	Angel Pérez	Creation	0.1

Table of Content

1 Problem

1.1 Description

Security problems can occur wherever external data (such as user input) is processed further without being checked.

In the present case, external data is injected into an SQL statement that is passed for execution by the ADBC API of the database. This could enable potential attackers to gain unauthorized access to the SAP database of the system by making unexpected input.

Data Manipulation Language (DML) statements used for managing data within schema objects (incomplete list):

- CALL - call a PL/SQL or Java subprogram
- DELETE - deletes all records from a table, the space for the records remain
- EXPLAIN PLAN - explain access path to data
- INSERT - insert data into a table
- LOCK TABLE - control concurrency
- MERGE - UPSERT operation (insert or update)
- SELECT - retrieve data from the a database
- UPDATE - updates existing data within a table
- ...

Unlike using Open SQL with dynamic clauses, ABAP Database Connectivity (ADBC) is used to specify complete SQL statements. This increases the level of risk. All classes of the ADBC begin with prefix "CL_SQL_*" and "CX_SQL_*".

1.2 Example

Bad code

```
PARAMETER: p_car TYPE string.
DATA lv_stmt TYPE string.
DATA sql TYPE REF TO cl_sql_statement.

CREATE OBJECT sql.
. . .
CONCATENATE 'UPDATE ZT_CARS SET model = ''VW'' ' 'WHERE car EQ '' ' p_car '
''' INTO lv_stmt.

TRY.
CALL METHOD cl_sql_statement->execute_update
  EXPORTING
    statement      = lv_stmt
  RECEIVING
```

```
     rows_processed = l_rows
     .
 CATCH cx_sql_exception .
 CATCH cx_parameter_invalid .
*error
ENDTRY..
```

If parameter LV_CARR contains malicious SQL commands, the above SQL syntax can be modified.
If a malicious user enters ' OR '1'<>'2' the WHERE clause will be as follows: *car = '' OR '1'<>'2'*

This will always match, changing all entries in that table. As a result, this database statement can be used to modify database entries without authorization.

2 Solution

2.1 General Approach

First check whether it is necessary to use dynamic ADBC. Switching to static OPEN SQL provides a full solution to the security problem. If this is not possible, consider a switch to dynamic Open SQL. This reduces the opportunities for attacks considerably.

If it is absolutely essential that you use ADBC, make sure that no user input is entered directly into the SQL statement. Use ? placeholders for dynamic components of the statement. The relevant ABAP data objects can then be connected to these placeholders by using the method SET_PARAM of class CL_SQL_STATEMENT.

In exceptional cases, it may still be necessary to create the SQL statement based on user entries. These entries must be thoroughly checked beforehand.

The class CL_ABAP_DYN_PRG can be used to implement input checks Example

2.2 Good code

```
PARAMETER: p_car TYPE string.
DATA lv_stmt TYPE string.
DATA sql TYPE REF TO cl_sql_statement.

CREATE OBJECT sql.
. . .

* check (') quotes
lv_car = cl_abap_dyn_prg=>quote( p_car ).

CONCATENATE 'UPDATE ZT_CARS SET model = ''VW'' ' 'WHERE car EQ '' ' lv_car
'''' INTO lv_stmt.

TRY.
CALL METHOD cl_sql_statement->execute_update
  EXPORTING
    statement      = lv_stmt
  RECEIVING
    rows_processed = l_rows
    .
 CATCH cx_sql_exception .
 CATCH cx_parameter_invalid .
*error
ENDTRY.
```

If restric usage of quotes, then the user are not available to insert the malicious code in p_car like car = '' OR '1'<>'2'

2.3 Good code

```
PARAMETER: p_car TYPE zt_cars-car.
. . .

UPDATE ZT_CARS SET model = 'VW'
          WHERE car EQ p_car..
```

With this code not possible to insert conditions because the UPDATE statement will crash.