SQL Injection with ABAP
Ascending from Open SQL Injection to ADBC Injection
Andreas Wiegenstein

- CTO and founder of Virtual Forge, responsible for R&D
- SAP Security Researcher, active since 2003
- Co-Author of "Secure ABAP Programming" (SAP Press)

Virtual Forge GmbH

- SAP security product company based in Heidelberg, Germany
- Focus on (ABAP) application security services
  - ABAP Security Scanner
  - ABAP Security Guidelines
  - ABAP Security Trainings
  - SAP Security Consulting
Belief: "Our SAP system is secure."

- Roles & Authorizations
- Segregation of Duties
- Secure Configuration & System / Service Hardening
- Encryption
- Secure Network Infrastructure
- Password Policies
- Patch Management
- Identity Management
- Single Sign-on

SUPER-GREEN!
"...and this is our ABAP security department."
1. About ABAP

2. SQL Injection revisited

3. Open SQL (OSQL) Overview, Risks & Mitigations

4. Native SQL

5. ABAP Database Connectivity (ADBC)
1. …and then there was ABAP
- Proprietary language, exact specification not (freely) available
- Platform-independent code
- Client separation built-in *
- Integrated auditing capabilities
- System-to-System calls via SAP Remote Function Call (RFC)
- Client-Server communication via SAP GUI (DIAG protocol)
- Various programming paradigms:
  - Programs & Forms, Reports, Function Modules, Dynpros
  - Classes & Methods, Business Server Pages, Web Dynpro ABAP
- Integrated platform-independent SQL Standard: Open SQL
- Built-in authentication, roles and (explicit) authorization model
- Thousands of well-known standard programs and database tables
- 150+ Million Lines of Code in an ECC6.0 System
A closer look at Client Separation

- Users log on to "clients"
- Clients represent business (and user) data of independent organizations
- The SAP system implicitly separates client data in the database
  - Done via a special column that indicates, if a table is client-dependent
- ABAP code is client-*independent*. Every program is available on all clients
Attack Surface of ABAP

Diagram showing the attack surface of an ABAP system, with direct UIs including SAP GUI, BSP, Integrated ITS, and Web Dynpro ABAP. The SAP ABAP System includes DB, Files, ABAP Code, PI, RFC, and Web Services. Indirect UIs include 3rd party Java applications, J2EE/Portal, Web Dynpro Java, JCO, and External systems like Stand-alone ITS, SAP System, and Non-SAP System.
2. SQL Injection revisited
SQL Injection Overview

- **Special form of In-band Signalling**
  1) Data (input) is combined with commands (SQL syntax)
  2) Result (data + commands) is executed
  3) Commands embedded in data can corrupt the intended SQL commands

- **Typical test patterns**
  - ' or 1=1 --
  - ' or 1=1 /*

- **Countermeasure: Prepared Statements**

- **SQL Injections are known at least since 12/1998 (Phrack.org issue #54)**
SAP-specific SQL Injection Risks

- Illegal access to data in other clients
- Modification of user accounts and user authorizations (SOX violation)
  - E.g. Assign unauthorized user SAP_ALL privileges
- Undocumented changes to critical database tables (SOX violation)
  - No records in CDHDR, CDPOS, …
- Read access to HR data (Privacy issue)
  - E.g. social security number (PA0002-PERID)
- Access to credit card data (PCI/DSS violation)
  - E.g. BSEGC-CCNUM
- Access to bank accounts of customers and suppliers
  - E.g. customer bank account data (KNBK-BANKN)
- Manipulation of financial data (SOX violation)
  - E.g. tampering with BSEG
3. Open SQL (OSQL) Overview, Risks & Mitigations
Open SQL commands are integrated in the ABAP language
- SELECT, UPDATE, INSERT, DELETE, MODIFY
- OSQL commands are compiled together with the ABAP program

Most ABAP Code (>95%) uses Open SQL for DB queries

Open SQL automatically enforces SAP security features
- Only defined database commands can be executed
- Client separation
- Logging
REPORT  SQL_01.

DATA lt_sec TYPE sbook.

PARAMETERS p_carrid TYPE string.

SELECT class passname fldate
  FROM sbook
  CLIENT SPECIFIED
  INTO CORRESPONDING FIELDS OF lt_sec
  WHERE carrid  = p_carrid
    AND reserved = ' '.

  WRITE : / lt_sec-class, lt_sec-passname, lt_sec-fldate.
ENDSELECT.
REPORT SQL_02.

PARAMETERS p_carrid TYPE string.

DATA lt_sec TYPE sbook.
DATA lv_where TYPE string.

CONCATENATE `carrid = '' p_carrid '' AND reserved = ''` INTO lv_where.

SELECT class passname fldate
  FROM sbook
  CLIENT SPECIFIED
  INTO CORRESPONDING FIELDS OF lt_sec
  WHERE (lv_where).

  WRITE : / lt_sec-class, lt_sec-passname, lt_sec-fldate.
ENDSELECT.
REPORT SQL_03.

PARAMETERS p_table TYPE string.

DATA lt_sec TYPE sbook.
DATA lv_table TYPE string.

CONCATENATE `S` p_table INTO lv_table.

SELECT *
FROM (lv_table)
CLIENT SPECIFIED
INTO CORRESPONDING FIELDS OF lt_sec.

WRITE : /
  lt_sec-class, lt_sec-passname, lt_sec-fldate.
ENDSELECT.
DEMO
SAP Mitigation(s)

- SAP Note 1520356 - Avoiding SQL Injections
- ABAP countermeasures available since 12/2010
ABAP strings are *usually* enclosed in ` (back ticks)

```plaintext
DATA str TYPE string.
str = `Hello string`.
```

ABAP char arrays are *usually* enclosed in ' (single quotation marks)

```plaintext
DATA chr TYPE c LENGTH 80.
chr = 'Hello char'.
```

Hence ` as well as ' can be used in dynamic OSQL to enclose variables

```plaintext
CONCATENATE `carrid = 'p_carrid` AND reserved = ` ` INTO str.
CONCATENATE 'carrid = `p_carrid` AND reserved = ` ` INTO chr.
```

SAP countermeasures include two methods to escape quotes

```plaintext
cl_abap_dyn_prg=>escape_quotes_str(str)
  ` -> ` `

cl_abap_dyn_prg=>escape_quotes(chr)
  ' -> ``
```
SAP Mitigation(s) have Risks

- The method-names suggest usage for a given variable type
  
  - cl_abap_dyn_prg=>escape_quotes_str
  - -> to use for strings
  - cl_abap_dyn_prg=>escape_quotes
  - -> to use for non-strings (character arrays)

- Careful: It's not the variable-type that's relevant but the *type of quote* used!

- Risk: The method-names are misleading and may confuse developers

  ```
  DATA lv_where TYPE string.
  P_carrid = cl_abap_dyn_prg=>escape_quotes_str( p_carrid ).
  CONCATENATE `carrid = `' p_carrid `' AND reserved = ` ` INTO lv_where.
  ```

  **WRONG ESCAPING**
- **Avoid**
  
  cl_abap_dyn_prg=>escape_quotes_str
  cl_abap_dyn_prg=>escape_quotes

- **Use**
  
  cl_abap_dyn_prg=>quote_str
  cl_abap_dyn_prg=>quote

- **These functions not only escape the input, but also wrap it in the same quote character they escape**

  DATA lv_where TYPE string.
  P_carrid = cl_abap_dyn_prg=>quote_str( p_carrid ).
  CONCATENATE `carrid = ` p_carrid ` AND reserved = ' ' INTO lv_where.

- **Examples**

  cl_abap_dyn_prg=>quote_str( )  O`Neill -> `O``Neill`
  cl_abap_dyn_prg=>quote( )  O'Neil -> 'O''Neill'
4. Native SQL
"Native SQL" is SQL placed inside specific ABAP commands
- EXEC SQL … ENDEXEC.
- Used when database-specific commands have to be executed that are not part of Open SQL
- Native SQL is always hard-coded
- Input is passed to placeholders (as in prepared statements)
- Native SQL bypasses SAP security features of Open SQL
  - Client separation
  - Restrictive access to SQL commands
- Native SQL can't access certain SAP tables
  - Cluster Tables and Pool Tables don't physically exist in the DB

No SQL Injection possible, but should not be used anyway
REPORT SQL_04.

DATA: f1 TYPE s_class.
DATA: f2 TYPE s_passname.
DATA: f3 TYPE s_date.

PARAMETERS p_carrid TYPE string.

EXEC SQL.

   SELECT CLASS, PASSNAME, FLDATE INTO :F1, :F2, :F3 FROM SBOOK
   WHERE CARRID = :p_carrid AND RESERVED = ''

ENDEXEC.

WRITE: / f1, f2, f3.
5. ABAP Database Connectivity (ADBC)
ADBC Overview

- ADBC allows to dynamically execute *arbitrary* SQL commands
- ADBC is technically based on SAP kernel calls
  ```
  CALL 'C_DB_EXECUTE' ...
  CALL 'C_DB_FUNCTION' ...
  ```
- ADBC is provided in ABAP classes CL_SQL_* and a function module
  ```
  CL_SQL_STATEMENT
  CL_SQL_PREPARED_STATEMENT
  DB_EXECUTE_SQL (Function Module)
  ```
- ADBC bypasses SAP security features provided by Open SQL
  - Client separation
  - Restrictive access to SQL commands
  - Precompiled SQL statements
- Like Native SQL, ADBC can't access certain SAP tables
REPORT SQL_05.

DATA: lv_len TYPE i.
DATA: lv_sqlerr TYPE i.

PARAMETERS lv_stmt TYPE c LENGTH 80.

lv_len = STRLEN( lv_stmt ).

CALL 'C_DB_EXECUTE' ID 'STATLEN' FIELD lv_len
  ID 'STATTXT' FIELD lv_stmt
  ID 'SQLERR' FIELD lv_sqlerr.

- Executes an *arbitrary* SQL command (except SELECT)
- Used in function module DB_EXECUTE_SQL

ADBC INJECTION
REPORT SQL_06.

PARAMETERS lv_stmt TYPE c LENGTH 80.

CALL 'C_DB_FUNCTION' ID 'FUNCTION' FIELD 'DB_SQL'
   ID 'FCODE'   FIELD 'PO'
   ID 'STMSTR'  FIELD lv_stmt
   ...

- Executes an *arbitrary* SQL command
- Used in class CL_SQL_STATEMENT
DEMO
SAP Note 1456569 – "Potential modification of persisted data"
(https://service.sap.com/sap/support/notes/1456569)

Virtual Forge Security Advisory SAP-NSI-01
Despite common belief, OSQL Injections are possible in ABAP.

Despite common belief, arbitrary SQL statements can be executed on SAP systems, using ADBC.

The criticality of an OSQL Injection depends on the affected table and whether it is read or write access.

A single ADBC Injection means complete compromise of the SAP system.
Organizations

BIZEC – Business Security Initiative
http://www.bizec.org

Literature

"Secure ABAP-Programming"
(Learn German first ;-)  
SAP Press 2009

If you find new zero days
secure@sap.com
Questions?

For the most current version of this document, visit http://www.VIRTUALFORGE.com/

Andreas(dot)Wiegenstein(at)virtualforge(dot)com

VirtualForge GmbH
Speyerer Straße 6
69115 Heidelberg
Deutschland

Phone:  + 49 (0) 6221 86 89 0 - 0
Fax:     + 49 (0) 6221 86 89 0 - 101
Disclaimer

SAP, R/3, ABAP, SAP GUI, SAP NetWeaver and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only.

The author assumes no responsibility for errors or omissions in this document. The author does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

The author shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of this document.

No part of this document may be reproduced without the prior written permission of Virtual Forge GmbH.

© 2011 Virtual Forge GmbH.