Bundesamt
für Sicherheit in der
Informationstechnik

Umsetzungshinweise zum Baustein APP.4.6. SAP ABAP-Programmierung

1. Einleitung

Häufig werden in SAP-Systemen Eigenentwicklungen programmiert. Die Gründe hierfür sind vielfältig, so können Geschäftsprozesse oder Anforderungen an das Reporting mithilfe von Eigenentwicklungen individuell auf die Institution angepasst oder spezielle Funktionen erstellt werden, die in der Standard-Auslieferung nicht vorhanden sind.

Eigenentwicklungen werden von Entwicklern der Institution oder von beauftragten Entwicklern programmiert. Im SAP-Umfeld wird dazu häufig ABAP (Advanced Business Application Programming) verwendet. Dabei handelt es sich um eine proprietäre, plattformunabhängige Programmiersprache der Firma SAP. Sie wurde für die Programmierung kommerzieller Anwendungen im SAP-Umfeld entwickelt und ähnelt in ihrer Grundstruktur entfernt der Sprache COBOL. Wichtige Merkmale sind:

- Integration eines Authentisierungs-, Rollen- und Berechtigungskonzepts,
- Verwendung eines proprietären, datenbankunabhängigen SQL-Derivats (Open SQL),
- Unterstützung der Kommunikation zwischen verschiedenen SAP-Systemen sowie
- Integration von Auditoptionen.

2. Maßnahmen

Im Folgenden sind spezifische Maßnahmen für die Anforderungen des Bausteins APP.4.6. *SAP ABAP-Programmierung* aufgeführt.

Alle Maßnahmen (gekennzeichnet mit M) sind aufsteigend nummeriert und korrespondieren mit den entsprechenden Anforderungen (gekennzeichnet mit A).

2.1. Maßnahmen zum Baustein APP.4.6. SAP ABAP-Programmierung

APP.4.6.M1 Absicherung von Reports mit Berechtigungsprüfungen (B)

Es gibt viele Möglichkeiten, Reports mit SAP-Standardmitteln oder selbst geschriebenen Programmen zu starten. Hierzu zählen neben den bekannten Transaktionen SE38 und SA38 auch

- Transaktion SE80,
- Transaktion START_REPORT und
- kundeneigene Module.

Ebenso können mit dem Kommando SUBMIT REPORT aus eigenen ABAP-Programmen weitere ABAP-Programme aufgerufen werden.

Nicht bei allen möglichen Startmethoden findet jedoch eine implizite Berechtigungsprüfung seitens SAP statt. Um sich nicht darauf verlassen zu müssen, dass jeder Aufrufer eines Programms zuvor eine passende Berechtigungsprüfung durchführt, sollte jedes Programm selbst eine explizite, zum Kontext des Programms passende Berechtigungsprüfung durchführen.

APP.4.6.M2 Formal korrekte Auswertung von Berechtigungsprüfungen (B)

Eine Berechtigungsprüfung schützt in einem ABAP-Programm nur dann vor unberechtigtem Zugriff auf Daten, wenn das Ergebnis einer Berechtigungsanfrage auch ausgewertet wird und dieses Ergebnis den weiteren Programmablauf beeinflusst. Deshalb muss nach jeder Berechtigungsprüfung durch das Kommando AUTHORITY-CHECK der Erfolg der Anfrage über die Systemvariable SY-SUBRC abgefragt werden.

Beispiel: Vor der Anzeige von Änderungsbelegen soll geprüft werden, ob der aktuelle Benutzer das Recht hat, diese Belege zu lesen:

```
AUTHORITY-CHECK OBJECT 'S_SCD0'
ID 'ACTVT' FIELD '08'.
IF SY-SUBRC NE 0.
“Nachricht wegen fehlender Berechtigung und Programmende
ELSE.
“Programmlogik zur Anzeige der Änderungsbelege
ENDIF.
```

APP.4.6.M3 Berechtigungsprüfung vor dem Start einer Transaktion (B)

ABAP-Programme können SAP-Transaktionen starten, indem sie die Befehle CALL TRANSACTION oder LEAVE TO TRANSACTION aufrufen. Während LEAVE TO TRANSACTION eine implizite Berechtigungsprüfung für den aktuell angemeldeten Benutzer durchführt, erfolgt dies bei CALL TRANSACTION in der Regel nicht (siehe unten, Parameter auth/check/calltransaction). Daher sollten Entwickler immer eine explizite Startberechtigungsprüfung erzwingen, wenn sie den Befehl CALL TRANSACTION verwenden. Ohne diese Prüfung könnten unberechtigte Benutzer an kritische Daten gelangen.

Zwar kann für CALL TRANSACTION über den Profilparameter auth/check/calltransaction auch eine implizite Berechtigungsprüfung erzwungen werden, jedoch sollte der Schutz von Transaktionsaufrufen generell nicht von Konfigurationseinstellungen eines Systems abhängig sein. Außerdem kann die hierdurch erzwungene implizite Prüfung auf das Berechtigungsobjekt S_TCODE in einigen Fällen unzureichend oder aber auch zu einschränkend sein.

Eine unzureichende Prüfung kann dann vorliegen, wenn eine Transaktion mit zusätzlichen Parametern aufgerufen wird. So kann z. B. das erste Bild übersprungen werden und die Transaktion wird im daraufhin eingeblendeten Bild fortgesetzt, wie es durch den aufgerufenen ABAP-Code definiert wird. Je nach Transaktion kann dieses Verhalten eventuell zusätzliche, detailliertere Berechtigungsprüfungen erfordern.

Es existieren jedoch auch Fälle, in denen eine Berechtigungsprüfung auf die gesamte Transaktion mittels S_TCODE kontraproduktiv wäre. Wenn durch das ausgewählte Programm (mittels MODE oder OPTIONS FROM) nur besondere Unterfunktionen einer Transaktion im Hintergrund aufgerufen werden, sollte es nicht erforderlich sein, dem Benutzer eine Genehmigung zum Starten der gesamten Transaktion zu gewähren.

Hinweis: Seit der SAP Netweaver Version 7.40 wird CALL TRANSACTION als obsolet betrachtet, wenn CALL TRANSACTION die neuen Optionen WITH AUTHORITY-CHECK / WITHOUT AUTHORITY-CHECK nicht verwendet. Daher sollten ab dann CALL TRANSACTION-bezogene Berechtigungsprüfungen mittels der Option WITH AUTHORITY-CHECK ausgeführt werden. Diese prüft die S_TCODE-Berechtigung, berücksichtigt aber auch Ausnahmen von Transaktionspaaren, die über Transaktion SE97 konfiguriert werden können. Besitzt der Benutzer unzureichende Zugriffsrechte, wird eine Ausnahme vom Typ cx_sy_authorization_error ausgelöst.

APP.4.6.M4 Verzicht auf proprietäre Berechtigungsprüfungen (B)

Das Berechtigungswesen in SAP ist ein mächtiges und komplexes Werkzeug. Um es korrekt in ABAP-Programme zu integrieren, muss entweder ein geeignetes SAP-Standard-Berechtigungsobjekt gefunden oder ein neues Berechtigungsobjekt angelegt werden. Anschließend muss definiert werden, welche Ausprägungen die darauf basierenden Berechtigungen haben können, z. B. Anzeigen, Ändern, Einfügen und Löschen eines betriebswirtschaftlichen Objekts. Im nächsten Schritt sind entsprechende Rollen zu definieren oder bestehende Rollen zu ergänzen. Danach müssen die Rollen noch den berechtigten Benutzern zugewiesen werden.

Da in diese Prozesskette zahlreiche Abteilungen involviert sind (Entwicklung, Berechtigungs- und Rollenarchitekten, Berechtigungsadministration), werden in einigen Fällen proprietäre Berechtigungsprüfungen implementiert. Diese fragen oft nur einen bestimmten Benutzernamen (IF SY-UNAME = ‚MUELLER‘.) oder andere Eigenschaften des Benutzers (Abteilung, Organisationseinheit etc.) ab. Allerdings können solche Berechtigungsprüfungen auch dazu benutzt werden, das Standardberechtigungskonzept ganz bewusst zu umgehen, um unrechtmäßigen Zugriff auf Daten zu erhalten oder andere Angriffe durchzuführen. Werden proprietäre Prüfungen im Quelltext von Drittanbietern (Outsourcing, Add-on-Produkte von externen Anbietern) gefunden, ist besondere Vorsicht angebracht.

Deswegen muss jede Berechtigungsprüfung in ABAP-Programmen technisch ausschließlich durch den Befehl AUTHORITY-CHECK <OBJECT> erfolgen (der Befehl kann dabei auch indirekt verwendet werden, zum Beispiel durch den Aufruf von SAP-Modulen, die ihn verwenden).

Durch die Entwicklungsrichtlinie muss verboten sein, Geschäftslogik basierend auf dem Namen oder anderen Eigenschaften des Benutzers auszuführen.

APP.4.6.M5 Erstellung einer Richtlinie für die ABAP-Entwicklung (S)

Es sollte eine Richtlinie (Entwicklungsrichtlinie) für die ABAP-Entwicklung erstellt werden, die genau definierte Regeln beinhaltet, wie sicherheitsrelevante Schwachstellen zu vermeiden sind. Die Anforderungen aus dem Baustein APP.4.6 *SAP ABAP-Programmierung* sollten in die Richtlinie aufgenommen werden. Diese Regeln müssen stets zeitnah aktualisiert werden. Gründe hierfür sind z. B. die kontinuierliche Erweiterung des Sprachumfangs der Programmiersprache ABAP selbst oder Erweiterungen, die zusammen mit dem Kern des SAP-Systems (SAP Netweaver Stack) zur Verfügung gestellt werden.

APP.4.6.M6 Vollständige Ausführung von Berechtigungsprüfungen (S)

Berechtigungsprüfungen werden innerhalb von ABAP-Programmen durch den Befehl AUTHORITY-CHECK <OBJECT> realisiert. Dabei wird geprüft, ob der aktuell angemeldete Benutzer über die passenden Zugriffsrechte für das mit übergebene Berechtigungsobjekt OBJECT verfügt. Das geschieht, indem eine festgelegte Liste von Feldern für das Berechtigungsobjekt überprüft wird.

Die Anweisung AUTHORITY-CHECK sollte immer alle Felder des entsprechenden Berechtigungsobjekts auswerten. Wird ein Feld bei der Berechtigungsprüfung weggelassen, dann werden die Rechte des Benutzers in Bezug auf dieses Feld nicht geprüft. Damit ist die Prüfung unvollständig und kann zu unberechtigtem Zugang führen.

Dazu ein Beispiel:

```
AUTHORITY-CHECK OBJECT 'S_DEVELOP'
ID 'DEVCLASS' FIELD 'ZPROGS'
ID 'OBJTYPE' FIELD 'FUNC'
ID 'OBJNAME' FIELD 'ZFT'
ID 'ACTVT' FIELD '02'.
IF sy-subrc = 0.
"Benutzer für Aktion berechtigt
ENDIF.
```

Obwohl hier das Objekt S_DEVELOP auch das Berechtigungsfeld P_GROUP beinhaltet, fehlt es im ABAP-Code und wird folglich nicht geprüft. Hat der Entwickler es vergessen oder absichtlich weggelassen und falls ja, warum? Über die Gründe kann insgesamt nur spekuliert werden.

Wenn bei einer Berechtigungsprüfung ein Feld tatsächlich nicht erforderlich ist, sollte es als DUMMY deklariert und somit ausdrücklich übergangen werden. Der Beispielcode sieht dann wie folgt aus:

```
AUTHORITY-CHECK OBJECT 'S_DEVELOP'
ID 'DEVCLASS' FIELD 'ZPROGS'
ID 'OBJTYPE' FIELD 'FUNC'
ID 'OBJNAME' FIELD 'ZFT'
ID 'P_GROUP' DUMMY "Not required for type 'FUNC'
ID 'ACTVT' FIELD '02'.
IF sy-subrc = 0.
"Benutzer für Aktion berechtigt
ENDIF.
```

Dabei sollten folgende Regeln befolgt werden:

- Eine Prüfung auf alle Felder des Objekts mit DUMMY sollte verboten sein.
- Eine DUMMY-Prüfung auf Felder, die eine Aktivität beschreiben (z. B. ACTVT, AUTHC, CO_ACTION) sollte verboten sein.

Zusammenfassend gilt also: Bei einer Berechtigungsprüfung in ABAP-Programmen sollten alle Felder des relevanten Berechtigungsobjekts explizit einbezogen werden. Werden einzelne Felder tatsächlich nicht benötigt, so sollten diese aus Gründen der Transparenz und Nachvollziehbarkeit als DUMMY gekennzeichnet werden. Zusätzlich sollte am Feld der Grund für die Ausnahme genannt werden. Eine Prüfung aller Felder eines Berechtigungsobjekts auf DUMMY oder eines Feldes, das eine Aktivität beschreibt, bietet keinen hinreichenden Schutz.

APP.4.6.M7 Berechtigungsprüfung während der Eingabeverarbeitung (S)

In ABAP-Dynpro-Anwendungen werden Menüeinträge und Bildschirmelemente oft basierend auf den Berechtigungen des Benutzers ein- oder ausgeblendet. Dadurch wird zwar verhindert, dass die entsprechende Funktion in der Dynpro-Oberfläche selbst ausgelöst werden kann, jedoch ist es weiterhin möglich, den zugehörigen Funktionscode durch Eingabe in das Kommandofeld der SAP GUI auszulösen. Dadurch kann der Benutzer eine Funktion ausführen, für die er eigentlich nicht berechtigt ist.

Wenn bestimmte Einträge eines Dynpro-Menüs ausgeblendet oder einzelne Schaltflächen deaktiviert werden sollen, dann muss also die Behandlung der zugehörigen Funktionscodes ebenfalls im Quellcode verhindert werden.

Andersherum sollten auch keine Funktionscodes existieren, die keinem Dynpro-Element zugeordnet sind.

APP.4.6.M8 Schutz vor unberechtigten oder manipulierenden Zugriffen auf das Dateisystem (S)

Aus ABAP-Programmen wird häufig lesend oder schreibend auf Dateien des SAP-Servers zugegriffen. Dabei kann es sich um Daten der Anwendung, aber auch um Konfigurations- oder Log-Informationen handeln.

Erlaubt ein Programm, die zu bearbeitende Datei über eine Benutzereingabe frei zu wählen, so besteht potenziell die Gefahr, dass beliebige Dateien des Servers gelesen, verändert oder gelöscht werden. Somit können Daten nicht nur ausspioniert oder manipuliert werden, sondern sogar das gesamte System kompromittiert werden.

Muss ein Teil des Pfades oder des Dateinamens durch Benutzereingaben bestimmbar sein, dann sollte die Eingabe vor dem Dateizugriff validiert werden.

Der SAP-Hinweis 1497003 "Mögliche Directory Traversals in Anwendungen" (siehe [1497003]) beschreibt gleichfalls eine Validierungsfunktion (Funktionsbaustein FILE_VALIDATE_NAME), die genutzt werden sollte, falls keine Whitelist gepflegt werden kann. Über diesen Funktionsbaustein kann der konkrete Dateiname des jeweiligen Aufrufs gegen eine vorher definierte Konfiguration (Transaction FILE) geprüft werden.

APP.4.6.M9 Berechtigungsprüfung in remote-fähigen Funktionsbausteinen (S)

ABAP-Programme können über SAP-Systemgrenzen hinweg mittels remote-fähiger Funktionsbausteine kommunizieren. Dabei handelt es sich um Funktionsbausteine, die mit dem Attribut remote-fähig gekennzeichnet sind. Die hierzu bereitgestellte Remote-Function-(RFC)-Schnittstelle kann jedoch nicht nur von ABAP-Programmen genutzt werden, sondern auch von allen anderen Systemen, die mit dem jeweiligen SAP-System verbunden sind.

Voraussetzung für die Ausführung remote-fähiger Funktionsbausteine ist lediglich eine erfolgreiche Authentifizierung über das RFC-Protokoll.

SAP prüft dabei implizit über das Berechtigungsobjekt S_RFC die Aufrufberechtigung auf Basis einzelner Funktionsbausteine. Viele Institutionen scheuen jedoch die zusätzliche Komplexität einer Berechtigungsvergabe, die neben der Businesslogik auch noch architektonische Aspekte der Codemodularisierung berücksichtigen soll. Hinzu kommt, dass diese Granularität der Prüfung erst in aktuelleren SAP-Netweaver-Versionen vorhanden ist. Vorher wurde nur auf Basis ganzer Funktionsgruppen geprüft. Entsprechend sind viele RFC-Berechtigungen in Rollen auch noch auf Basis dieser Funktionsgruppen definiert. Aus diesen Gründen sind RFC-Berechtigungen oftmals sehr großzügig vergeben oder überhaupt nicht eingeschränkt.

Alle remote-fähigen Funktionsbausteine sollten deshalb selbst im Code explizit prüfen, ob der Aufrufer berechtigt ist, die zugehörige Businesslogik auszuführen. Die zu prüfenden Berechtigungen sollten hierzu zusammen mit den Fachverantwortlichen oder mit Sicherheitsexperten bestimmt werden.

APP.4.6.M10 Verhinderung der Ausführung von Betriebssystemkommandos (S)

Es existieren verschiedene Methoden, aus ABAP-Programmen Kommandos an das Betriebssystem zu senden. Zwar haben Administratoren die Möglichkeit, alle erlaubten Betriebssystemkommandos mit den Transaktionen SM49 und SM69 zu definieren, jedoch werden diese nur dann validiert, wenn der Aufruf des Betriebssystemkommandos über die dafür vorgesehenen SAP-Funktionsbausteine erfolgt.

Besonders kritisch sind Betriebssystemaufrufe, wenn Benutzereingaben als Teil des Kommandos verwendet werden (die definierten, erlaubten Kommandos also Parameter beinhalten) oder wenn der Entwickler andere Methoden als die Standardfunktionsbausteine benutzt.

In der Entwicklungsrichtlinie sollte deshalb festgelegt werden, dass Betriebssystemaufrufe ausschließlich über die SAP-Funktionsbausteine SXPG_CALL_SYSTEM oder SXPG_COMMAND_EXECUTE erfolgen. Sind

parametrisierte Kommandos in SM49 oder SM69 definiert, ist sicherzustellen, dass die Werte hierfür in ABAP-Programmen nicht von Benutzereingaben bereitgestellt werden. Sollten die Parameter der Kommandos dennoch ganz oder teilweise durch Benutzereingaben befüllt werden, so muss eine Eingabevalidierung vorgenommen werden.

Jedem Aufruf eines erlaubten Betriebssystemkommandos muss außerdem eine entsprechende Berechtigungsprüfung (Berechtigungsobjekt S_LOG_COM) vorangestellt werden.

APP.4.6.M11 Vermeidung von eingeschleustem Schadcode (S)

Mithilfe der ABAP-Befehle INSERT REPORT und GENERATE SUBROUTINE POOL können aus einem ABAP-Programm heraus weitere ABAP-Programme dynamisch generiert und nachfolgend auch gestartet werden. Werden Programme aber erst zur Laufzeit erzeugt, entziehen sie sich damit einer vorherigen Überprüfung (Codeanalyse). Das stellt ein sehr hohes Sicherheitsrisiko dar.

So ist es zum Beispiel ohne weiteres möglich, einen SAP-Standard-Report mittels READ REPORT in eine interne Tabelle einzulesen, die Sätze der Tabelle nachfolgend zu bearbeiten, um dann das so geänderte Coding wieder mit INSERT REPORT unter dem alten Programmnamen im System abzulegen. Diese Änderungen werden weder vom SAP-Änderungs- und -Transportwesen aufgezeichnet, noch wird das betroffene Programm vor der Änderung versioniert.

Noch gefährlicher ist es, wenn der Inhalt des dynamisch erzeugten Programms von Benutzereingaben abhängt. Werden solche Eingaben nicht sorgfältig im Code validiert, können Angreifer auf diesem Weg Schadcode in das System einschleusen.

Beispiel:

```
TYPES: line(255) TYPE c.

DATA: lt_tab TYPE STANDARD TABLE OF line.

DATA: lv_input TYPE string.

DATA: lv_code TYPE string.

lv_input = me->request→get_form_field('lv_input').

CONCATENATE 'WRITE "'lv_input"'.INTO lv_code.

APPEND 'report ZTF_REPORT.'TO lt_tab.

APPEND 'write: "Anzahl Kilometer: ".TO lt_tab.

APPEND lv_code TO lt_tab.

INSERT REPORT 'ZTF_REPORT' FROM lt_tab.

SUBMIT ('ZTF_REPORT').
```

Der Code generiert ein ABAP-Programm, in dem die Kilometeranzahl ausgegeben werden soll, die zuvor vom Benutzer eingegeben wurde. Gibt der Benutzer "100" ein, erscheint die Ausgabe "Anzahl Kilometer: 100". Die selbe Ausgabe erscheint, wenn der Benutzer

```
"100'. DELETE FROM MARA. DELETE FROM KNA1. WRITE '"
```

eingibt. Allerdings hat er so auch noch sämtliche Material- und Kundenstammdaten gelöscht.

Auch wenn das genannte Beispiel durch eine einfache Validierung der Eingabe auf numerische Zeichen behoben werden kann, so sollte aus Gründen der Transparenz und Testbarkeit auf dynamischen Code verzichtet werden.

Die Verwendung von Techniken, die ABAP-Quelltext zur Laufzeit erzeugen, sollte generell durch die Enwicklungsrichtlinie verboten sein. Sollten trotzdem dynamisch generierte Programme benötigt werden, so müssen diese Funktionen genehmigt und dokumentiert werden. Weiterhin muss darauf geachtet werden, dass möglichst keine direkten Benutzereingaben in den generierten Quellcode übertragen werden, sondern der Quellcode z. B. aus konstanten Textliteralen zusammengesetzt wird.

Wird der generierte Quellcode dennoch aus externen Parametern versorgt, so ist zwingend eine Eingabevalidierung vorzunehmen, z. B. mithilfe einer vordefinierten Liste erlaubter Werte oder einer Suche nach nicht erlaubten Zeichen.

APP.4.6.M12 Vermeidung von generischer Modulausführung (S)

Die Ausführung von ABAP-Modulen über die entsprechenden SAP-Standardtransaktionen ist durch eine implizite Berechtigungsprüfung geschützt.

Technisch gesehen werden ABAP-Module durch ABAP-Befehle aufgerufen. Diese Befehle erlauben es auch, den Namen des aufgerufenen Moduls dynamisch zur Laufzeit anzugeben. Ist dieser Name durch Benutzereingaben beeinflussbar, können hierüber potenziell beliebige Module des aufgerufenen Modultyps (Transaktionen, Reports, Methoden, Funktionsbausteine) ausgeführt werden.

Beispiel:

```
DATA: request TYPE REF TO if_http_request.
DATA: my_report TYPE string.
DATA: event TYPE string.
my_report = request->get_form_field( 'myreport' ).
TRY.
SUBMIT my_report.
RETURN.
ENDTRY.
```

Die generische Ausführung von ABAP-Modulen muss deshalb durch die Entwicklungsrichtlinie verboten sein.

Sollte es jedoch wichtige Gründe für eine generische Ausführung geben, muss detailliert dokumentiert werden, wo und warum das geschieht. Zusätzlich sollte dann eine Whitelist definiert werden, die alle erlaubten Module enthält. Bevor ein Modul aufgerufen wird, sollte die Benutzereingabe mit der Whitelist abgeglichen werden. Eine weitere benutzerbezogene Differenzierung kann durch eine passende vorangestellte Berechtigungsprüfung (AUTHORITY-CHECK) erfolgen.

APP.4.6.M13 Vermeidung von generischem Zugriff auf Tabelleninhalte (S)

Für den Zugriff auf Tabelleninhalte stellt der SAP-Standard verschiedene Transaktionen zur Verfügung, z. B. SM30, SE16 und SE16N. Diese Transaktionen sollten durch eine gezielte Berechtigungsvergabe vor unberechtigten Zugriffen geschützt werden.

Die SAP-Standardtransaktionen, mit denen sich Datenbanktabellen auslesen lassen, basieren auf dem Befehl SELECT. Er erlaubt es, den Namen der zu lesenden Tabelle dynamisch zur Laufzeit anzugeben. Wird ein solcher generischer SELECT-Befehl in Eigenentwicklungen benutzt und ist der Tabellenname durch Benutzereingaben beeinflussbar, kann hierüber potenziell jede dem SAP Data Dictionary bekannte Tabelle aus der Datenbank gelesen werden. Damit besteht das Risiko, dass durch Eigenentwicklungen die Absicherung von Transaktionen (z. B. SM30, SE16 und SE16N) ausgehebelt wird.

Beispiel:

```
DATA: lv_tab TYPE string.
DATA: lv_year TYPE string.
DATA: lv_ccvar TYPE string.
DATA: request TYPE REF TO IF_HTTP_REQUEST.
lv_tab = request->get_form_field( 'reference' ).
lv_year = '2010'.
```

```
SELECT ccdata FROM (lv_tab) INTO lv_ccvar WHERE lyear = lv_year.
```

Enthält der Parameter "reference" den Namen einer Tabelle mit einer Spaltenbezeichnung "ccdata" und einer Spaltenbezeichnung "lyear" können die selektierten Spalten ohne Berechtigung gelesen werden.

Das generische Auslesen von Tabelleninhalten sollte deshalb durch die Entwicklungsrichtlinie verboten sein.

Sollte es wichtige Gründe für einen generischen Zugriff geben, muss detailliert dokumentiert werden, wo und warum das geschieht. Außerdem sollte dann gewährleistet sein, dass sich der dynamische Tabellenname auf eine kontrollierbare Liste von Werten beschränkt. Eine dynamische OSQL-Abfrage muss in diesem Fall mindestens eine Präfix-Zeichenkette mit statischen Daten verwenden. Zusätzlich können die statischen Methoden CHECK_TABLE_NAME_STR und CHECK_TABLE_NAME_TAB der Klasse CL_ABAP_DYN_PRG benutzt werden, um zu prüfen, ob eine gültige (vorhandene) Tabelle übergeben wurde und ob diese Tabelle zu einer definierten Liste gehört.

Beispiel:

```
DATA: lv_tab TYPE string.

DATA: lv_year TYPE string.

DATA: lv_ccvar TYPE string.

DATA: request TYPE REF TO IF_HTTP_REQUEST.

lv_tab = request->get_form_field( 'reference' ).

lv_year = '2010'.

*Es sind nur Tabellen erlaubt, die mit 'ZFINT' beginnen.

CONCATENATE 'ZFINT' lv_tab INTO lv_tab.

SELECT ccdata FROM (lv_tab) INTO lv_ccvar WHERE lyear = lv_year.
```

APP.4.6.M14 Vermeidung von nativen SQL-Anweisungen (S)

Der Sprachumfang von ABAP umfasst mit OPEN SQL ein proprietäres SQL-Derivat, dessen Anweisungen portabel, also vom darunterliegenden Datenbankmanagementsystem unabhängig sind. Open SQL-Anweisungen arbeiten standardmäßig mit einer automatischen Mandantenbehandlung. Anweisungen, die auf mandantenabhängige Anwendungstabellen zugreifen, lesen und bearbeiten nur die Daten des aktuellen Mandanten.

Durch direkte Verwendung der ABAP-Anweisung EXEC SQL oder durch Verwendung der Klassen der ABAP Database Connectivity (ADBC) lassen sich zusätzlich auch native, datenbankspezifische SQL-Kommandos ausführen. Derartige spezielle Befehle gehen weit über den Sprachumfang von OPEN SQL hinaus und können schwerwiegende Sicherheitsprobleme hervorrufen, wenn zum Beispiel Befehle auf dem Betriebssystem ausgeführt werden.

Des Weiteren ist zu beachten, dass mit EXEC SQL ausgeführte SQL-Befehle nicht in den SAP-Prüfprotokollen erscheinen. Zudem führt EXEC SQL keine automatische Mandantenbeschränkung aus, wie es bei Open SQL der Fall ist (siehe auch APP.4.6.M20 Keine Zugriffe auf Daten eines anderen Mandanten). Aus diesem Grund stellt jeder Zugriff auf eine SAP-Datenbank mit einer mandantenabhängigen Tabelle unter Umständen einen Compliance-Verstoß dar.

Native SQL-Nutzung ist eventuell akzeptabel, wenn Daten in einer externen Datenbank verarbeitet werden, d. h. nicht in der SAP-Datenbank. Das primäre Sicherheitsrisiko von EXEC SQL liegt in der Umgehung des Open SQL Layer und darin, dass potenziell schädliche SQL-Anweisungen in der SAP-Datenbank ausgeführt werden. Werden die EXEC-SQL-Befehle an eine externe Datenbank gesendet, hängt das Risiko von den Schutzanforderungen dieser externen Datenbank ab. Falls die externe Datenbank keine kritischen Daten enthält, sind die durch EXEC SQL ausgeführten Befehle nicht gefährlich, und eine solche Verwendung kann als annehmbar betrachtet werden.

Die Verwendung von EXEC SQL oder ADBC sollte über die Entwicklungsrichtlinie verboten sein. Benutzereingaben sollten nicht Teil von ADBC-Befehlen sein.

APP.4.6.M15 Vermeidung von Datenlecks (S)

SAP-Systeme verwalten oft geschäftskritische Daten. Ein Risiko liegt insbesondere dann vor, wenn solche Daten unberechtigt

- angezeigt werden (z. B. SAP GUI, HTML, UI5, Smartforms, Adobe Forms),
- in Dateien gespeichert werden (sowohl server- als auch clientseitig),
- an andere Anwendungen oder Benutzer übertragen werden (z. B. via RFC, HTTP, FTP, E-Mail).

Deshalb müssen Institutionen zunächst festlegen, welche Daten im SAP-System geschäftskritisch oder vertraulich sind oder auch besonderen gesetzlichen Anforderungen unterliegen. Es müssen immer Berechtigungsprüfungen durchgeführt werden, wenn diese Daten angezeigt, übermittelt oder exportiert werden. Jeder (beabsichtigte) Abfluss dieser Daten muss dokumentiert sein.

APP.4.6.M16 Verzicht auf systemabhängige Funktionsausführung (S)

Mit der Systemvariablen SY-SYSID kann ein ABAP-Programm abfragen, auf welchem System es gerade ausgeführt wird. Sie kann somit dazu verwendet werden, den Kontrollfluss eines Programms systemabhängig zu beeinflussen.

Entwickler stellen mit SY-SYSID beispielsweise sicher, dass während des Tests auf dem Qualitätssicherungssystem keine produktiven Schnittstellen mit Testdaten bedient werden. Da die System-ID eines Qualitätssicherungssystems nicht der System-ID des Produktivsystems entspricht, werden die betreffenden Codeabschnitte in der Testphase nicht durchlaufen. Diese Vorgehensweise kann jedoch dazu führen, dass ungetestete Codeabschnitte oder auch Schadcode in das Produktivsystem übertragen werden.

Das folgende Beispiel zeigt, wie geschäftskritische Daten im Produktivbetrieb an einen anderen Empfänger umgeleitet werden können:

```
IF sy-sysid = 'P23'
lv_mailtarget = 'vicky.lieks@datenklau.ko'.
ELSE.
lv_mailtarget = 'hr@careless-company.com'.
ENDIF.
```

Sollten systemabhängige Funktionsausführungen tatsächlich erforderlich sein, sollte detailliert dokumentiert werden, wo sie verwendet werden und warum. Dadurch kann die Qualitätssicherung zumindest mittels manueller Codeanalyse erfolgen.

APP.4.6.M17 Verzicht auf mandantenabhängige Funktionsausführung (S)

Mit der Systemvariablen SY-MANDT kann ein ABAP-Programm abfragen, in welchem Mandanten es gerade ausgeführt wird. Sie kann somit dazu verwendet werden, den Kontrollfluss eines Programms mandantenabhängig zu beeinflussen. Entwickler benutzten die Variable oft in Mehrmandantensystemen, um leicht abweichende Anforderungen für verschiedene Mandanten innerhalb eines Programms abzubilden.

Da die Mandantenstruktur eines Qualitätssicherungssystems nicht notwendigerweise der des Produktivsystems entspricht, werden die betreffenden Codeabschnitte in der Testphase nicht durchlaufen. Diese Vorgehensweise kann jedoch dazu führen, dass ungetestete Codeabschnitte oder Schadcode in das Produktivsystem übertragen werden. Das folgende Beispiel zeigt, wie geschäftskritische Daten im Produktivbetrieb an einen anderen Empfänger umgeleitet werden können:

```
IF sy-mandt = '100'
lv_mailtarget = 'vicky.lieks@datenklau.ko'.
ELSE.
lv_mailtarget = 'hr@careless-company.com'.
```

```
ENDIF.
```

Sollten mandantenabhängige Funktionsausführungen trotzdem erforderlich sein, muss detailliert dokumentiert werden, wo sie verwendet werden und warum. Dadurch kann die Qualitätssicherung auf den entsprechenden Mandanten stattfinden oder zumindest mittels manueller Codeanalyse erfolgen.

APP.4.6.M18 Vermeidung von Open-SQL-Injection-Schwachstellen (S)

Mit dynamischem Open SQL können bestimmte Teile eines Open SQL-Befehls auch zur Laufzeit übergeben werden. Werden in einem solchen dynamischen Teil des Open SQL-Befehls Benutzereingaben verarbeitet, kann der betroffene Befehl manipuliert werden. Dadurch kann ein Angreifer sich unberechtigten Zugang zu Daten verschaffen und diese, je nach Art der Schwachstelle, sogar manipulieren oder löschen.

Beispiel:

Der Entwickler eines ABAP-Programms hat vorgesehen, dass ein Benutzer aus einer Tabelle nur Sätze löschen darf, die zu diesem Benutzer gehören. Zusätzlich soll der Benutzer die zu löschenden Sätze aber zeitlich weiter einschränken können. Hierzu wird ihm die Eingabe eines weiteren Filters auf das Feld TA_REF angeboten.

```
DATA: cl_where TYPE string.

DATA: cref TYPE string.

DATA: request TYPE REF TO IF_HTTP_REQUEST.

cref = request->get_form_field( 'reference' ) .

CONCATENATE ‘uname = ‘’’ sy-uname ‘’’ INTO cl_where.

CONCATENATE cl_where ‘ AND ta_ref = ‘’’ cref ‘’’ INTO cl_where.

DELETE FROM orders WHERE (cl_where).
```

Mit der Eingabe des Wertes "000100" werden also alle Einträge gelöscht, die der Bedingung:

```
UNAME = <aktueller Benutzer> AND TA_REF = ‘000100‘
```

genügen.

Ändert der Benutzer jedoch seine Eingabe in *’OR TA_REF LIKE ,%‘*, so wird daraus die Gesamtbedingung:

```
UNAME = <aktueller Benutzer> AND TA_REF = ‘ ‘ ’ OR TA_REF LIKE ,%‘
```

In diesem Fall würde die Tabelle also komplett gelöscht werden.

Die Verwendung von dynamischem Open SQL sollte deshalb durch die Entwicklungsrichtlinie verboten sein. Sollten dennoch Datenbankzugriffe mit dynamischen SQL-Bedingungen notwendig sein, sollten möglichst keine Benutzereingaben in die Abfrage übertragen werden. Wenn das dennoch der Fall ist, sollte die Benutzereingabe zwingend geprüft werden (Output Encoding). Der SAP-Hinweis 1520356 "SQL-Injections vermeiden" (siehe [1520356]) beschreibt Module, die dazu verwendet werden können.

Zusätzlich ist in höheren Releases auch die eingebaute Funktion ESCAPE verfügbar, mit der das Encoding durchgeführt werden kann.

Beispiel:

```
DATA: cl_where TYPE string.

DATA: cref TYPE string.

DATA: request TYPE REF TO IF_HTTP_REQUEST.

cref = request->get_form_field( 'reference' ) .

cref = cl_abap_dyn_prg=>escape_quotes( cref ).

CONCATENATE ‘uname = ‘’’ sy-uname ‘’’ INTO cl_where.

CONCATENATE cl_where ‘ AND ta_ref = ‘’’ cref ‘’’ INTO cl_where.
```

```
DELETE FROM orders WHERE (cl_where).
```

APP.4.6.M19 Schutz vor Cross-Site-Scripting (S)

Bei Cross-Site-Scripting-Angriffen (XSS) werden spezielle Befehlssequenzen in eine HTML-Seite eingefügt, wenn in dieser Seite (Benutzer-)Eingaben erfolgen. Ruft der Benutzer die HTML-Seite auf, wird der eingeschleuste Code im Browser ausgeführt.

Cross-Site-Scripting stellt in der Regel einen Angriff auf Client-Systeme dar. Um XSS-Angriffe zuverlässig abzuwehren, sollte jede Benutzereingabe mithilfe der ESCAPE_XSS_*-Methoden der SAP-Standard-Klasse CL_ABAP_DYN_PRG encodiert werden. Zusätzlich ist in höheren Releases auch die eingebaute Funktion ESCAPE verfügbar, mit der das Encoding durchgeführt werden kann.

Der beste Schutz wird jedoch erreicht, indem auf selbst entwickeltes HTML in Business-Server Pages-(BSP)-Anwendungen oder HTTP-Handlern verzichtet wird. Rendering-Mechanismen wie Web Dynpro oder UI5 generieren HTML automatisiert. Da Entwickler nicht mehr in das HTML-Rendering eingreifen können, sind diese Plattformen deutlich sicherer.

APP.4.6.M20 Keine Zugriffe auf Daten eines anderen Mandanten (S)

Ein SAP-System besitzt immer mindestens einen, oft aber auch mehrere Mandanten. Ein Mandant ist die organisatorisch höchste Einheit im System. Jeder Mandant kann betriebswirtschaftlich als Konzern, Unternehmen, Behörde oder Betrieb aufgefasst werden. Der Mandant stellt somit eine Einheit dar, die handelsrechtlich, organisatorisch und auch datentechnisch abgeschlossen ist. Er verfügt über von anderen Mandanten getrennte Sätze an Tabellen und Daten.

Werden in ABAP-Programmen nur Open SQL-Befehle verwendet, ist durch den SAP-Kernel sichergestellt, dass standardmäßig nur auf Daten aus dem aktuellen Mandanten zugegriffen wird. Dadurch ist gewährleistet, dass die Daten unterschiedlicher Mandanten voneinander getrennt sind. Es gibt jedoch verschiedene Möglichkeiten, diese automatische Mandantentrennung zu umgehen. So kann mit Open SQL durch den Zusatz CLIENT SPECIFIED auf Daten aus einem anderen Mandanten zugegriffen werden. Auch wenn natives SQL benutzt wird, ist ein direkter Zugriff auf andere Mandanten möglich (siehe APP.4.6.M14 Vermeidung von nativen SQL-Anweisungen). Ein solcher Zugriff auf Daten eines anderen Mandanten ist vor allem bei Systemen gefährlich, deren SAP-Mandanten verschiedene Unternehmen abbilden. Das betrifft insbesondere Hosting-Systeme.

Deswegen sollte es Entwicklern über die Entwicklungsrichtlinie verboten werden, die Open SQL-Option CLIENT SPECIFIED sowie EXEC SQL zu benutzen.

APP.4.6.M21 Verbot von verstecktem ABAP-Quelltext (S)

In allen SAP-Netweaver-Versionen älter als 7.50 ist es möglich, den ABAP-Quellcode durch Voranstellen einer bestimmten Zeichensequenz zu verstecken. Diese Methode wird z. B. von Drittanbietern verwendet, um die Entschlüsselung von Lizenzschlüsseln nicht offenlegen zu müssen. In Eigenentwicklungen sollte diese Technik jedoch verboten sein, da der ABAP-Code sonst weder manuell überprüft, noch von automatischen Codeanalysen erfasst werden kann. Bestenfalls ist ein funktionaler Test möglich, dessen Ergebnis durch Verwendung weiterer illegaler Methoden aber leicht verfälscht werden kann.

Techniken, die das Lesen von Quellcode verhindern, sollten deswegen nicht erlaubt sein. Das Verbot ist in der Entwicklungsrichtlinie zu dokumentieren.

APP.4.6.M22 Einsatz von ABAP-Codeanalyse Werkzeugen (H)

Werden SAP-Systeme in Umgebungen mit erhöhtem Schutzbedarf eingesetzt, so sollte eine automatisierte Überprüfung der Eigenentwicklungen auf sicherheitsrelevante Programmierfehler, funktionale und technische Fehler sowie qualitative Schwachstellen durchgeführt werden. Dies kann mit Werkzeugen von der SAP wie SAP ABAP Test Cockpit oder SAP Code Inspector erfolgen oder mit Werkzeugen von Fremdanbietern.

3. Weiterführende Informationen

3.1. Wissenswertes

Hier werden ergänzende Informationen aufgeführt, die im Rahmen der Maßnahmen keinen Platz finden, aber dennoch beachtenswert sind. Derzeit liegen für diesen Baustein keine entsprechenden Informationen vor. Sachdienliche Hinweise nimmt die IT-Grundschutz-Hotline gerne unter grundschutz@bsi.bund.de entgegen.

3.2. Quellenverweise

[1497003] SAP-Hinweis 1497003: "Mögliche Directory Traversals in Anwendungen", SAP SE

[1520356] SAP-Hinweis 1520356: "SQL-Injections vermeiden", SAP SE