נוהל פיתוח מאובטח בABAP

הקדמה

בקוד ABAP קיימים סיכוני אבטחה רבים העלולים לאפשר לתוקף לנצל אותן על מנת להשיג מידע רגיש, לשנות נתונים ואף להשבית מערכות. נוהל זה מיועד למפתחים אשר מפתחים קוד במערכות ABAP בארגון. פיתוח על-פי הנהלים המופיעים במסמך זה יוריד משמעותית את הסיכונים הנובעים מקוד לא מאובטח במערכות השונות.

סוגי הסיכונים בקוד ABAP

מרבית הסיכונים נובעים ממידע שנכנס למערכת (למשל מידע שמשתמש הזין) ולא עבר בדיקות כראוי. שימוש בקוד SQL דינאמי למשל יכול לאפשר לתוקף לגשת בצורה לא מורשית לבסיס הנתונים. באותה שיטה שימוש בשם קובץ דינאמי עלול לאפשר לתוקף לגשת, לשנות או למחוק קבצים משרת האפליקציה. להלן סוגי הסיכונים השונים, כל סיכון יפורט בהמשך.

1. Manipulation of dynamic Open SQL (Open SQL Injection)
2. Manipulation of SQL statements (Native SQL Injection)
3. Manipulation of dynamically generated ABAP code (ABAP Command Injections)
4. Injections of operating system commands
5. Potential unauthorized access to directories and files (Directory Traversal)
6. Insuffient authorization checks of user administration bypassed
7. Potential back doors
8. Possible attacks using Web technologies
9. Further checks

Manipulation of dynamic Open SQL (Open SQL Injection)

שימוש ב-WHERE דינאמי

השימוש ב-WHERE בצורה דינאמית חושף את השליפה להזרקת תנאים נוספים כמו OR, ובכך מאפשר להגדיל את כמות הנתונים שנשלפים וחוזרים לתכנית בצורה לא מבוקרת.

הפתרון האידיאלי הוא להפחית את השימוש ככל שניתן ב-WHERE דינאמי, היכן שלא ניתן, אנו ממליצים לבצע בדיקות קלט מקיפות על הטקסט שמועבר ל-WHERE.

מומלץ לעשות שימוש בפונקציונאליות של ה-CL_ABAP_DYN_PRG' CLASS' על מנת לחסום את הזרקת הקוד. למשל השימוש במתודה QUOTE מוסיפה גרשיים לכל פקודת ה-WHERE ובכך חוסמת אפשרות לבצע מניפולציות לדוגמא:

```
PARAMETERS: column   TYPE c LENGTH 30 ,

        value  TYPE c LENGTH 30 .



AT SELECTION-SCREEN .

 TRY .

    cl_abap_dyn_prg=>check_column_name( column ) .

  CATCH cx_abap_invalid_name .

    MESSAGE 'Not allowed' TYPE 'E .'

 ENDTRY .



START-OF-SELECTION .



 value = cl_abap_dyn_prg=>quote( value ) .

 DATA(cond_syntax) = column && ` = ` && value .
```

```
TRY .
   SELECT *
        FROM spfli
        WHERE (cond_syntax)
        INTO TABLE @DATA(spfli_tab) .
  CATCH cx_sy_dynamic_osql_error .
   MESSAGE `Wrong WHERE condition!` TYPE 'I .'
ENDTRY.
```

בדוגמא למעלה, במידה ונכניס למשתנה value את הערך "'LH' OR CARRID <> 'LH" ולא נשתמש במתודה QUOTE כמו בדוגמא, נגרום לWHERE בDB להיראות ככה:
"'LH' <> 'LH' OR CARRID = "' וזה כמובן יחזיר את כל הטבלה.

בדומה לשימוש ב-WHERE דינאמי כמו הדוגמא לעיל, ניתן לבצע הזרקה של קוד גם לשדה I_FILTER כאשר נעשה שימוש במתודה CREATE_QUERY. גם פה הפתרון יהיה דומה.

הערה: נדרש לבצע את אותה הבדיקה גם ל-HAVING.

שימוש ב-SET דינאמי בפקודות UPDATE

תוקף פוטנציאלי עלול לעשות שימוש ב-SET דינאמי על מנת להזריק קוד שיוביל לשינוי של ערכים נוספים באופן לא צפוי.

בדומה לסעיף הקודם, שימוש בפקודות SQL סטטיות ולא דינאמיות פותר את כל בעיות האבטחה ב-OPEN SQL, אך אם יש צורך לעשות שימוש ב-SET דינאמי, נדרש לבדוק את הטקסט לפני שליחתו לשליפה.

גם פה ההמלצה היא להשתמש במתודה QUOTE של ה-CLASS 'CL_ABAP_DYN_PRG'.

דוגמא:

```
DATA(set_expr) =

 COND string( WHEN name IS NOT INITIAL

          THEN ` NAME && ` =

          cl_abap_dyn_prg=>quote( name ) && (

 COND string( WHEN street IS NOT INITIAL

          THEN ` STREET &&  ` =

          cl_abap_dyn_prg=>quote( street ) && (

 COND string( WHEN city IS NOT INITIAL

          THEN ` CITY && ` =

          cl_abap_dyn_prg=>quote( city ) && (

 COND string( WHEN postcode IS NOT INITIAL

          THEN ` POSTCODE && ` =

          cl_abap_dyn_prg=>quote( postcode ) .(
```

```
TRY .
   UPDATE scustom SET (set_expr) WHERE id = @id .
 CATCH cx_sy_dynamic_osql_syntax .
   cl_demo_output=>display( 'Wrong input' ) .
ENDTRY.
```

שימוש בשם טבלה דינאמי (הן ב-SELECT והן ב-UPDATE/INSERT)

שימוש בשם טבלה דינאמי מסוכן במיוחד מכיוון שניתן על-ידי הזרקת קוד להגיע לטבלאות אחרות במערכת. דוגמא נפוצה היא השימוש בשליפה על מנת להשיג פרטים של משתמשים חזקים וכו'.

בדומה לסעיפים הקודמים, אנו צריכים לוודא כי אנו מוודאים ששם הטבלה שאנו מעבירים הוא נכון ואכן רלוונטי למה שאנו שולפים.

על מנת לבצע את האימות, נעשה שימוש במתודות ולידציה שונות מתוך ה-CLASS 'CL_ABAP_DYN_PRG'. במקרה הזה השימוש במתודה QUOTES לא רלוונטי ולכן נעשה שימוש במתודות אחרות.

ראשית נבצע בדיקה שהטקסט שהגיע הוא טבלה והיא נמצאת ב-Package הרלוונטי לתכנית ע"י השימוש במתודה CHECK_TABLE_NAME_STR, המתודה תבדוק האם שם הטבלה קיים והאם היא ב-Package שצויין. כך נוכל למנוע זליגה של נתונים מתחומים שלא קשורים למה שהתכנית עוסקת בו.

לדוגמא:

```
  dbtab = to_upper( dbtab ).
  TRY.
    dbtab = cl_abap_dyn_prg=>check_table_name_str(
          val = dbtab
          packages = 'SAPBC_DATAMODEL.) '
   CATCH cx_abap_not_a_table.
    out->display( 'Database table not found' ).
    LEAVE PROGRAM.
   CATCH cx_abap_not_in_package.
    out->display(
'       Only tables from the flight data model are allowed.) '
```

```
    LEAVE PROGRAM.
ENDTRY.


TRY.
    CREATE DATA dref TYPE (dbtab).
    ASSIGN dref->* TO <wa.<
    SELECT*
        FROM (dbtab) UP TO @rows ROWS
        INTO @<wa.<
     out->write( <wa> ).
    ENDSELECT.
    out->display.( )
  CATCH cx_sy_create_data_error.
    out->display( 'Error in data creation' ).
ENDTRY.
```

בנוסף ניתן לבצע וולידציה נוספת על-ידי שימוש בWhitelist בעזרת המתודות CHECK_WHITELIST_TAB ו-CHECK_WHITELIST_STR .

שימוש בשמות דינאמיים של עמודות

בעת שימוש בשמות דינאמיים של עמודות, אנו עלולים לאפשר לגורם עוין להזריק שמות אחרים או נוספים לשליפה. גם כאשר משתמשים בפקודה INTO CORRESPONDING FIELDS, התוקף עלול להשתמש בשינוי השם בתוך השליפה ("COL1 AS COL2") כדי להחזיר את העמודה החדשה לתוך עמודה קיימת.

ההמלצה היא להשתמש בWHITELIST הכולל את שמות העמודות המורשות לשימוש. ניתן לבצע זאת בעזרת אחת המתודות CHECK_WHITELIST_TAB או CHECK_WHITELIST_STR .

לדוגמא:

```
DATA(whitelist) = VALUE whitelist( ( `STRING1` ) ( `STRING2` ) ( `STRING3` ) ).

 TRY.

 p_input = cl_abap_dyn_prg=>check_whitelist_tab(

 val = to_upper( p_input )

 whitelist = whitelist.)

 CATCH cx_abap_not_in_whitelist.

 cl_demo_output=>write)

` Only the following strings are allowed.) `:

 cl_demo_output=>display( whitelist ).

 LEAVE PROGRAM.

 ENDTRY.
```

הערה: נדרש לבצע את אותה הבדיקה גם ל-GROUP BY.

Manipulation of SQL statements (Native SQL Injection)

בניגוד לשימוש ב Dynamic Open SQL, השימוש בNative SQL מעביר פקודות שלמות בטקסט לבסיס הנתונים ללא שום בדיקה בשרת האפליקציה. דבר שמגדיל משמעותית את הסיכון. לבן ההמלצה היא לא לעשות שימוש בNative SQL בכלל. במידה ואין ברירה, נשתמש בכללים הבאים כדי למנוע מתוקף לעשות שימוש בקוד שלנו בצורה זדונית:

1. אין לעשות שימוש בטקסט שהתקבל על-ידי המשתמש או על-ידי ממשק חיצוני בתוך השליפה.
2. שימוש ב-? כהכנה למשתנים דינאמיים היכן שנדרש, ושימוש בפונקציה SET_PARAM ב-Class CL_SQL_STATEMENT על מנת לקבוע את ערך המשתנים.
3. לכל משתנה שנכנס לשליפה יש לבצע בדיקת קלט עם המתודות הבאות של ה-CL_ABAP_DYN_PRG Class:
   - ESCAPE_QUOTES
   - QUOTE
   - CHECK_CHAR_LITERAL
   - CHECK_INT_VALUE
   - CHECK_VARIABLE_NAME
   - CHECK_COLUMN_NAME
   - CHECK_TABLE_OR_VIEW_NAME_STR
   - CHECK_TABLE_OR_VIEW_NAME_TAB
   - CHECK_TABLE_NAME_STR
   - CHECK_TABLE_NAME_TAB
   - CHECK_WHITELIST_STR
   - CHECK_WHITELIST_TAB

הסיכונים בשימוש ב-Native SQL:

- ביצוע SQL Injection על מנת לאחזר או לשנות מידע במערכת (בדומה לOpen SQL)
- ביצוע SQL Injection על מנת להריץ פקודות DDL על מנת לשנות מבני נתונים במערכת
- ביצוע SQL Injection על מנת להפעיל API של בסיס הנתונים למשל: יצירת משתמש, איפוס סיסמה וכו'.

Manipulation of dynamically generated ABAP code

(ABAP Command Injections)

שימוש בקוד ABAP אשר מחולל דינאמית עלול לאפשר לתוקף פוטנציאלי להריץ קוד זדוני במספר דרכים. בכל שימוש בחילול קוד דינאמי יש לבצע בדיקות קלט מקיפות של כל חלק בקוד שהגיע ממקור חיצוני.

הדרכים להכנסת קוד זדוני:

1. בעת השימוש ב INSERT REPORT ו-GENERATE SUBROUTINE POOL ניתן להזריק קוד זדוני שירוץ ביחד עם הקוד המתוכנן.
2. בעת השימוש בפונקציה RFC_ABAP_INSTALL_AND_RUN התוקף יכול ליצר ולהריץ כל קוד שיבחר גם במערכות מרוחקות.
3. בעת השימוש בWHERE דינאמי מעל Internal Table (ראה סעיף Open SQL)

בשתי הדרכים הראשונות אנו עלולים לאפשר לתוקף להריץ כל פעולה במערכת ולעקוף בדיקות הרשאה אם יבחר בכך. לכן החשיבות להגן על ערכים טקסטואלים הנכנסים בקלט למערכת במיוחד.

את ההגנה נעשה גם כן תוך שימוש במתודות הוולידציה מתוך ה-CL_ABAP_DYN_PRG CLASS.

Injections of operating system commands

שימוש בפקודות מערכת הפעלה (OS Command)

השימוש בפקודות מערכת הפעלה מסוכן מאוד ואין לעשות בו שימוש.

במקרים חריגים בהם לא ניתן למצוא שימוש אחר יש לעשות שימוש אך ורק דרך SM69, שם נגדיר פקודה חדשה. את הפקודה צריך להקים לכל סוג של מערכת הפעלה שקיים בשרת אפליקציה.

בפקודה ניתן לציין פרמטרים שיעברו לפקודה תמיד בעת הרצתה וניתן לסמן
'Additional Parameters Allowed', השימוש בהעברת פרמטרים מהקוד לפקודת מערכת הפעלה מסוכן עוד יותר ולכן העדיפות היא לא לסמן.

במידה וקיים צורך כזה ואכן מעבירים פרמטרים מהתוכנית, נדרש לבדוק את כל הפרמטרים שמועברים ע"י שימוש במתודות הוולידציה מתוך ה-CL_ABAP_DYN_PRG CLASS. ניתן ואף מומלץ ליישם את הבדיקה ב Function Module נפרד שמבוסס על SXPG_DUMMY_COMMAND_CHECK, ולציין את שם הFunction Module תחת ה 'Check Module' בתוך SM69. כך למעשה נוודא שלא משנה מי מריץ ומאיפה, הבדיקה תעשה.

OS Injection גישה לקבצים בעזרת OPEN DATASET

בגישה לקבצים במערכת ההפעלה לא קיים סיכון להזרקת קוד אלא אם עושים שימוש בFILTER כחלק מפקודת הOPEN DATASET. השימוש בFILTER גורם ל-SAP לשרשר ב-PIPE את הערך שמועבר בפקודת הFILTER למערכת ההפעלה ובכך אנו חשופים ל OS Injection.

אין לעשות שימוש בFILTER כחלק מפקודת OPEN DATASET.

Potential unauthorized access to directories and files (Directory Traversal)

מניפולציה על שם הקובץ בגישה ל-File system

בשימוש בפקודות המפרוטות בהמשך קיימת סכנה של הפעלת מניפולציה על שם הקובץ ובכך לגשת לקבצים או תיקיות במערכת ההפעלה שאליהם לא אמורה להיות גישה למשתמשי ה-SAP.

- OPEN DATASET
- DELETE DATASET
- המתודה CREATE_UTF8_FILE_WITH_BOM ב-CL_ABAP_FILE_UTILITIES CLASS

על מנת להגן על הפרמטרים שעוברים לפקודות הנ"ל, ההמלצה היא להשתמש ב Logical file דרך הטרנזקציה FILE ולא להשתמש בשם קובץ דינאמי שהמשתמש יכול לשנות.

במידה ונדרש לתת למשתמש לבחור שם של קובץ, יש לבדוק את הנתיב שאותו בחר המשתמש אל מול Whitelist ובן להשתמש בפונקציה FILE_VALIDATE_NAME על מנת לוודא ששם הקובץ או התיקייה הם חוקיים.

מומלץ להשתמש בפונקציה זו גם בשימוש ב Logical file name לאחר אחזור שם הקובץ.

Insufficient authorization checks of user administration bypassed

שימוש בAUTHORITY-CHECK ובדיקת התוצאה

כבכל אצבע, לפני גישה לנתונים (קריאה או כתיבה) באופן ישיר שלא דרך BAPI, נדרש לבדוק האם למשתמש יש הרשאה לביצוע הפעולה. הבדיקה כמובן תתבצע עם AUTHORITY-CHECK ומיד לאחר מכן לבדוק את התוצאה ב- SY-SUBRC ולפעול לפיה (להמשיך או להפסיק את התכנית).

בדיקת התוצאה לאחר שימוש בפונקציות אבטחה

לאחר השימוש בפונקציות האבטחה הבאות, יש לבדוק מיד לאחר מכן את התוצאה (למשל בSY-SUBRC) ולא לאחר פקודות אחרות שרצות שעלולות לשנות את הערך של התוצאה:

- AUTHORITY_CHECK
- AUTHORITY_CHECK_TCODE
- SU_RAUTH_CHECK_FOR_USER
- VIEW_AUTHORITY_CHECK
- FILE_VALIDATE_NAME
- AUTHORITY_CHECK_DATASET
- AUTHORITY_CHECK_RFC

בדיקת הרשאה למשתמש אחר (לא המשתמש הפעיל)

ניתן לבדוק הרשאה למשתמש אחר על-ידי השימוש בפקודה AUTHORITY-CHECK והתוספת FOR USER או השימוש בפונקציה SU_RAUTH_CHECK_FOR_USER.

השימוש הנ"ל נדרש כאשר, למשל, רוצים לבדוק האם קיימת הרשאה למשתמש שאמור להריץ ג'וב מתוזמן.

הסיכון בשימוש הנ"ל קיים במידה ויש למשתמש יכולת לבצע מניפולציה על הערך של המשתמש שמועבר לפקודה. בעזרת מניפולציות ניתן לגרום לפקודה לאשר שקיימת הרשאה גם אם לא קיימת.

לכן יש לבדוק את הערך שמועבר לפקודה אל מול WHITELIST שמוגדר מראש.

בנוסף, בדוגמא לעיל, נדרש לבדוק את ההרשאה גם ברמת הג'וב שרץ.

בבדיקת הרשאה למשתמש הפעיל, **לא** נדרש להשתמש בתוספת FOR USER, ולהעביר כפרמטר את sy-uname או syst-uname.

בדיקת הרשאות בשימוש ב CALL TRANSACTION

כאשר אנו משתמשים בפקודה CALL TRANSACTION יש להגדיר האם נדרש לבצע בדיקות הרשאה על הפעלת הטרנזקציה או לא. ההחלטה תתבצע ע"י אחת מהתוספות הבאות:

- WITH AUTHORITY-CHECK
- WITHOUT AUTHORITY-CHECK

התוספת משפיעה אך ורק על הבדיקה על הפעלת הטרנזקציה, כלומר:

- בדיקת האובייקט S_TCODE עם שם הטרנזקציה
- אובייקט ההרשאה שהוגדר לטרנזקציה בSE93

ברירת המחדל צריכה להיות להשתמש בתוספת WITH AUTHORITY-CHECK, רק במקרים חריגים מאוד נדרש להפעיל טרנזקציה ללא בדיקת ההרשאות.

Potential back doors

שימוש בשמות משתמשים Hard Coded

שימוש בשמות משתמשים בקוד לעיתים משמש תוקפים פוטנציאלים. לדוגמא תוקף יכול להסיק לאיזה משתמש אמורה להיות הרשאה מסויימת ולנסות להשתמש במשתמש הזה על מנת לבצע פעולה רגישה.

אין לעשות שימוש בשמות משתמשים בתוך הקוד.

שימוש בשמות שרתים Hard Coded

שימוש בשמות של שרתים בתוך הקוד בדרך כלל מסמן קוד של בדיקות שנשכח ועלול לעזור לתוקף להשיג מידע שאינו מורשה לו.

אין לעשות שימוש בשמות שרתים בתוך הקוד.

ניתן לבדוק האם API כלשהו יכול להחזיר את שם השרת הרלוונטי (לדוגמא SY-HOST או CL_ABAP_SYST=>GET_HOST_NAME)

שימוש בשמות מערכות (SID) Hard Coded

שימוש בשמות המערכות בתוך הקוד בדרך כלל מסמן קוד של בדיקות שנשכח ועלול לעזור לתוקף להשיג מידע שאינו מורשה לו.

אין לעשות שימוש בשמות מערכות בתוך הקוד.

ניתן לבדוק האם שימוש נכון בקסטומיזציה או API כלשהו יכול להחזיר את שם המערכת (למשל שימוש במשתנים בטרנזקציה FILE)

שימוש במספר הClient - Hard Coded

שימוש במספר ה-Client בתוך הקוד בדרך כלל מסמן קוד של בדיקות שנשכח ועלול לעזור לתוקף להשיג מידע שאינו מורשה לו.

אין לעשות שימוש במספר ה-Client בתוך הקוד.

ניתן לבדוק האם שימוש נכון בקסטומיזציה או API כלשהו יכול להחזיר את שם המערכת (למשל שימוש במשתנים בטרנזקציה FILE או SY-MANDT)

שימוש בסיסמאות Hard Coded

סיסמאות אשר נשמרות בתוך הקוד חשופות למשתמשים רבים במערכת ולא ניתנות לניהול ובקרה. לכן אין לעשות שימוש בסיסמאות Hard Coded בשום צורה. הסוגיה נכונה בין אם הסיסמה שמורה בקוד עצמו, בהזנה על-ידי המשתמש, בקריאה מבסיס הנתונים או בכל דרך אחרת.

השימוש הנכון בסיסמאות היכן שנדרש הוא באמצעות ה- Security Store, מנגנון אשר שומר את הסיסמאות בצורה מוגנת ומוצפנת ומאפשר לקבל אותם לפי הרשאה ספציפית.

שימוש בחיבור מוגדר במערכת (SM59) עושה שימוש ב-Secure Store ולכן מוגן.

Possible attacks using Web technologies

cross-site scripting (XSS)

בניית HTML דינאמי שמסתמך על קלט חיצוני (מה-GUI, RFC וכו') עלול לסכן את המשתמש בכך שהקלט יזריק קוד (למשל JS) שיעבור דרך המנוע למשתמש וירוץ שם.

כדי למנוע מקרים כאלו, נדרש לבצע בדיקה או עיבוד כלשהו על הקלט לפני שהוא עובר לHTML הדינאמי.

נבחן את האופציה הטובה ביותר לעשות זאת לפי סוג האפליקציה.

באפליקציות BSP ניתן לקבוע ENCODING גלובאלי ע"י השימוש בפרמטר forceEncode, לדוגמא:

```
<%@page language="abap" forceEncode="html|url|javascript|css"%>
```

ניתן ללמוד יותר בNote מספר 887168.

באפלקציות Web אחרות, ניתן לעשות שימוש במתודות יעודיות לפי הגירסה בה אנו נמצאים:

בגירסאות SAP_BASIS 7.31 ומעלה:

ניתן להשתמש בפונקציה ESCAPE, לדוגמא:

```
out = escape(val = val format = cl_abap_format=>e_xss_ml)

out = escape(val = val format = cl_abap_format=>e_xss_js)

out = escape(val = val format = cl_abap_format=>e_xss_url)

out = escape(val = val format = cl_abap_format=>e_xss_css)
```

בגירסאות נמוכות יותר ניתן להשתמש בCL_ABAP_DYN_PRG - CLASS, לדוגמא:

```
CL_ABAP_DYN_PRG=>ESCAPE_XSS_XML_HTML

CL_ABAP_DYN_PRG=>ESCAPE_XSS_JAVASCRIPT

CL_ABAP_DYN_PRG=>ESCAPE_XSS_CSS

CL_ABAP_DYN_PRG=>ESCAPE_XSS_URL
```

ביצוע Redirect

שימוש במידע חיצוני בקלט עבור ביצוע Redirect ו/או שינוי ערכי הHeader של בקשת ה-HTTP עלול לסכן את המשתמש בביצוע Redirect לאתר אחר ללא ידיעתו.

לדוגמא:

```
DATA RESPONSE TYPE REF TO IF_HTTP_RESPONSE

RESPONSE->REDIRECT( LV_URL ).

RESPONSE->SET_HEADER_FIELD( NAME = 'LOCATION' VALUE = LV_URL ).

RESPONSE->SET_HEADER_FIELD( NAME = 'REFRESH' VALUE = LV_URL ).
```

ההמלצה היא לא להשתמש בקלט חיצוני לביצוע פעולות אלו, במידה וקיימת דרישה כזאת יש לבדוק את הקלט.

לשימוש בRedirect, קיימת מתודה שבודקת האם הURL מאושר:

CL_HTTP_UTILITY=>CHECK_HTTP_WHITELIST

בשינוי של ערכי Header יש לבצע בדיקות ידניות לפי התוכן שאמור להתקבל בכל שדה.

שימוש במתודות Obsolete לביצוע Escape

השימוש במתודות Escape ב-Classes: CL_HTTP_UTILITY, CL_HTTP_SERVER ו-CL_HTTP_CLIENT לא מאושרות יותר לשימוש מכיוון שהן לא עומדות בתקני אבטחת המידע OWASP.

יש להשתמש אך ורק בפונקציה ESCAPE().

Further checks

שימוש בפקודה COMMUNICATION

השימוש בפקודה COMMUNICATION לא מאושר יותר לשימוש על-ידי SAP בשל בעיות האבטחה הקיימות בפקודה. את הפקודה החליפו פונקציות ה-RFC החדשות יותר. לכן אין לעשות שימוש בפקודה COMMUNICATION לחלוטין.

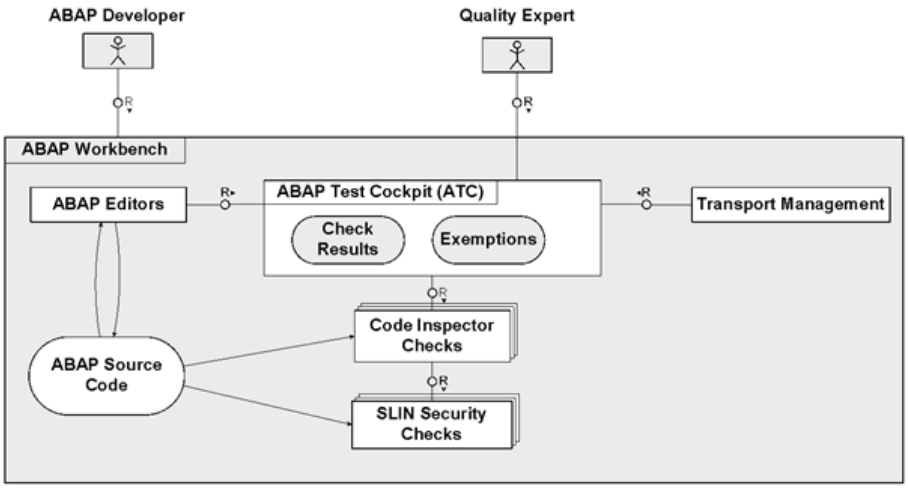
גישה ישירה לטבלאות רגישות

אין לבצע גישה ישירה לטבלאות רגישות כמו טבלאות ניהול המשתמשים וכד'. הגישה לטבלאות אלו צריכה להיות תמיד דרך פונקציות סטנדרטיות בשל הרגישות של הנתונים היושבים בטבלאות. כחלק מהשימוש בפונקציות סטנדרטיות אנו מקבלים מעטפת בדוקה של בדיקת הרשאות, Audit במידה ומופעל וכו'.

ABAP Test Cockpit – ATC

ATC הוא framework של SAP המאפשר להריץ בדיקות סטטיות (Code Inspector) ו Unit tests על תוכניות ABAP בדי לשפר את איכות הקוד. ישנה אינטגרציה מלאה בין הATC לסביבת הפיתוח וניהול הטרנספורטים
בנוסף, בתוך הATC ישנו ניווט ישירות לקוד הנבדק, לדוקומנטציה והמלצות לתיקונים.

ארביטקטורה



מפתח עובד בסביבת הWorkbench וכותב קוד בטרנזקציות השונות (..se38, se37, se24).
בסיום הפיתוח המפתח יכול להריץ באופן ישיר, מתוך ה ABAP Source Code בדיקותATC . בדיקות אלו מריצות את ה Code inspector (איכות הקוד), בדיקות מורחבות (SLIN) ומחלקות Zיות של Unit testing.

באשר עולות שגיאות בATC, על המפתח לטפל בכל אחת מהשגיאות באחת מהדרכים הבאות:
- לטפל בשגיאה על פי ההמלצה של SAP.
- לאשר את השגיאה בעזרת pragma.
- להגיש בקשה ל"פטור" מהשגיאה בגין false positive – לדוגמא חסר בדיקה שהטבלה מלאה לפני FOR ALL ENTERIES אבל יש בדיקה ברמה גבוהה יותר.
- להגיש בקשה ל"פטור" מביוון שאין פתרון אחר - לדוגמא שגיאה בקוד מג'ונרט.

מלבד הרצה מתוך הSource Code, ניתן לקבוע שבדיקות הATC ירוצו אוטומטית בעת שחרור Task או Transport ולהגדיר האם השגיאות עוצרות את תהליך שחרור הTransport או לא.
בנוסף, ניתן לתזמן ג'וב תקופתי אשר בודק את כלל המערכת.

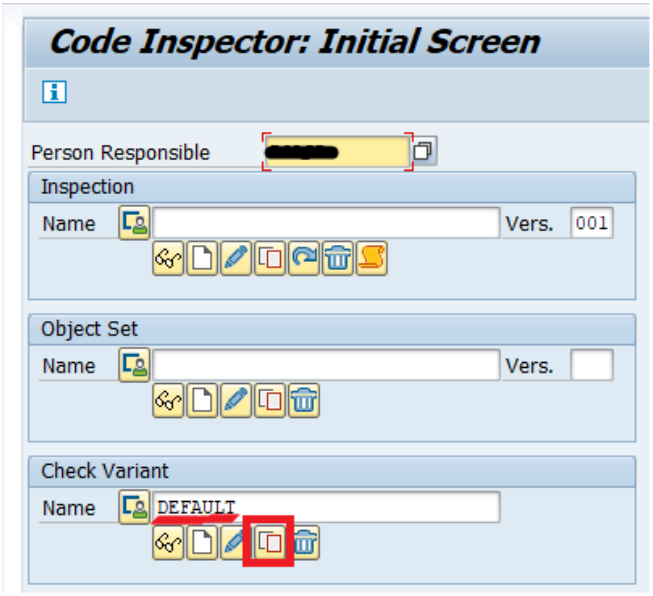
הפעלת הATC

1. קסטום Code Inspector – טרנזקציה SCI

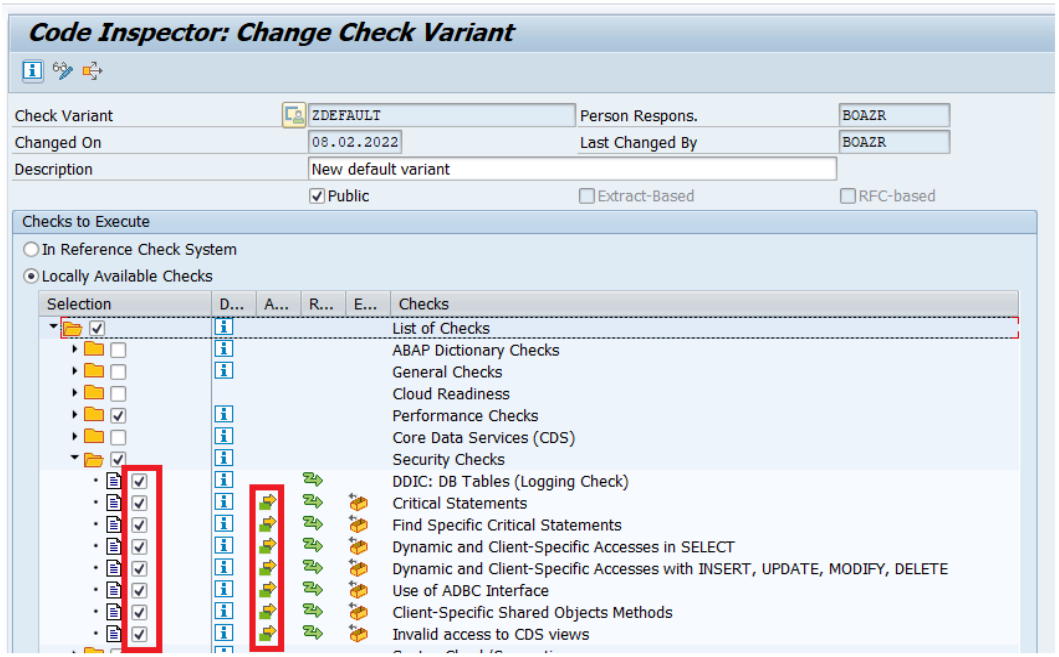
בCode Inspector ישנם הרבה סוגי בדיקות שיכולות להתבצע ועבור כל אחד מהסוגים ישנו סט של בדיקות אפשרי.
לדוגמא – עבור בדיקות מסוג אבטחת מידע, ניתן לבדוק האם קיים select דינאמי, האם יש שימוש בפקודות מערכת קריטיות וכו'. כל אחת מהבדיקות האלו ניתן להתאים אישית, לדוגמא, עבור איזה פקודת מערכת נרצה להתריע.
בנוסף, נוכל עבור כל סוג הודעה לקבוע האם תהיה שגיאה, אזהרה או רק אינפורמטיבית.

ישנו ואריאנט דיפולטי של SAP עם סט בדיקות מומלץ, אך במובן שתמיד יש מקום לביצוע התאמות לארגון.
לדוגמא – בבדיקה האם שמות המשתנים סטנדרטיים, נרצה לקסטם מהם תחיליות המשתנים החוקיות בארגון.
דוגמא נוספת – בארגון שלנו מקפידים על אבטחת מידע ועל כן נרצה להגביר את רמת האכיפה על פרצות בקוד.

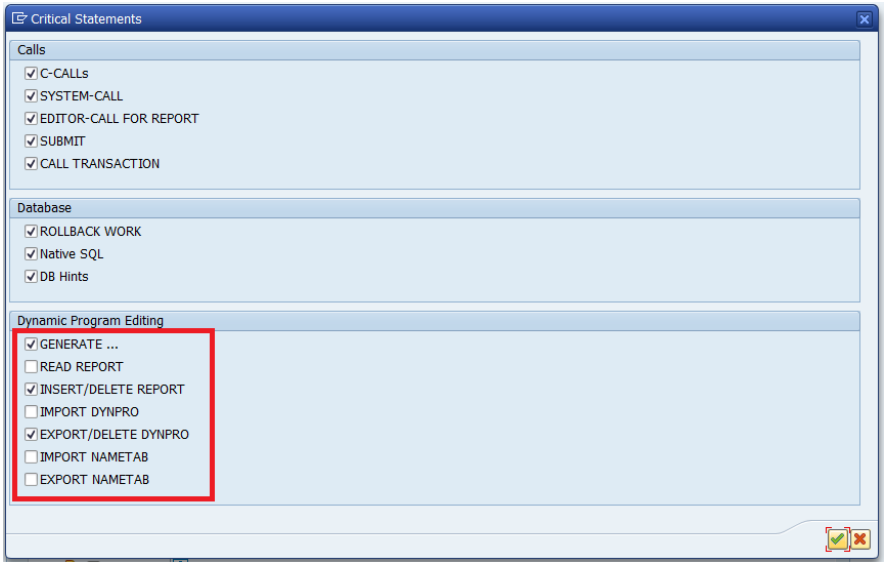
בבדי ליצור ואריאנט חדש לCode Inspector, ניגש לטרנזקציה SCI ונשכפל את ואריאנט DEFAULT של SAP (או ניצור אחד חדש).



בתצוגת הואריאנט, נפתח את המסך לעריכה ונסמן את הCheck boxes הרלוונטיים לנו:



בלחיצה על החץ הירוק, נוכל להתאים את הבדיקה לארגון, לדוגמא לא נרצה לבדוק את כל הפקודות:



2. קסטום הATC

בטרנזקציה ATC ניתן לקסטם ולהתאים את המנגנון אל הארגון. נפרט כאן רק על אופן הפעלת הExemptions ולא על הקסטום הכללי של הATC.

Basic Settings

- נשנה את וריאנט הבדיקות של הCode Inspector לואריאנט החדש שלנו

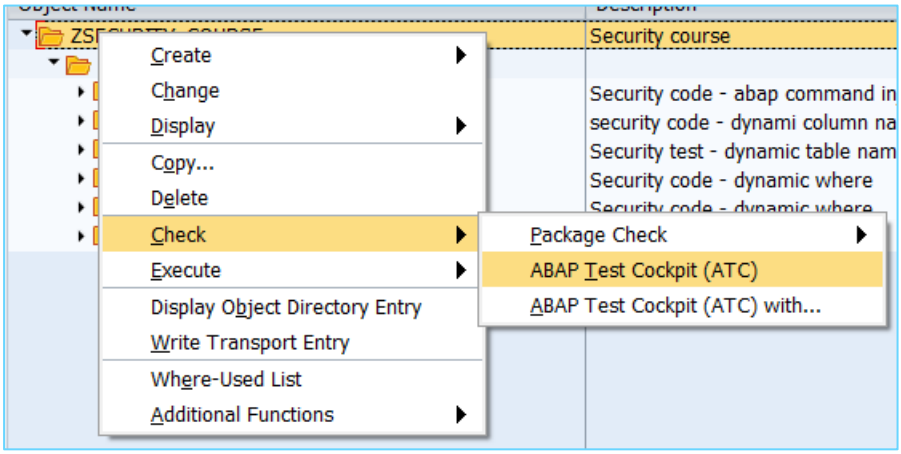
- נגדיר את סביבת הQA להיות ה-Master system
- נאפשר הגשת בקשות לפטורים עבור כל התוצאות.

Exemptions

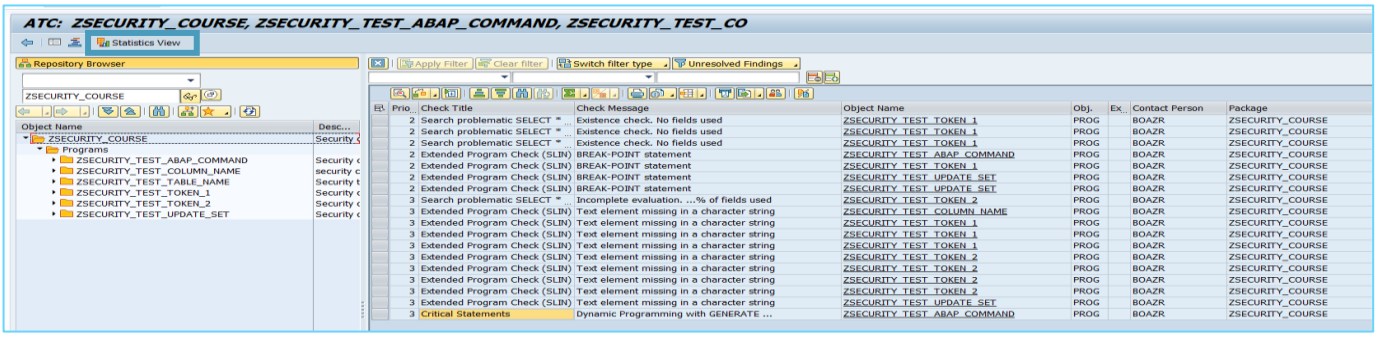
- נתחזק את רשימת המאשרים בסביבת הQA

הרצת הATC

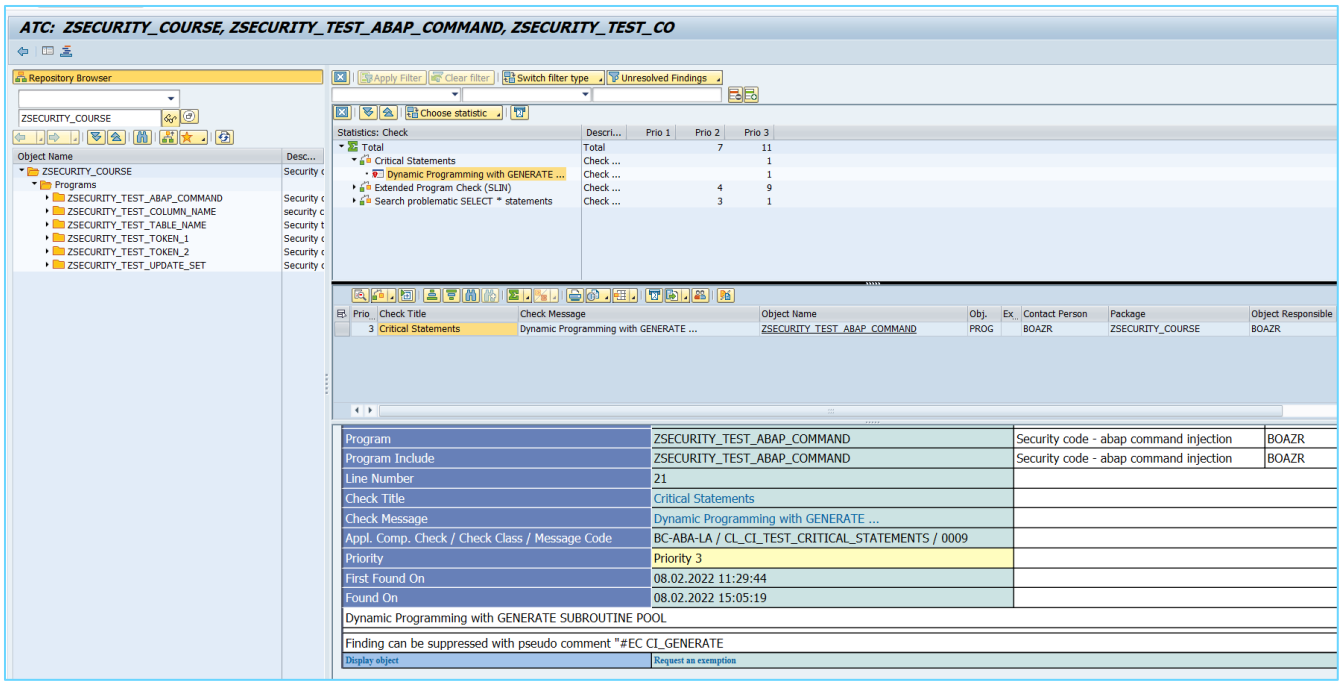
ניתן להריץ מתוך הABAP Workbench את הATC:



ואז מתקבל דו"ח שגיאות, אותו ניתן לרכז בצורה סטטיסטית:



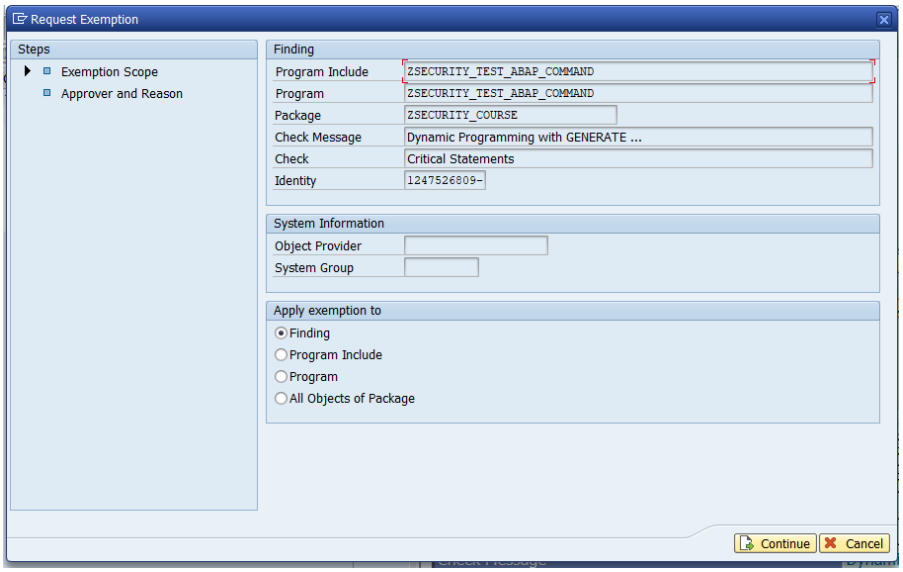
כך הדו"ח יראה בתצוגה סטטיסטית:



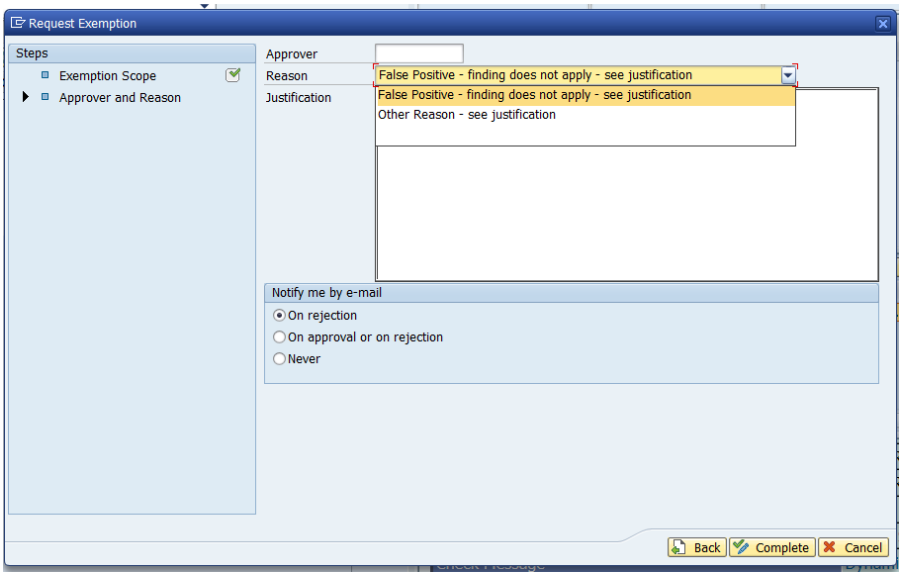
בחלק העליון נראה את עץ סוגי הבדיקות ותחת כל סוג בדיקה נראה את הבדיקות הספציפיות שנמצאו בהן הודעות.
בדאבל קליק על סוג הודעה ספציפית, תופיעה טבלה מתחת עם כל ההודעות של אותה בדיקה ספציפית.
בדאבל קליק על הודעה ספציפית יוצג מידע נוסף על אותה שגיאה – על איזה קוד, חומרת השגיאה, איזו פרגמה מאשרת את השגיאה ואפשרות להגשת בקשה לפטור.

בקשת פטור

במידה ורוצים להגיש בקשה לפטור משינויים עבור שגיאה שהתקבלה בהרצת הATC, בפרטי השגיאה למטה יש לינק להגשת בקשה.
עבור כל בקשה יש לבחור על מה בדיוק מתבקשת ההחרגה – עבור השגיאה הספציפית הזאת, עבור כל התכנית וכו'.



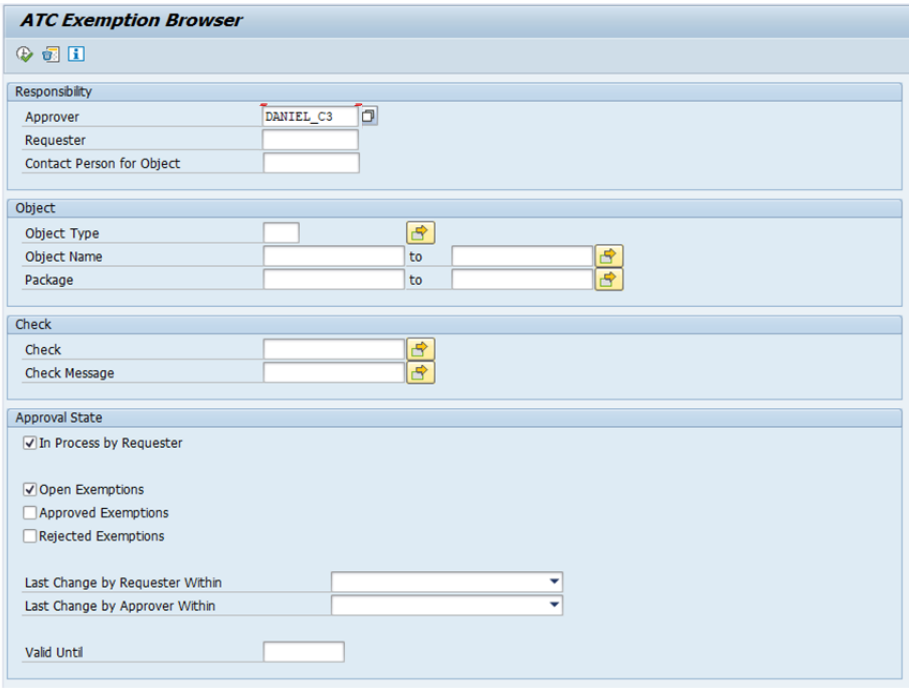
במסך הבא נבחר את המאשר הרלוונטי, סיבה הבקשה ותיאור הבקשה:



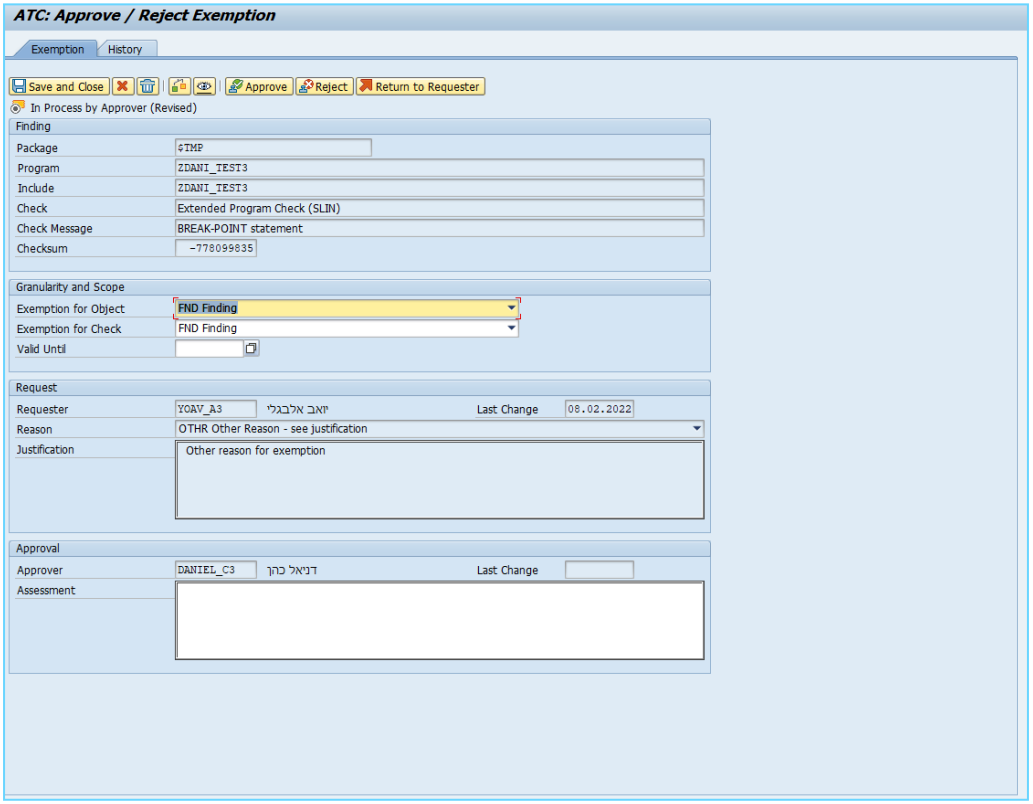
נשים לב שכדי שמנגנון ההתראות יפעל כראוי, יש לקסטם אותו בATC טרם הגשת הבקשה.

אישור פטור

ניגש ל Exemptions Browser דרך הטרנזקציה ATC בסביבת הQA (Master system). נחפש את כל הבקשות הממתינות לטיפולנו:



נבחר את הבקשה בה אנחנו רוצים לטפל, נבחן אותה ונבצע פעולה בהתאם בעזרת הכפתורים בשורה העליונה:



לאחר הפעולה, נראה כי סטטוס הבקשה השתנה: