Secure ABAP Development

דניאל כהן

01/2022

תיאום ציפיות

- הסדנה מנגישה את המסמך 'נוהל פיתוח מאובטח'
- מצלמות פתוחות ☺ 📷

על מה נדבר?

- למה זה חשוב
- סוגי מתקפות ואיך ניתן להתמודד איתן
- ABAP Test cockpit (ATC)

מהי אבטחת מידע?

אבטחת מידע היא הענף העוסק בהגנה מפני גישה, שימוש, חשיפה, ציתות, שיבוש, העתקה או השמדה של מידע ומערכות מידע מצד גורמים שאינם מורשים או זדוניים ולספק סודיות, שלמות וזמינות של המידע ללא תלות בסוג המידע או בצורת האחסון, פיזית או אלקטרונית.

- **שלמות המידע** - הגנה מפני שינוי זדוני של המידע או השמדתו.
- **סודיות המידע** - הגבלת גישה או חשיפה לא מורשית של מידע, כולל הגנה על פרטיות וזכויות קנייניות במידע.
- **זמינות המידע** - שמירה על זמינות ויעילות הגישה אל המידע בכל זמן נתון.

ויקיפדיה

הצורך באבטחת המידע

- בעולם של היום, בו מערכות ארגוניות מניעות את רוב התהליכים הארגוניים והקישוריות בין הרשתות והמערכות הולכת וגדלה, אבטחת המידע בארגון היא הכרחית.
- פגיעה במערכות הארגון יכולה להוביל להפסדים כספיים ופגיעה בלקוחות ופרטיותם.
- על אף מה שחושבים, למרות מנגנון ההרשאות המתקדם בSAP ישנם עדיין סיכוני אבטחת מידע רבים במערכת העלולים לאפשר לתוקף לנצל אותם ולהשיג מידע רגיש, לשנות נתונים ואף להשבית את המערכת.
- כפי שנראה בהמשך, מרבית בעיות האבטחה נובעות מפיתוח לא מאובטח ועל כן האחריות לצמצום הסיכונים הינה אצל מפתחי הSAP.

מה קורה בפועל..

- בעת אפיון ופיתוח תכולה חדשה, כאשר עסוקים בארכיטקטורה, עיצוב, פונקציונאליות, ביצועים וכו' אנחנו שוכחים מהפן של אבטחת המידע בפיתוח ו"בורחים" אל הפתרונות הקלים והנגישים לנו:
  - "נעשה שליפה דינאמית כדי לחסוך בקוד"
  - "בשביל תכנית אחת/זמנית לא נקים Logical File"
  - "לא צריך לבדוק הרשאות לפעולה כי אם למשתמש יש הרשאה עד כאן, למה שלא יהיה לו מכאן והלאה..?"

- צריך לזכור שלתוקף יש את כל הזמן הדרוש כדי למצוא את הדלת האחורית למערכת שלנו.
- לכן ישנו נוהל פיתוח מאובטח המגדיר כיצד יש לפעול ומספק פתרונות לבעיות השונות.

סוגי סיכונים בקוד ABAP

- Manipulation of dynamic Open SQL (Open SQL Injection)
- Manipulation of SQL statements (Native SQL Injection)
- Manipulation of dynamically generated ABAP code (ABAP Command Injections)
- Injections of operating system commands
- Potential unauthorized access to directories and files (Directory Traversal)
- Insufficient authorization checks of user administration bypassed
- Potential back doors
- Possible attacks using Web technologies
- Further checks

Injection attacks

- התקפות "הזרקה" מתייחסות למגוון רחב של מתקפות – SQL Injection, Command Injections...
- במתקפות מסוג זה התוקף מספק לתוכנית קלט "לא תמים". קלט זה מעובד בחלק מפקודה או שליפה ומשנה את אופן התנהגות התוכנית.
- מתקפות מסוג זה הן מהוותיקות ביותר ומאפשרות לתוקף לגנוב, לעדכן ולהשמיד מידע. כמובן שבעזרת פעולות אלו התוקף יכול גם לייצר ולמצוא דלתות אחוריות נוספות למערכת.
- ההגנה כנגד מתקפות כאלו היא יחסית פשוטה – וידוא הקלט המוזן ע"י המשתמש.
- נשתמש בעיקר במחלקה CL_ABAP_DYN_PRG כדי לוודא את תקינות הקלט.

Open SQL Injection

Dynamic Where

- השימוש ב Where דינאמי בשליפה, חושף את השליפה להזרקת תנאים נוספים בעזרת OR ובכך מאפשר להגדיל את כמות הנתונים הנשלפים וחוזרים לתוכנית באופן לא מבוקר.
- הפתרון האידיאלי הוא להפחית כמה שניתן את השימוש בWhere דינאמי והיכן שלא ניתן לוותר על כך נבצע בדיקות מקיפות על הקלט המועבר לWhere.

- ללא קשר לפיתוח מאובטח, כדאי להימנע מקוד דינאמי ע"י מידול בOOP ובעזרת הפעלת תנאים בשליפה לפי משתנה חיצוני (New syntax 7.4) לצורך קריאות הקוד.

New syntax

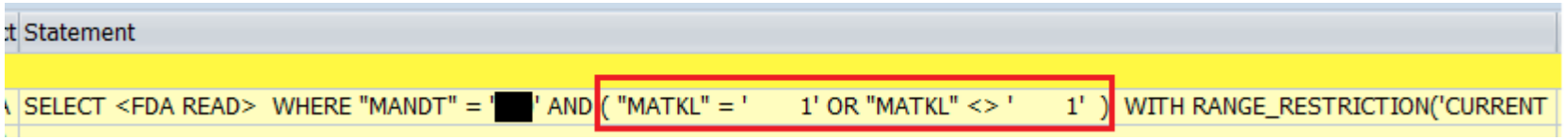
```
PARAMETERS: p_op1 TYPE abap_bool RADIOBUTTON GROUP rg1,
            p_op2 TYPE abap_bool RADIOBUTTON GROUP rg1.

START-OF-SELECTION.

  SELECT *
    FROM ekpo
    WHERE ( @p_op1 = @abap_true AND ebelp = 10 ) OR
          ( @p_op2 = @abap_true AND ebelp = 20 )
    INTO TABLE @DATA(lt_data).
```

Dynamic Where

- הסיכון ב Token דינאמי הוא הוספת תנאי OR לשליפה שיוביל לשליפת נתונים לא רצויים.

```
PARAMETERS: p_input TYPE char30.

START-OF-SELECTION.

  DATA(lv_unsecured_where) = |MATKL = | && p_input.

  SELECT *
    FROM mara
    WHERE (lv_unsecured_where)
    INTO TABLE @DATA(lt_mara).
```

- מה יקרה אם המשתמש יזין למשתנה קלט את הערך `1 OR MATKL <> 1` ?

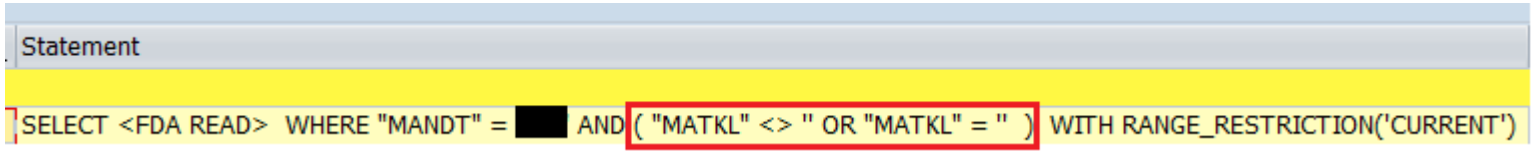


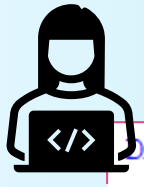
Dynamic Where

- באופן דומה, יש לנהוג בזהירות גם כאשר שם השדה דינאמי ומגיע מקלט מהמשתמש:

```
PARAMETERS: p_column TYPE c LENGTH 30,
            p_value  TYPE c LENGTH 30.


START-OF-SELECTION.

  DATA(lv_unsecured_where) = p_column &&   ` = @p_value`.

  SELECT *
         FROM mara
         WHERE (lv_unsecured_where)
         INTO TABLE @DATA(lt_mara).
```

- מה יקרה אם המשתמש יזין למשתנה קלט את הערך `MATKL <> @p_value OR MATKL` ?





Dynamic Where

- ישנם כמה דרכים להתגונן מפני מתקפה כזו:
  - שימוש בשם המשתנה ולא לשרשר את הערך שלו
  - עטיפת הערך בגרשיים
  - בדיקה האם שם העמודה קיים

```
DATA(lv_secured_where) = `MATKL = @p_input`.

SELECT *
    FROM mara
    WHERE (lv_secured_where)
    INTO TABLE @DATA(lt_mara2).
```

```
DATA(lv_secured_input) = cl_abap_dyn_prg=>quote( p_input ).
DATA(lv_secured_where2) = `MATKL = @lv_secured_input`.

SELECT *
    FROM mara
    WHERE (lv_secured_where2)
    INTO TABLE @DATA(lt_mara3).
```

```
AT SELECTION-SCREEN.
  TRY.
      cl_abap_dyn_prg=>check_column_name( p_column ).
    CATCH cx_abap_invalid_name.
      MESSAGE 'Not allowed' TYPE 'E'.
  ENDTRY.

START-OF-SELECTION.

  DATA(lv_secured_where) = p_column &&  ` = @p_value`.

  TRY.
      SELECT *
             FROM mara
             WHERE (lv_secured_where)
             INTO TABLE @DATA(lt_mara).
      cl_demo_output=>display( lt_mara ).
    CATCH cx_sy_dynamic_osql_error.
      MESSAGE `Wrong WHERE condition!` TYPE 'I'.
  ENDTRY.
```

SET Clause

- תוקף פוטנציאלי עלול לעשות שימוש ב- SET דינאמי על מנת להזריק קוד שישנה ערכים באופן לא צפוי.
- גם כאן – נעדיף תמיד פקודות סטטיות על פני דינאמיות. במידה ואין ברירה אלא להשתמש בפקודה דינאמית – נבדוק היטב את הקלט לפני ביצוע הפקודה.

SET Clause

- בדוגמה הזו אנחנו בונים את השדות לעדכון בשורה ספציפית:

```
DATA(lv_unsecured_set) = COND string( WHEN p_name IS NOT INITIAL
                                      THEN ` NAME = ` && p_name && ',' ) &&
                         COND string( WHEN p_street IS NOT INITIAL
                                       THEN ` STREET = ` && p_street && ',' ) &&
                         COND string( WHEN p_city IS NOT INITIAL
                                       THEN ` CITY = ` && p_city && ',' ) &&
                         COND string( WHEN p_postcd IS NOT INITIAL
                                      THEN ` POSTCODE = ` && p_postcd && ',' ).

DATA(lv_len) = strlen( lv_unsecured_set ).
lv_unsecured_set = substring( val = lv_unsecured_set len = lv_len - 1 ).

UPDATE scustom SET (lv_unsecured_set) WHERE id = @p_id.
```

- מה יקרה אם המשתמש יזין לשדה p_city את הערך `'', DISCOUNT = '90'` ?

Statement

UPDATE SET "CITY" = '', DISCOUNT" = '090' WHERE "MANDT" = '[redacted]AND "ID" = '00000001' WITH RANGE_RESTRICTION('CURRENT')

SET Clause

- גם כאן הדרך להתגוננות היא בעזרת עיטוף כל ערך בגרשיים, כך שהוא נבדק כערך ממש ולא מתורגם לקוד.

```
DATA(lv_secured_set) = COND string( WHEN p_name IS NOT INITIAL
                                    THEN ` NAME = ` && cl_abap_dyn_prg=>quote( p_name ) && ',' ) &&
                       COND string( WHEN p_street IS NOT INITIAL
                                    THEN ` STREET = ` && cl_abap_dyn_prg=>quote( p_street ) && ',' ) &&
                       COND string( WHEN p_city IS NOT INITIAL
                                    THEN ` CITY = ` && cl_abap_dyn_prg=>quote( p_city ) && ',' ) &&
                       COND string( WHEN p_postcd IS NOT INITIAL
                                    THEN ` POSTCODE = ` && cl_abap_dyn_prg=>quote( p_postcd ) && ',' ).

lv_len = strlen( lv_secured_set ).
lv_secured_set = substring( val = lv_secured_set len = lv_len - 1 ).

UPDATE scustom SET (lv_secured_set) WHERE id = @p_id.
```

- תוצאה:

Cust. No.	Name	Title	Street	PO Box	Post. code	City	Ctry	Region	Tel.	B/P cust.	Discount	Language	E-Mail	Name
1	Danielle					", DISCOUNT = '90'				B	90			

Dynamic Table Name

- שימוש דינאמי בשם הטבלה לשליפה/לעדכון הוא מסוכן מכיוון שכך התוקף יכול להגיע לכל טבלה אחרת במערכת.
- דוגמה נפוצה לשימוש במתקפה מסוג זה היא השימוש בשליפה על מנת להשיג את המשתמשים החזקים במערכת.
- במקרה הזה נרצה לוודא ששם הטבלה שקיבלנו מהמשתמש הוא נכון ואכן רלוונטי לתהליך.
- ניתן לבדוק זאת בשתי דרכים:
  - בדיקה שהטבלה המבוקשת נמצאת בPackage שהוגדר כרלוונטי לתהליך
  - בדיקה שהטבלה המבוקשת נמצאת בWhitelist של טבלאות מותרות

Dynamic Table Name

- ניתן לבדוק שהטבלה רלוונטית לתהליך בשתי דרכים:

```
DATA(lv_table) = cl_abap_dyn_prg=>check_table_name_str(
                   EXPORTING
                     val               = p_table
                     packages          = 'MG, ME'
                   incl_sub_packages = abap_false
                   bypass_buffer     = abap_false
                 ).
```

- בדיקה שהטבלה המבוקשת נמצאת בPackage שהוגדר ברלוונטי לתהליך

```
lv_table = cl_abap_dyn_prg=>check_whitelist_str(
  EXPORTING
    val       =  p_table
    whitelist = 'MARA, MARC'
).
```

- בדיקה שהטבלה המבוקשת נמצאת בWhitelist של טבלאות מותרות

oneXacTech

Dynamic Columns Names

- כאשר אנחנו משתמשים בשמות עמודות דינאמית, אנחנו חשופים למתקפה בעזרתה התוקף יוכל לשלוף מידע שאנחנו לא מעוניינים שיצא.
- גם כאשר משתמשים בפקודה `INTO CORRESPONDING FIELDS` אנחנו עדיין חשופים למתקפה מפני שהתוקף יכול לשנות את שם העמודה בשליפה (col1 `AS` col2).

- גם כאן נבדוק שהעמודה המתבקשת לשליפה נמצאת בWhitelist מוגדר בעזרת המתודה `check_whitelist_str` או `check_whitelist_tab`.

Native SQL Injection

Native SQL Injection

- בניגוד לשימוש ב-Dynamic Open SQL, השימוש ב-Native SQL מעביר פקודות שלמות בטקסט לבסיס הנתונים ללא שום בדיקה בשרת האפליקציה, דבר שמגדיל משמעותית את הסיכון.

- הסיכונים בSQL Injection בעת שימוש בNative SQL:
  - אחזור או שינוי מידע במערכת.
  - הרצת פקודות DDL המשנות מבני נתונים במערכת.
  - הפעלת APIs של בסיס הנתונים – לדוגמא יצירת משתמש חדש, איפוס סיסמא וכו'.

- ההמלצה היא לא לעשות שימוש כלל ב-Native SQL. במידה ואין ברירה, נבצע בדיקה על כל קלט שהמשתמש הזין וצריך לבצע בו שימוש בשליפה.

Native SQL Injection

- במידה וכן משתמשים בNative SQL, יש לבצע את השליפה בעזרת המחלקה **CL_SQL_STATEMENT**
- נשתמש ב-? כהכנה למשתנים דינאמיים היכן שנדרש ונקבע את ערך המשתנים בעזרת המתודה SET_PARAM.

- גם כאן נבצע בדיקות על הקלט בעזרת המחלקה **CL_ABAP_DYN_PRG.**

Native SQL Injection

```
* create the query string
  CONCATENATE
    'select * from' c_tabname 'where COL1 <= ? and COL2 >= ?'
    INTO l_stmt SEPARATED BY space.                          "

* create a statement object
  l_stmt_ref = p_con_ref->create_statement( ).

* bind input variables
  GET REFERENCE OF l_col1 INTO l_dref.
  l_stmt_ref->set_param( l_dref ).
  GET REFERENCE OF l_col2 INTO l_dref.
  l_stmt_ref->set_param( l_dref ).
```

ABAP Command Injections

- ניתן לג'נרט קוד ABAP בזמן ריצה באמצעות:
  - שימוש בפקודה INSERT REPORT
  - שימוש בפקודה GENERATE SUBROUTINE POOL
  - שימוש בפונקציה RFC_ABAP_INSTALL_AND_RUN – (תלוי גרסה) יצירת קוד והפעלתו במערכת מרוחקת.

- שימוש בקוד ABAP המג'ונרט באופן דינאמי עלול לאפשר לתוקף להריץ קוד זדוני ולכן יש להימנע מכך.
- אם בכל זאת יש צורך בכך – יש לבדוק היטב את הקלט בעזרת מתודות הוולידציה במחלקה CL_ABAP_DYN_PRG.
- במידה והקלט הוא תכנית שלמה ולא שורה אחת בתבנית, בדיקות הוולידציה מורכבות. במצב כזה נוכל לבדוק הרשאות פיתוח והאם המערכת הינה סביבת פיתוח או ייצור.

ABAP Command Injections

- במצב כזה המשתמש יכול לבצע איזה פעולה שהוא רוצה במערכת:

```
TYPES prog TYPE TABLE OF string WITH EMPTY KEY.
DATA: lv_text TYPE string.

CALL FUNCTION 'DEMO_INPUT_TEXT'
  CHANGING
    text_string = lv_Text
  EXCEPTIONS
    canceled    = 4.
IF sy-subrc = 4.
  LEAVE PROGRAM.
ENDIF.
SPLIT lv_text AT |\n| INTO TABLE DATA(prog).

GENERATE SUBROUTINE POOL prog NAME DATA(pool).
IF sy-subrc = 0.
  PERFORM do_it IN PROGRAM (pool).
ENDIF.
```

Injections of operating system commands

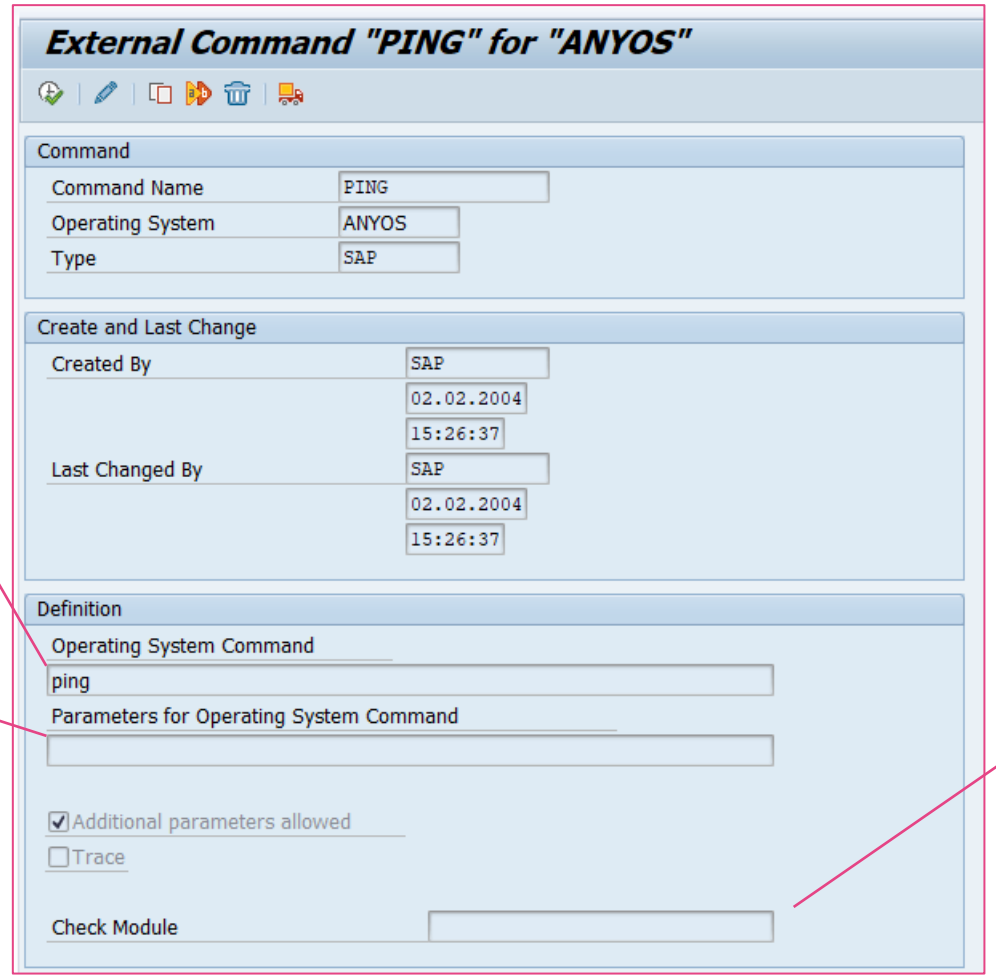
```
OPEN DATASET dest … FILTER opcom
```

- בגישה לקבצים במערכת ההפעלה לא קיים סיכון להזרקת קוד אלא אם עושים שימוש ב-Filter בחלק מפקודת ה`OPEN DATASET`. ההרחבה `FILTER` נועדה כדי להעביר למערכת ההפעלה פקודות.
- השימוש ב-Filter גורם ל-SAP לשרשר ב-PIPE את הערך שמועבר בפקודת ה-Filter למערכת ההפעלה, ללא בקרה.
- תוקף יכול לנצל פרצה זו ולהזריק פקודות נוספות למערכת ההפעלה וכך לשנות את התנהגות השרת באופן לא צפוי.
- לכן יש להימנע מהשימוש ב`FILTER` ובמקום זאת להשתמש בטרנזקציה SM69.

Injections of operating system commands

- בטרנזקציה SM69 נגדיר פעולות למערכת ההפעלה באופן בטוח.
- נעדיף ליצור פעולה בלי פרמטרים כדי לשמור על המערכת מאובטחת ככל הניתן.
- במידה וחייבים להשתמש בפרמטרים נקשר אל הפעולה check module המבוסס על הפונקציה SXPG_DUMMY_COMMAND_CHECK ונממש בדיקות שונות על הקלט בעזרת המחלקה CL_ABAP_DYN_PRG.
  באופן הזה נוודא שהבדיקות על הקלט מתבצעות לא תלות במי מריץ ומאיפה מריץ.
- נפעיל את הפקודה בעזרת הפונקציה SXPG_COMMAND_EXECUTE.

Injections of operating system commands



פקודה למערכת הפעלה

פרמטרים לפקודה

פונקציה לבדיקת קלט

XacTech

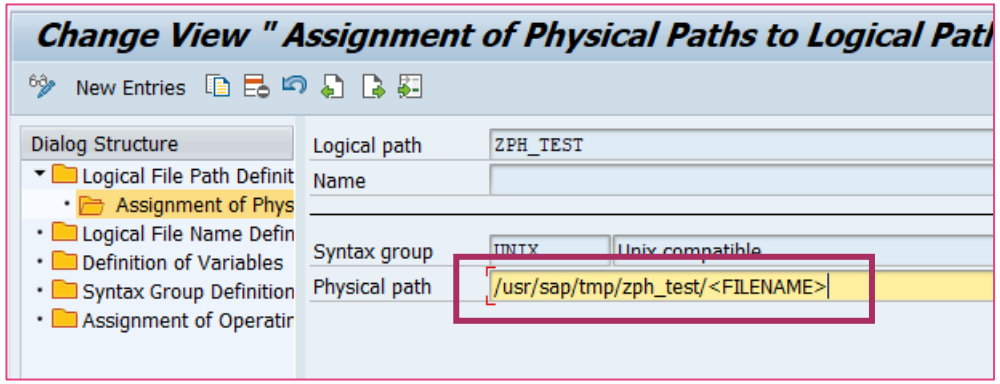
Directory Traversal

- סכנה נוספת שנרצה להגן מפניה היא גישה לא מורשת לקבצים או תיקיות במערכת ההפעלה ע"י מניפולציה על שם הקובץ.
- סכנה זו קיימת בעת שימוש בפקודות:
  - `OPEN DATASET`
  - `DELETE DATASET`
  - CL_ABAP_FILE_UTILITIES=>CREATE_UTF8_FILE_WITH_BOM

- כדי להגן מפני גישה לא מורשת:
  - נשתמש בLogical file אותו נגדיר בטרנזקציה FILE.
  - נבדוק את הנתיב שהמשתמש בחר מול whitelist
  - נבדוק את שם הקובץ בעזרת הפונקציה FILE_VALIDATE_NAME

Directory Traversal

- דוגמא מתוך FILE:



Potential back doors

שימוש בשמות משתמשים Hard Coded

- אין לעשות שימוש בשמות משתמשים בתוך הקוד.
- שימוש בשמות משתמשים בקוד לעיתים משמש תוקפים פוטנציאלים. לדוגמא תוקף יכול להסיק לאיזה משתמש אמורה להיות הרשאה מסוימת ולנסות להשתמש במשתמש הזה על מנת לבצע פעולה רגישה.
- אם שם המשתמש הנוכחי נדרש בתוכנית, נשתמש במתודה GET_USER_NAME של המחלקה CL_ABAP_SYST שכן היא מאובטחת יותר מאשר לקיחת הערך משדה המערכת sy-uname שקל יותר לתמרון.

שימוש בשמות שרתים Hard Coded

- אין לעשות שימוש בשמות שרתים בתוך הקוד.
- שימוש בשמות של שרתים בתוך הקוד עלול לעזור לתוקף להשיג מידע שאינו מורשה לו.
- 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) / מספר ה-Client Hard Coded

- אין לעשות שימוש בשמות מערכות או מספר הClient בתוך הקוד.
- גם כאן שימוש Hard-Coded בתוך הקוד בדרך כלל יכול לעזור לתוקף להשיג מידע שאינו מורשה לו.
- ניתן לבדוק האם שימוש נכון בקסטומיזציה או API כלשהו יכול להחזיר את שם המערכת או מספר הclient (למשל שימוש במשתנים בטרנזקציה FILE).

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. הסוגייה 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.

Insufficient authorization checks

Authority check

- כלל האצבע אומר שלפני גישה לנתונים באופן ישיר שלא דרך BAPI, נדרש לבדוק האם למשתמש יש הרשאה לביצוע הפעולה.
- הבדיקה כמובן תתבצע בעזרת הפקודה `AUTHORITY-CHECK` כאשר מיד לאחריה נבדוק את תוצאת ה- SY-SUBRC ונפעל בהתאם.

- בנוסף, ניתן להשתמש באחת מהפונקציות הבאות לצורך הבדיקה:
  - AUTHORITY_CHECK
  - AUTHORITY_CHECK_TCODE
  - SU_RAUTH_CHECK_FOR_USER
  - VIEW_AUTHORITY_CHECK
  - FILE_VALIDATE_NAME
  - AUTHORITY_CHECK_DATASET
  - AUTHORITY_CHECK_RFC

Insufficient authorization checks

```
AUTHORITY-CHECK OBJECT 'S_DEVELOP'
  ID 'DEVCLASS' FIELD '$TMP'
  ID 'OBJTYPE'  FIELD 'PROG'
  ID 'OBJNAME'  DUMMY
  ID 'P_GROUP'  DUMMY
  ID 'ACTVT'    FIELD '02'.
IF sy-subrc <> 0.
  LEAVE PROGRAM.
ENDIF.
```

- דוגמא לבדיקת הרשאה מפורשת בה נבדק האם למשתמש הנוכחי יש הרשאה ליצור תוכניות זמניות.
- ניתן להשתמש בבדיקה זו כדי לאבטח מפני ABAP Command Injections.

Call transaction authority check

- באשר 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.

Possible attacks using Web technologies

Cross-site scripting (XSS)

- בדומה לABAP Command Injection גם בניית HTML דינאמי עלול לסכן את המשתמש.
- 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 ל-HTML הדינאמי.
- נבחן את האופציה הטובה ביותר לעשות זאת לפי סוג האפליקציה.

Cross-site scripting (XSS)

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

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

  - גלובלי זה מבצע ENCODE לכל מחרוזת שמתקבלת
  - ניתן ללמוד יותר בNote מספר 887168.

- אפליקציות Web אחרות :
  - נבדוק את הקלט באמצעות הפונקציה escape (מורידה תווים בהתאם לפורמט המתקבל).

```
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)
```

Cross-site scripting (XSS)

- בגרסאות נמוכות יותר מ7.3 ניתן להשתמש במתודות מתאימות במחלקה CL_ABAP_DYN_PRG

- שימו לב שהשימוש במתודות Escape ב-Classes: CL_HTTP_UTILITY, CL_HTTP_SERVER ו-CL_HTTP_CLIENTלא מאושרות יותר לשימוש מכיוון שהן לא עומדות בתקני אבטחת המידע OWASP.

Redirect

- שימוש במידע חיצוני כקלט עבור ביצוע Redirect ו/או שינוי ערכי ה-Header של בקשת ה-HTTP עלול לסכן את המשתמש בביצוע Redirect לאתר אחר ללא ידיעתו.
- לדוגמא: ההמלצה היא לא להשתמש בקלט חיצוני לביצוע פעולות אלו, במידה וקיימת דרישה בזאת יש לבדוק את הקלט.

```
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 ).
```

- לשימוש ב-Redirect, 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.

Further checks

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 וכיו"ב.

הפקודה COMMUNICATION

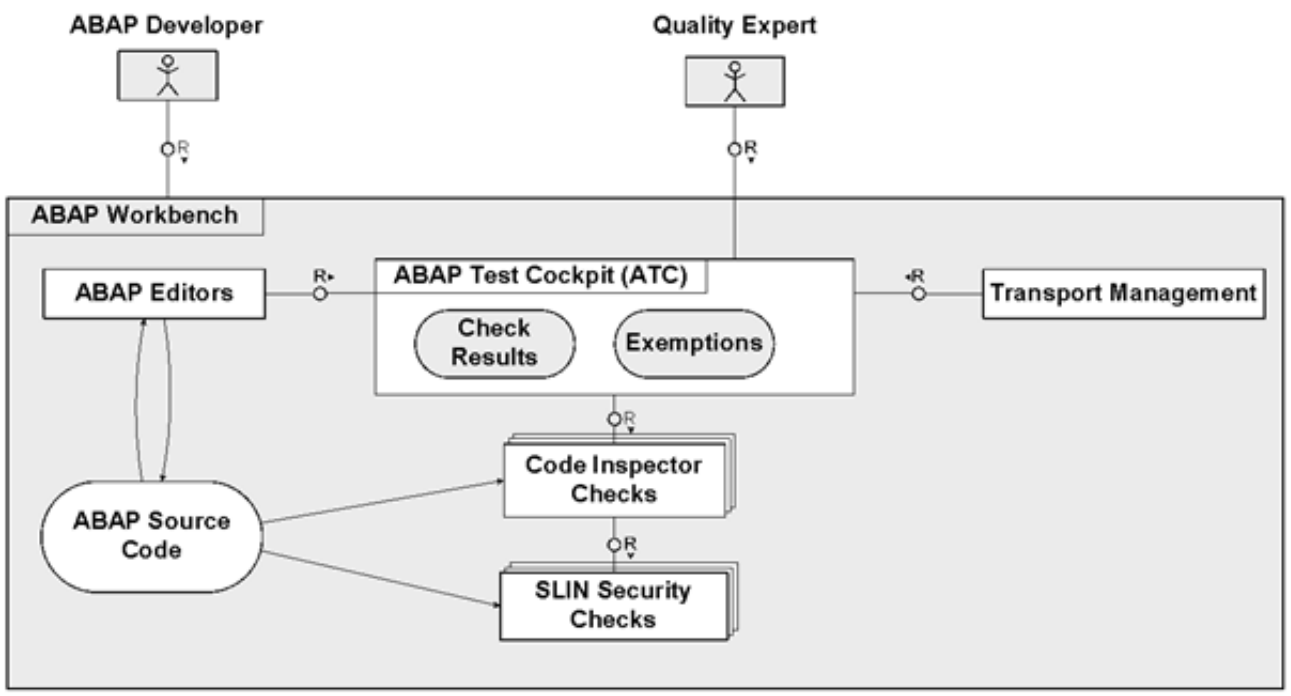
- השימוש בפקודה COMMUNICATION לא מאושר יותר לשימוש על-ידי SAP בשל בעיות האבטחה הקיימות בפקודה.
- היום יש להשתמש בפונקציות RFC.

ABAP Test Cockpit (ATC)

מה זה ABAP Test Cockpit (ATC)?

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

ארכיטקטורה



- כאשר בסיום הפיתוח 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" מכיוון שאין פתרון אחר - לדוגמא שגיאה בקוד מג'ונרט.

בדיקות ATC

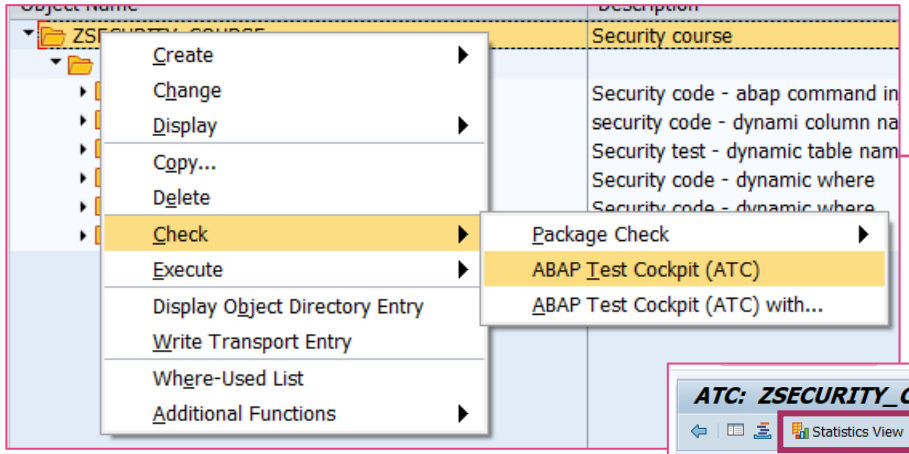
- ATC מריץ את ה Code Inspector, SLIN ומחלקות בדיקות בZיות במידה וקיימות.
- ניתן להריץ את הבדיקות באופן ישיר מתוך ABAP Source Code (se38, se24, se80…)
- ניתן להפעיל את הATC באופן אוטומטי בכל פעם שמשוחרר Task/Transport ולקבוע האם השגיאות עוצרות את תהליך השחרור.
- ניתן לתזמן ג'וב תקופתי המריץ בדיקות על המערכת.

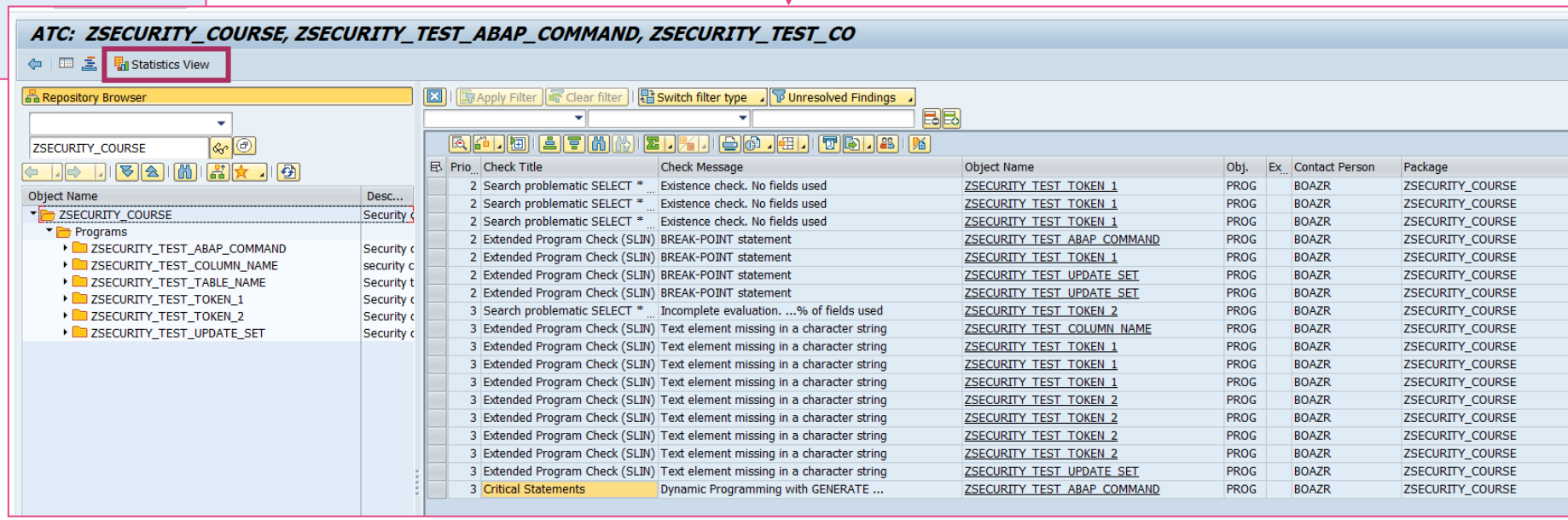
הפעלת הATC

- טרנזקציה SCI – Check Variant –
  - נשכפל את הואריאנט הסטנדרטי DEFAULT ונפעיל את בדיקות האבטחה
  - ניתן לשנות סוגי ההודעות שיתקבלו על כל בדיקה

- הגדרת הATC –טרנזקציה ATC
  - Basic Settings –
    - נשנה את הואריאנט הנבדק להיות הואריאנט החדש
    - נאפשר Exemptions
  - Exemptions
    - בMaster system (בכל הנראה QA) את רשימת המאשרים

הרצת הATC





one XacTech

הרצת ATC

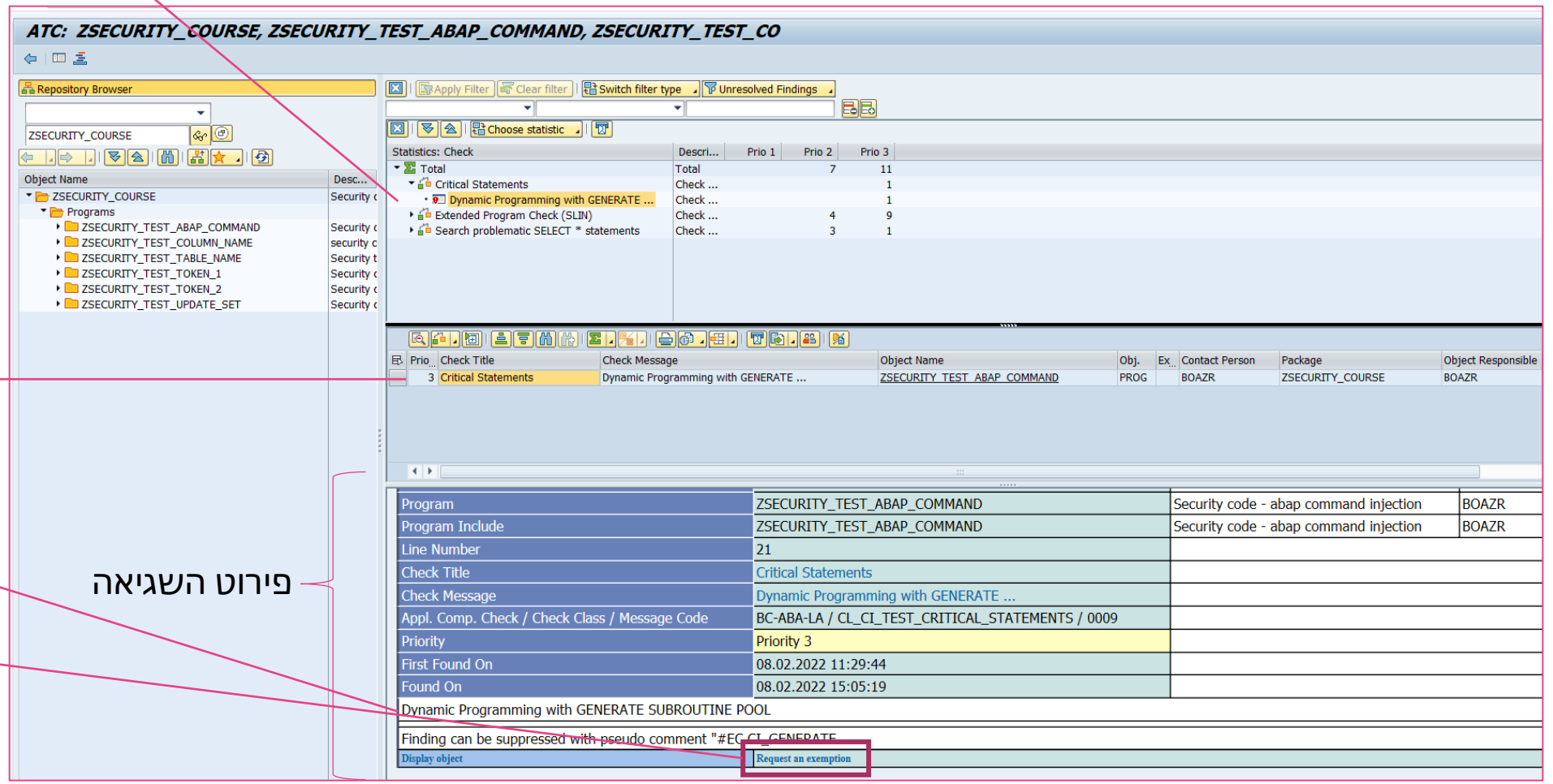
סוגי הבדיקות שנפלו –
דאבל קליק מציג את הטבלה עם השגיאות

טבלת השגיאות –
דאבל קליק מעביר
לפירוט השגיאה

Pragma לאישור השגיאה

פירוט השגיאה

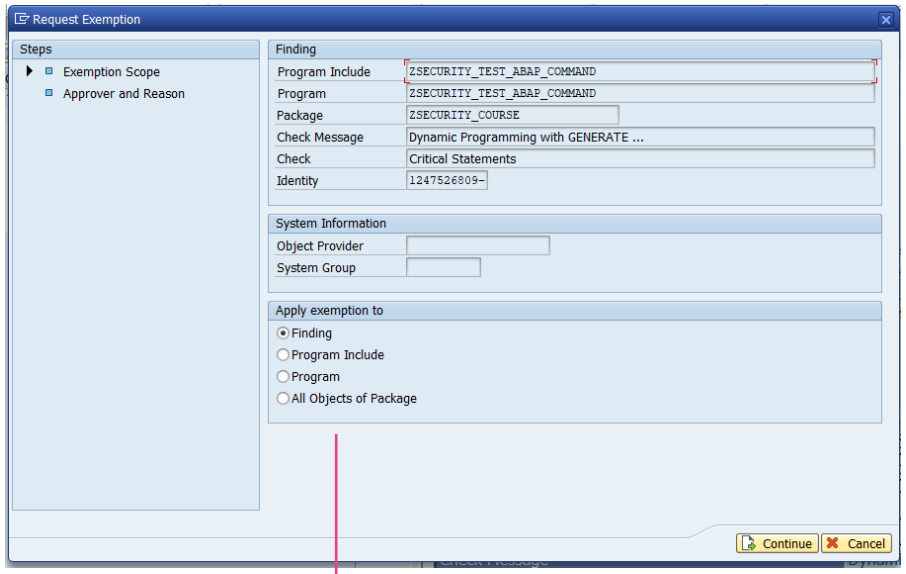
בקשת "פטור"



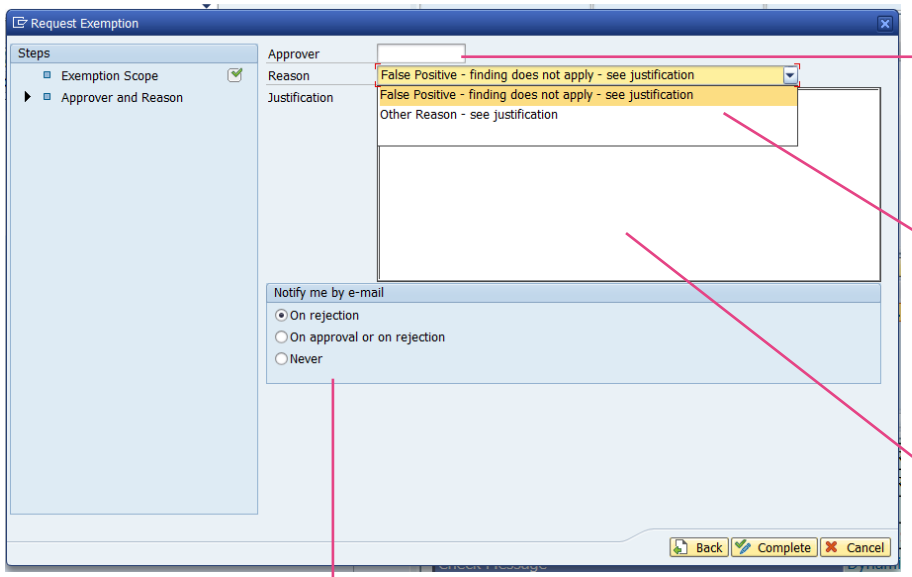
one XacTech

בקשת "פטור"

- לאחר הרצת בדיקות הATC, מפתח מטפל בכל השגיאות שהוא יכול.
- כאשר ישנה הודעה שאינה מוצדקת בעיניו או אינה ניתנת לטיפול, הוא יכול להוסיף בקוד פראגמה (#EC CI_GENERATE) או להגיש בקשה ל"פטור" מאותה בדיקה או באופן כללי מכל הATC.



היקף הפטור הנדרש



מתי ליידע אותי

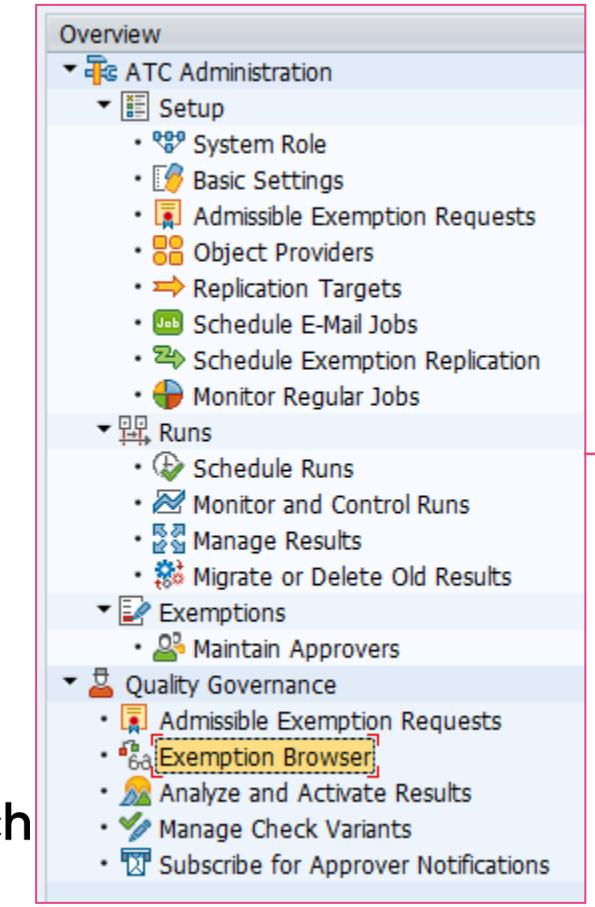
בחירה מרשימת המאשרים שהגדרנו – מגיש הבקשה לא יכול להיות המאשר

סיבה לבקשה – טעות בזיהוי או סיבה מוצדקת אחרת

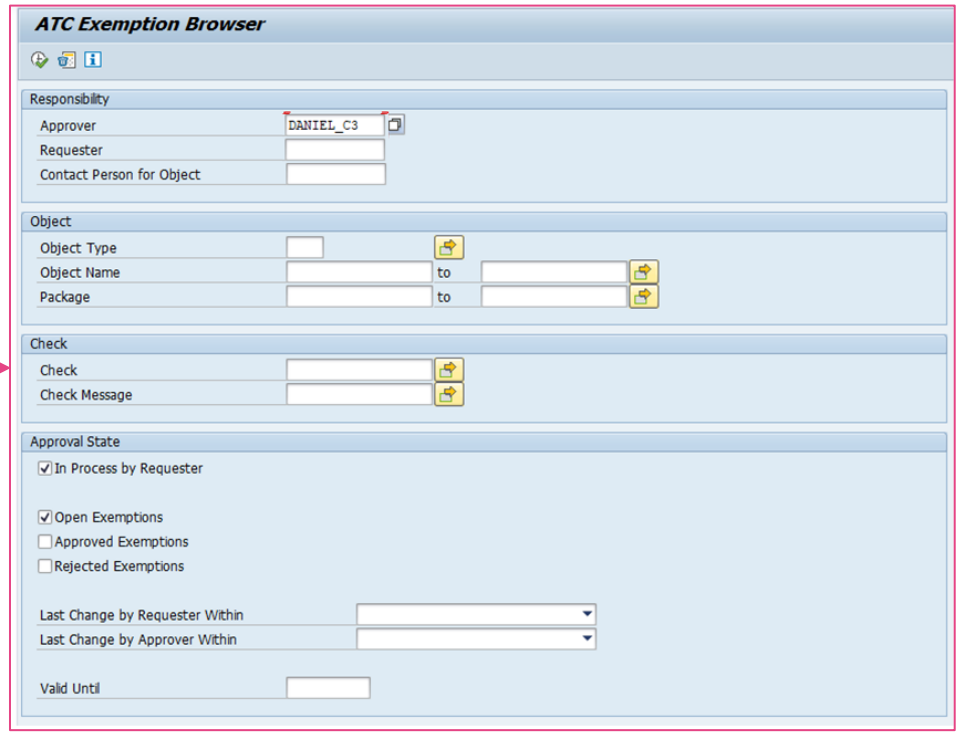
פירוט

אישור בקשת פטור

ATC



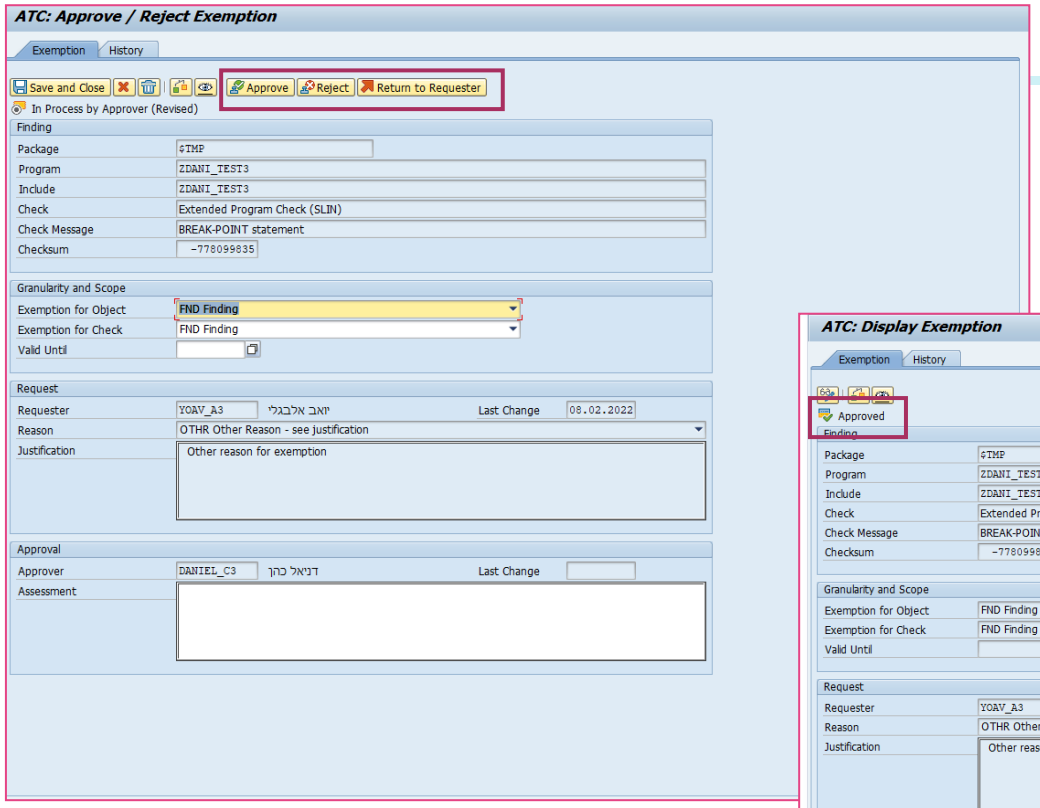
דו"ח בקשות

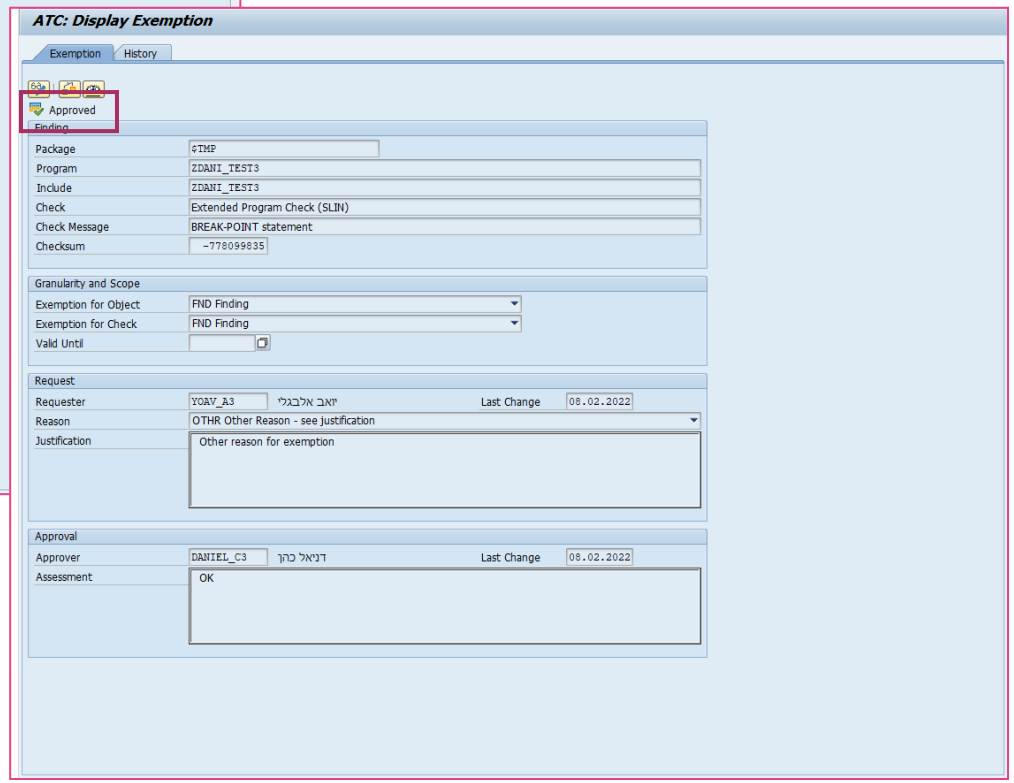


one XacTech

אישור בקשת פטור

- ניתן לאשר, לדחות או להחזיר למבקש
- ניתן לכתוב הערה





one XacTech

להעמקת הידע

- Code Vulnerability Analyzer Checks
- ABAP Test Cockpit (ATC)- Your Way to Secure ABAP Code

תודה רבה

דניאל כהן