Modernize Mainframe Security Admin From USS Using Python: Sailing to Clear Waters

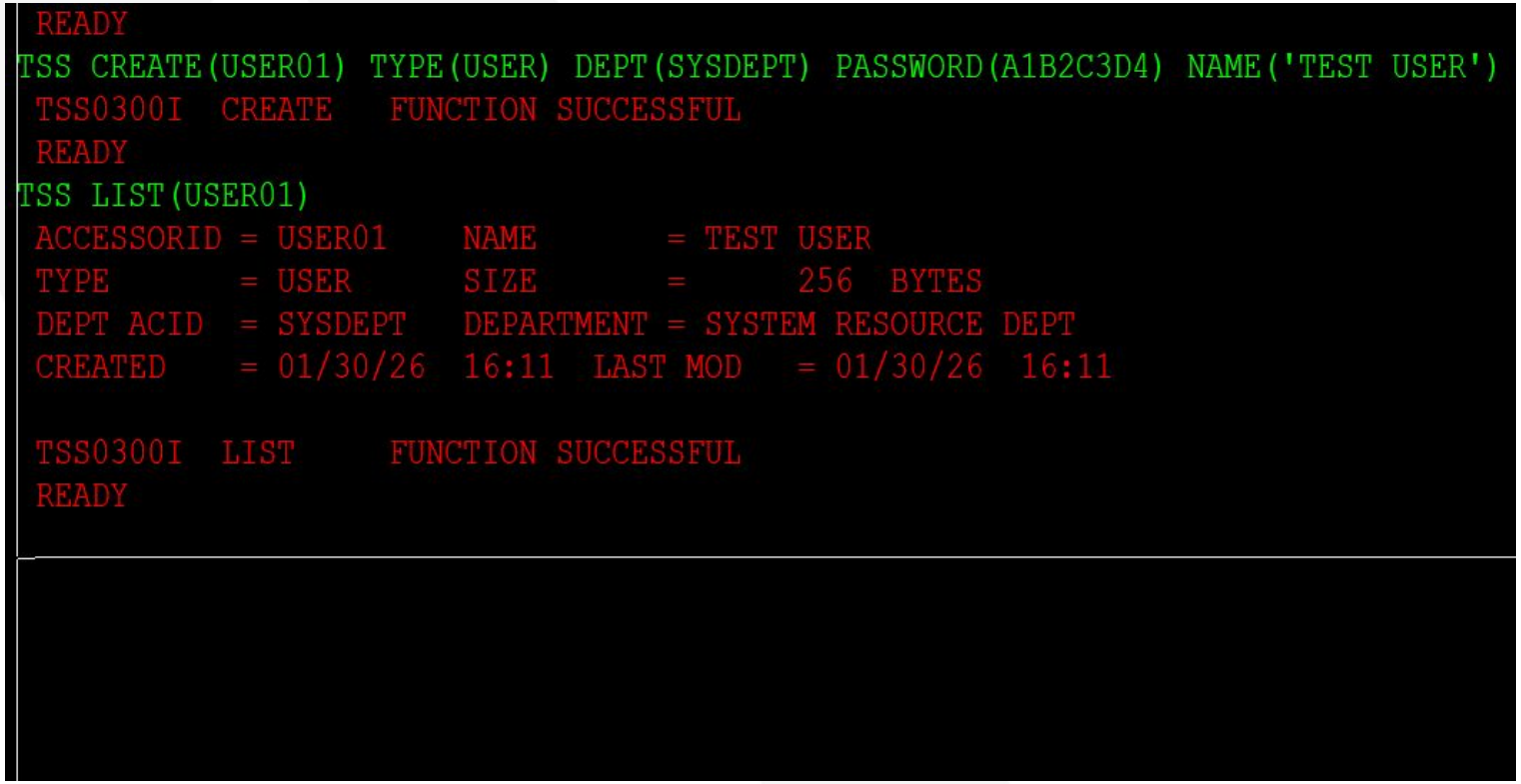
Udatta Bhattacharya (Udi)
Software Engineer | Cyber Security and Compliance
Broadcom Mainframe Software
udatta.bhattacharya@broadcom.com

The Dual Environments of z/OS

- **MVS (Multiple Virtual Storage):**
  - The traditional "heart" of the mainframe.
  - Home to **RACF, ACF2, and Top Secret** (the External Security Managers).
  - Driven by JCL, TSO commands, and ISPF panels.
- **USS (Unix System Services):**
  - A certified Unix environment running natively on z/OS.
  - Provides a hierarchical file system and support for modern languages like **Python**.
- **The Administrator's Bridge:**
  - Modern Security Administration is no longer restricted to the "Green Screen."
  - **USS** acts as a powerful staging ground to automate **MVS** security tasks.
  - Administrators can leverage Unix-style scripting to manage mainframe security identities and audit logs.

ESMs and their Command Interfaces

- **Primary Toolsets**: Administration for RACF, ACF2, and Top Secret is largely driven by TSO commands.
- **Traditional Workflow**: Administrators typically operate via 3270 "Green Screen" sessions.
- **Direct Interaction**: Commands are issued manually by the TSO user at the READY prompt.



Command Interfaces - Beyond Manual Entry

```
//PROD001T JOB  1,'USER SETUP',CLASS=A,
//         MSGCLASS=X,TIME=1440,NOTIFY=&SYSUID
//*
//STEP010  EXEC PGM=IKJEFT01
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
TSS CREATE(USER01) TYPE(USER) -
    DEPT(SYSDEPT)             -
    PASSWORD(A1B2C3D4)        -
    NAME('TEST USER')
/*
//STEP010  EXEC PGM=IKJEFT01
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
TSS LIST(USER01)
/*
```

- **Batch Job Execution**: Utilizes utility programs to run command lists in the background.
  - **IKJEFT01**: Standard utility for RACF and Top Secret.
  - **ACFBATCH**: Dedicated utility for ACF2.
- **Scripting for Efficiency**: CLIST and REXX are used to automate command generation.
  - Reduces manual entry and minimizes human error.
- **Programmatic Execution**: Commands can be triggered via application code.
  - Often requires execution within an authorized environment for elevated privileges. More on this later.

Command Interfaces - Nuances

ACF2

- Command issued under ACF subcommands mode.
- PGM=ACFBATCH needs to be called under batch to execute a list of commands.
- In REXX, ACF2 subcommands need to be queued up before issuing the command “ACF” under “ADDRESS TSO”
- From USS and TSO, the acfunix utility can be used to issue ACF2 commands.

Top Secret

- Command issued from TSO
- PGM=IKJEFT01 can be used to batch-submit multiple commands.
- Commands issued under “ADDRESS TSO” are sufficient in REXX environment.

RACF

- Command issued from TSO
- PGM=IKJEFT01 can be used to batch-submit multiple commands.
- Commands issued under “ADDRESS TSO” are sufficient in REXX environment.

Programmatic Interfaces

The `r_admin` Programmatic Interface

- **Unified SAF Interface**: r_admin serves as a SAF callable service for executing security functions programmatically.
- **Language Support**: Accessible via low-level environments including Assembler (HLASM) and Metal C.
- **Command Execution**: Uses the specific function **`ADMN_RUN_COMD`** to trigger ESM commands.
- **ESM Agnostic**: The interface remains identical whether running on RACF, ACF2, or Top Secret, ensuring cross-platform consistency.
- **Modern Accessibility**:
  - Can be invoked directly from **TSO** via the CALL command.
  - Accessible through **Unix System Services (USS)**.
  - Supports modern automation via **Python** integration.

Programmatic Interfaces - r_admin C Interface

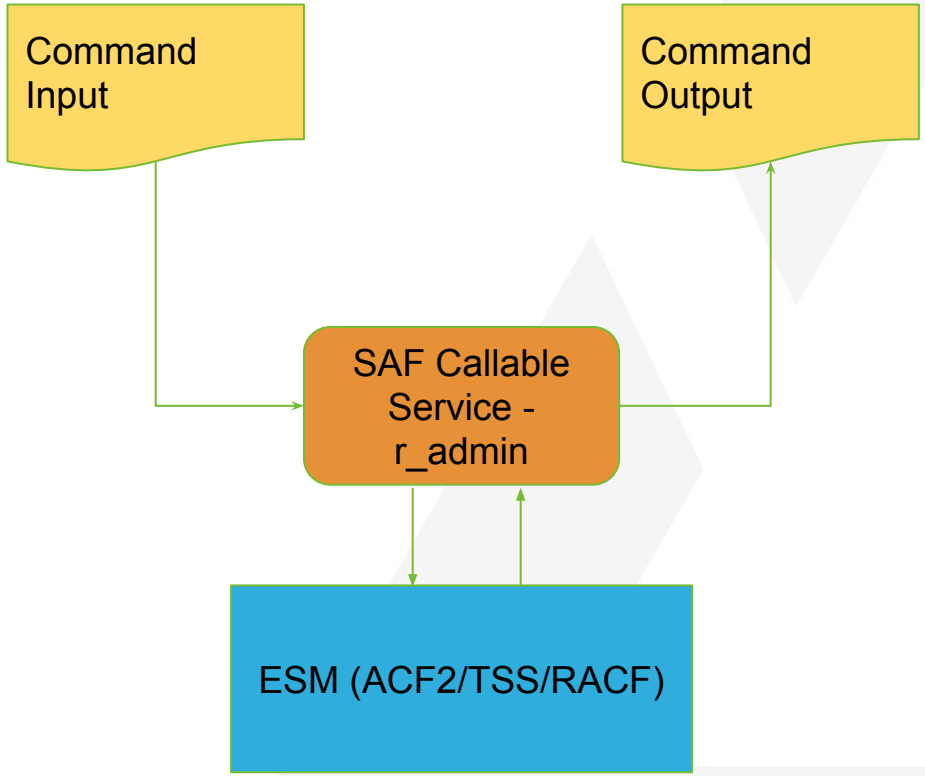
```c
/**
    call_r_admin
    @brief Calls the SAF callable service r_admin
    @param[in,out] plist A pointer to the Radmin parameter list structure.
    @returns SAF Return code from r_admin
*/
static int call_r_admin(struct Radmin* plist) {

    IRRSEQ00 (
        plist->work,
        &plist->alet, &plist->safrc,
        &plist->alet, &plist->racfrc,
        &plist->alet, &plist->racfrsn,
        &plist->function,
        plist->cmd,
        plist->userid_buf,
        &plist->acee,
        &plist->subpool,
        &plist->output
    );

    return plist->safrc;

}
```

Programmatic Interfaces - r_admin C Interface



- **The Interface Stub**: Uses **IRRSEQ00** as the primary stub to facilitate direct communication between C programs and the **`r_admin`** service.
- **Capturing ESM Outputs**:
  - **Assembler/Metal C**: Captures system messages by opening a Data Control Block (DCB) and routing them to an output DD.
  - **LE-C (Language Environment)**: Simplifies the process by piping messages directly to STDOUT using standard printf functions.
- **Python Integration**: Outputs captured via Unix System Services (USS) can be processed and leveraged by Python for advanced reporting or automation.

Programmatic Interfaces - Python

- **Platform Availability**: Python is natively available and supported on Unix System Services (USS).
- **IBM z Open Automation Utilities (ZOAU)**:
  - Can be installed on z/OS 2.5 or later.
  - Provides a bridge between modern scripting and traditional mainframe functions.
- **Extended Capabilities**: Enables interaction with MVS programs, operator commands, and comprehensive dataset management.
- **Execution Flexibility**:
  - Supports both authorized and unauthorized program execution.
  - Runs modules directly from **PDS/PDSE load libraries** resident on z/OS.

Invocation Medium - JCL

- **JCL Fundamentals**: Job Control Language (JCL) is used to define and execute batch jobs for program invocation.
- **Minimal Setup Requirements**:
  - **Program Name**: Specific module to be executed.
  - **Program Library**: The PDS/PDSE where the load module resides.
  - **Arguments**: Parameters passed directly to the program via the PARM field.
  - **Output Library**: Destination for results (Sequential dataset or PDS/PDSE member).
- **Error Handling**: Accesses program-returned Return Codes (RC) as JCL Condition Codes to control job flow.
- **Job Submission**: Files are submitted to a batch processor, such as JES2, for scheduling and execution.

Invocation Medium - JCL Components

JCL to execute a program:

```
000001 //PROD001A JOB  1,'CALL SAFPTKT',CLASS=A,MSGCLASS=X,
000002 //          TIME=1440,NOTIFY=&SYSUID
000003 //*
000004 //STEP010  EXEC PGM=SAFPTKT,PARM='USER(PROD001),APPL(DB2TOOLS)'
000005 //STEPLIB  DD   DSN=CASAF.PUBLIC.LOADAPF,DISP=SHR
000006 //SYSOUT   DD   SYSOUT=*
```

Name of Utility or Program to be executed.

A library (PDS/PDSE) containing the executable load module of the program.

Input argument string

Invocation Medium - JCL Output

JCL to execute a program:

```
000001 //PROD001A JOB  1,'CALL SAFPTKT',CLASS=A,MSGCLASS=X,
000002 //         TIME=1440,NOTIFY=&SYSUID
000003 //*
000004 //STEP010  EXEC PGM=SAFPTKT,PARM='USER(PROD001),APPL(DB2TOOLS)'
000005 //STEPLIB  DD   DSN=CASAF.PUBLIC.LOADAPF,DISP=SHR
000006 //SYSOUT   DD   SYSOUT=*
```

SYSOUT DD Contents after execution:

```
****** ********************************************************** Top of Data ***********
000001 Generated passticket <9XOKLUOK> for USERID <PROD001> and APPL <DB2TOOLS>
****** ********************************************************* Bottom of Data *********
```

Invocation Medium - ZOAU commands on USS

- **MVS Program Execution**: ZOAU provides native Unix commands to bridge the gap between USS and MVS environments.
- **Execution Types**:
  - **mvscmd**: Used for executing unauthorized programs.
  - **mvscmdauth**: Required for executing authorized programs (APF-authorized).
- **Additional Utility Commands**:
  - **opercmd**: Issue system operator commands.
  - **apfadm**: Manage Authorized Program Facility (APF) lists.
  - **zinfo**: Retrieve system and hardware information.
  - **dgrep**: Search for strings within datasets.

Invocation Medium - ZOAU Utility Command

```
PROD001@B054:~$ mvscmdauth --pgm=SAFPTKT --steplib=CASAF.PUBLIC.LOADAPF --args='USER(PROD001),APPL(DB2TOOLS)' --sysout=* -J
{
        "data": {
                "program": "SAFPTKT",
                "STEPLIB": "CASAF.PUBLIC.LOADAPF",
                "DDs": [
                        {
                                "type": "console",
                                "ddname": "SYSOUT",
                                "name": "PROD001.P3554460.T0626151.C0000000",
                                "retained": false,
                                "content": "Generated passticket <Z4IFW8EJ> for USERID <PROD001> and APPL <DB2TOOLS>\n",
                                "content_length": 74
                        }
                ]
        },
        "program": "mvscmdauth",
        "options": "--pgm=SAFPTKT --args=USER(PROD001),APPL(DB2TOOLS) --sysout=* -J ",
        "rc": "0"
}
PROD001@B054:~$
```

Invocation Medium - Python Script on USS

- **Parallels to JCL**: Python scripts utilizing ZOAU function similarly to a JCL job stream, defining logical execution steps.
- **Defining "Job Steps"**: ZOAU APIs allow developers to wrap MVS program execution within a Python script running on USS.
- **Consistent Requirements**: Requires the same core parameters as JCL:
  - **Program Name**
  - **STEPLIB** (Load Libraries)
  - **DD Statements** (Data Definitions)
  - **PARMS** (Execution Arguments)
- **Flexible Output Handling**:
  - Results can be printed directly to **STDOUT**.
  - Enables real-time **parsing and data processing** within the Python environment for enhanced automation.

Invocation Medium - Python Script using ZOAU

```python
from zoautil_py.mvscmd import execute_authorized
from zoautil_py.ztypes import DDStatement, ZOAUResponse

def call_safptkt():
    """
    Calls SAFPTKT and prints the output.
    """
    # Define DD Concatenations
    dd_statements: list[DDStatement] = []
    dd_statements.append(DDStatement(name="STEPLIB", definition='CASAF.PUBLIC.LOADAPF'))
    dd_statements.append(DDStatement(name="SYSOUT", definition="*"))
    arg_string: str = 'USER(PROD001),APPL(DB2TOOLS)'

    # Call Program with the parms
    resp: ZOAUResponse = execute_authorized(pgm='SAFPTKT', pgm_args=arg_string, dds=dd_statements)

    # Get outputs
    print(resp.stdout_response)

if __name__ == "__main__":
    call_safptkt()
```

Invocation Medium - Pythonize Your Commands

SHARE
EDUCATE • NETWORK • INFLUENCE

```python
if __name__ == "__main__":

    # Read arguments
    parser = argparse.ArgumentParser(description="Call r_admin to run a command on the ESM")
    parser.add_argument("-c", "--cmd", required=True, help="The ESM Command.")
    args = parser.parse_args()

    # Define DD Concatenations
    dd_statements: list[DDStatement] = []
    loadlib: DatasetDefinition = DatasetDefinition("CASAF.PUBLIC.LOADAPF")
    dd_statements.append(DDStatement(name="STEPLIB", definition=loadlib))
    dd_statements.append(DDStatement(name="SYSOUT", definition='*'))

    # Call Program with the parms
    resp: ZOAUResponse = execute_authorized(pgm='ESMCMD', pgm_args=args.cmd, dds=dd_statements)

    # Get outputs
    if resp.stdout_response != '': print(resp.stdout_response)
    if resp.stderr_response != '': print(resp.stderr_response)
```

Tying it all together - Run an ESM Command

```
PROD001@B054:~/mcltest/esmcmd$ python3 ./esmcmd.py --cmd 'TSS LIST(PROD006)'
ACCESSORID = PROD006    NAME        = TSO USER
TYPE       = USER       SIZE        =     1536  BYTES
FACILITY   = BATCH
   ADMIN BY= BY(LANRO03 )    SMFID(GOLD)   ON(08/11/2015)  AT(12:34:07)
FACILITY   = STC
   ADMIN BY= BY(LANRO03 )    SMFID(GOLD)   ON(08/11/2015)  AT(12:34:07)
FACILITY   = TSO
   ADMIN BY= BY(LANRO03 )    SMFID(GOLD)   ON(08/11/2015)  AT(12:34:07)
FACILITY   = UNICNTR
   ADMIN BY= BY(MASTER  )    SMFID(MVSX)   ON(01/16/2024)  AT(18:35:53)
DEPT ACID  = TSODEPT1  DEPARTMENT = TSO DEPARTMENT
CREATED    = 08/11/15  12:34  LAST MOD    = 11/24/25  22:10
PROFILES   = IZUUSER   PRODUSER
GROUPS     = CFZUSRGP  OMVSGRP   USERGRP1
LAST USED  = 11/24/25 22:08 CPU(B054) FAC(TSO     ) COUNT(00005)
DFLTGRP    = USERGRP1

TSS0300I  LIST     FUNCTION SUCCESSFUL
```

Auditing ESM Events

Auditor’s Friend - SMF Reporting

- **Event Journaling:** Both **ACF2** and **Top Secret** record critical security events into **SMF (System Management Facilities)** records.
- **Monitored Activities:**
  - Dataset access (loggings and violations).
  - Program usage and unauthorized access attempts.
  - Security database updates and administrative changes.
  - **OMVS (Unix)** Callable service traces.
- **Dedicated Reporting Utilities:**
  - **ACF2:** Uses **ACFRPTOM** for OMVS event tracing.
  - **Top Secret:** Uses **TSSOERPT** for OMVS event tracing.
- **Workflow Evolution:**
  - **Legacy:** Auditors rely on **JCL** or **ISPF panels** to generate manual reports.
  - **Automation:** Utilities can be integrated into **CLIST** or **REXX** EXECs.
  - **Modern: Python** can now be used to trigger and process these reporting utilities for streamlined auditing.

SMF Reporting - Using JCL

Case: Running TSSOERPT for OMVS Tracing

- **Utility Characteristics: TSSOERPT** is treated as a standard executable program for generating Top Secret OMVS Callable Service traces.
- **Library Management: STEPLIB** is unnecessary, as the utility usually resides within the system's **linklist concatenation**.
- **Input Configuration:**
  - **SMF Datasets:** Allocated to the job using specific **DD statements**.
  - **SYSIN DD:** Contains the control parameters and filters for the report.
- **Output Handling:**
  - **SYSPRINT DD:** Captures the generated SMF report for review or processing.

SMF Reporting - A JCL Example

```
****** ********************************** Top of Data **********************************
000001 //PROD001A JOB  1,'TSSOERPT',CLASS=A,MSGCLASS=X,
000002 //         TIME=1440,NOTIFY=&SYSUID
000003 //ACFRPTOM EXEC PGM=TSSOERPT
000004 //SYSUDUMP DD   SYSOUT=*
000005 //RECMAN1  DD   DISP=SHR,DSN=SYS1.MAN1
000006 //RECMAN2  DD   DISP=SHR,DSN=SYS1.MAN2
000007 //RECMAN3  DD   DISP=SHR,DSN=SYS1.MAN3
000008 //SYSPRINT DD   SYSOUT=*
000009 //SYSIN    DD   *
000010 SERVICE(R_admin)
000011 DETAIL
000012 //*
****** ******************************** Bottom of Data ********************************
```

SMF Reporting - JCL Components

```
****** ********************************* Top of Data *****
000001 //PROD001A JOB  1,'TSSOERPT',CLASS=A,MSGCLASS=X,
000002 //         TIME=1440,NOTIFY=&SYSUID
000003 //ACFRPTOM EXEC PGM=TSSOERPT
000004 //SYSUDUMP DD   SYSOUT=*
000005 //RECMAN1  DD   DISP=SHR,DSN=SYS1.MAN1
000006 //RECMAN2  DD   DISP=SHR,DSN=SYS1.MAN2
000007 //RECMAN3  DD   DISP=SHR,DSN=SYS1.MAN3
000008 //SYSPRINT DD   SYSOUT=*
000009 //SYSIN    DD   *
000010 SERVICE(R_admin)
000011 DETAIL
000012 //*
****** ********************************* Bottom of Data ***
```

Name of Utility or Program to be executed.

DDs allocated to SMF datasets to be read as input by the utility.

DD allocated to an input dataset/file/instream-data containing parameters for the utility.

SMF Reporting - The Python Equivalent

```python
def call_tssoerpt():
    """
    Calls tssoerpt and prints the output.
    """

    sysin_file = NamedTemporaryFile(mode='w+b', delete=True)

    sysin_writer = TextIOWrapper(sysin_file, encoding='cp1047')
    sysin_writer.write("SERVICE(R_admin)\n")
    sysin_writer.write("DETAIL\n")
    sysin_writer.flush()

    # Define DD Concatenations
    dd_statements: list[DDStatement] = []
    dd_statements.append(DDStatement(name="RECMAN1", definition='SYS1.MAN1'))
    dd_statements.append(DDStatement(name="RECMAN2", definition='SYS1.MAN2'))
    dd_statements.append(DDStatement(name="RECMAN3", definition='SYS1.MAN3'))
    dd_statements.append(DDStatement(name="SYSPRINT", definition='*'))
    dd_statements.append(DDStatement(name="SYSIN", definition=FileDefinition(sysin_file.name)))

    # Call Program with the parms
    resp: ZOAUResponse = execute(pgm='TSSOERPT', dds=dd_statements)

    # Get outputs
    print(resp.stdout_response)
    sysin_writer.close()
```

Tying it all together - Run an SMF Report on USS

```
PROD001@B054:~/zoau-esm/src$ python3 ./om_report.py
1Mainframe Security          - z/OS USS Event Log                    - PAGE    1
 DATE 01/28/26 (26.028) TIME 18.21

      SERVICE       USERID    GROUP         UID          GID     SAF      RC     RSN
        DATE           TIME     JOBNAME    SOURCE    SYSID    CPU    SECLABEL

   R_admin           PROD001  USERGRP1         1026            4    0       0       0
   01/28/26  26.028    18.21.43 PROD0011                      B054
   Successful - Logging active by Trace/Audit options
```

Bonus - ACF2 Productivity Tools

Tools for productivity on ACF2, and more:
https://github.com/BroadcomMFD/broadcom-product-scripts/tree/main/cybersecurity-scripts/ACF2

- **List Utility -** LI
  Executes the **LIST** subcommand. Moves output to a temporary dataset for interactive scrolling (PF7/8) and searching.
- **System Info -** SHOW
  Processes **SHOW** subcommand arguments into a reviewable format, bypassing the need for batch report submissions.
- **Access Audit -** AC
  Captures **ACCESS** subcommand results interactively to analyze rule permissions without manual screen-scrolling.
- **Universal Utility -** ACFB / ACFE
  Browse (ACFB) or **Edit (ACFE)** mode for *any* ACF2 command. Provides a full ISPF-like environment for command output.

Resources

- Get python on USS - https://www.ibm.com/docs/en/python-zos/3.14.0
- Get ZOAU on USS - https://www.ibm.com/products/z-open-automation-utilities

- Verify your python Installation on USS:
  ```
  python3 --version
  ```

- Verify your ZOAU Installation on USS:
  ```
  zoauversion
  ```

SHARE
EDUCATE • NETWORK • INFLUENCE

Happy Coding!

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation

Join a Mainframe Technical Exchange

Apr. 21-23, 2026	June 23-25, 2026	Oct. 20-22, 2026
European MTE Prague, Czech Republic	Virtual MTE Held Virtually	North American MTE Plano, TX

- Network with peers and Mainframe technical experts
- Participate in technical how-to sessions and hands-on workshops
- No registration fee!

Questions