41

RACF

update

August 2005

In this issue

A xephon PUBLICATION

RACF Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

RACF Update on-line

Code from *RACF Update*, and complete issues in Acrobat PDF format, can be downloaded from http://www.xephon.com/racf; you will need to supply a word from the printed issue.

Subscriptions and back-issues

A year's subscription to *RACF Update* (four quarterly issues) costs $290.00 in the USA and Canada; £190.00 in the UK; £196.00 in Europe; £202.00 in Australasia and Japan; and £200.50 elsewhere. The price includes postage. Individual issues, starting with the August 2000 issue, are available separately to subscribers for $72.75 (£48.50) each including postage.

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *RACF Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Access RACF data using LDAP in just five steps

This article is aimed at beginners who wish to quickly set up an LDAP server on z/OS to make RACF data available to LDAP clients. RACF data (such as user profiles, group profiles, and resources profiles) can be added/altered from LDAP clients with the proper user credentials using the SDBM database of the LDAP server implementation on z/OS.

IMPLEMENTATION STEPS

It is assumed that RACF is already installed and up and running. Only the simplest of LDAP SDBM implementations has been considered, and the version considered is z/OS 1.6. Definitions in the steps may vary from version to version. The reader's discretion is advised while applying the steps below at their installation:

- Step 1 – copy the configuration files from the */usr/lpp/ldap/etc* directory to the */etc/ldap*.
- Step 2 – customize the LDAP server configuration file, *slapd.conf*, in */etc/ldap*. This file is divided into sections, such as top section, SDBM section, TDBM section, etc. For this implementation, only the top section and SDBM section are relevant and would need editing.

  A couple of definitions are required in the top section to specify on which port the LDAP server would be listening and who would be the LDAP server administrator. In the sample definition below, the LDAP server would be listening on port 389 and RACF userid LDAPADM would be the administrator:

  ```
  listen ldap://:389
  adminDN "racfid=ldapadm, profiletype=user, sysplex=mvs1"
  ```

  A couple of definitions are also required in the SDBM section to inform the LDAP server that SDBM is enabled

and to specify its base DN. In the sample definition below, the base DN is sysplex=mvs1:

```
database sdbm GLDBSDBM
suffix "sysplex=mvs1"
```

- Step 3 – create the required started PROC for the LDAP server from sample member LDAPSRV in HLQ.SGLDSAMP. A sample JCL PROC is given below:

```
//LDAPSRV  PROC REGSIZE=64M,
//  PARMS='',
//  OUTCLASS='A'
//GO       EXEC PGM=GLDSLAPD,REGION=&REGSIZE,TIME=1440,
//         PARM=('/&PARMS >DD:SLAPDOUT 2>&1')
//STEPLIB  DD DISP=SHR,DSN=SYS1.SIEALNKE
//SLAPDOUT DD SYSOUT=&OUTCLASS
//SYSOUT   DD SYSOUT=&OUTCLASS
//SYSUDUMP DD SYSOUT=&OUTCLASS
//CEEDUMP  DD SYSOUT=&OUTCLASS
```

- Step 4 – configure RACF for the LDAP server. A RACF userid would be required to run the LDAP server. In the sample below, the RACF group LDAP and userid LDAPSRV are created:

```
LDAP SUPGROUP(SYS1) OMVS(GID(5))
ADDUSER LDAPSRV NOPASSWORD DFLTGRP(LDAPGRP)
                                  OMVS(UID(5) PROGRAM ('/bin/sh'))
```

The userid that is going to run the LDAP server as a started task should be permitted in the BPX.SERVER facility class:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(LDAPSRV) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

For the start-up procedure, the started class profile should also be defined as below:

```
DEFINE STARTED LDAPSRV.** STDATA(USER(LDAPSRV))
SETROPTS RACLIST(STARTED) REFRESH
```

- Step 5 – now LDAPSRV is ready to start. Issue the command **START LDAPSRV** to start the LDAP server.

Arun Kumaar R
MVS Systems Programmer (India)

RACF in focus – finding redundant RACF groups

This is a regular column focusing on specific aspects of RACF. In this issue, we will discuss the problem of redundant RACF groups in the RACF database, and look at a programmed approach to addressing this issue.

Most RACF databases have all kinds of redundancies. These crop up as a result of the daily RACF administrative work, and there is not much we can do about it except find ways (hopefully, automated) to address the issue.

Keeping your RACF database free from redundancies is in your own best interest. Not only will it satisfy the auditors, it will also simplify and reduce the daily RACF administrative workload. If it is not done regularly, the problem will grow and become unmanageable in the future.

What redundancies? Just to mention a few... there are the userids of users who have left the company; then there are resource profiles that no longer make sense; you may even have CICS transaction profiles for transactions that don't exist in CICS any more; and dataset profiles that were created at some time in the past for datasets that no longer exist; and most installations will have redundant RACF groups.

In this article we will look at the last issue, that of redundant RACF groups. These RACF groups were created at one time for a reason, but may no longer be useful, or even valid.

DEFINITION OF A REDUNDANT RACF GROUP

How can we define a redundant RACF group? We can safely say that a RACF group is redundant if *all* these conditions are met:

1 *It has no users connected to it*. The normal RACF administrative process of connecting and removing userids from groups can sometimes leave a RACF group with no users connected.

2 *It has no sub-groups*. Of course, if a RACF group *has* sub-groups then it is not redundant; even though it may not have users connected to it, it cannot be deleted.

3 *It is not a dataset High-Level Qualifier (HLQ).* If it is, then it may meet the two conditions above, but it is still not redundant, because in RACF it is a requirement that all dataset high-level qualifiers be defined as RACF groups.

Of course, even when all these conditions are met, a RACF group may still not be redundant. We can never say for sure, because it could be a newly-defined group, and user connections may have been planned for the future. But if we can somehow list all groups in the RACF database that meet the above criteria, we can then do a further manual check to make sure the groups are indeed redundant before deleting them.

AUTOMATED SOLUTION

Proposed below is an automated method that will identify all RACF groups meeting the three criteria listed above. This process can be run once a month to:

1 Identify all potentially redundant RACF groups.

2 Review manually the output list to verify that the groups are indeed redundant.

3 Execute the **delete** commands generated by the REXX routine to remove the redundant groups.

You will find that you will catch many redundant groups the first time you run this process. After that, on an on-going basis, you will find only a few.

The solution requires a REXX routine to be run in batch mode. The REXX routine uses RACF database unload records. It requires as its input sorted lists of all type 100, 101, and 102 records from the RACF database. You should already have these datasets created nightly. If not, your systems programmers can help you to create them.

Presented below is the JCL needed to run the REXX routine, followed by the REXX routine itself.

BATCH JCL

```
//REDGRP    JOB ( …),'YOUR NAME', MSGCLASS=X,CLASS=X,NOTIFY=&SYSUID
//STEPØ1    EXEC PGM=IKJEFTØ1,REGION=2M
//SYSTSPRT  DD SYSOUT=*
//INDSNHLQ  DD  DSN=HLQ.LIST,DISP=SHR
//INDBU1ØØ  DD  DSN=REC.TYPE1ØØ,DISP=SHR
//INDBU1Ø1  DD  DSN=REC.TYPE1Ø1,DISP=SHR
//INDBU1Ø2  DD  DSN=REC.TYPE1Ø2,DISP=SHR
//OUTGROUP  DD  DSN=OUTPUT.FILE.PDS(JUNØ5),DISP=SHR
//SYSTSIN   DD   *
REDUNDGP
/*
```

Input files:

- INDSNHLQ – this DDname points to the file containing a *sorted* list of all valid dataset high-level qualifiers at your installation. The dataset high-level qualifiers start at column 15.
- INDBU100 – this DDname points to the *sorted* file containing all type 100 records from the RACF unloaded database.
- INDBU101 – this DDname points to the *sorted* file containing all type 101 records from the RACF unloaded database.
- INDBU102 – this DDname points to the *sorted* file containing all type 102 records from the RACF unloaded database.

Output file:

- OUTGROUP – this DDname points to the output file that will contain the list of possible redundant groups. The dataset is a PDS, with a member name reflecting the month on which the list was produced.

THE REXX ROUTINE

This REXX routine (REDUNDGP) generates a list of potentially redundant groups in dataset OUTPUT.FILE.PDS(JUN05). The list is as follows:

```
DELGRX GROUP1
DELGRX GROUP88
…
etc.
```

The program deliberately produces DELGRX commands instead of the correct spelling DELGRP. This allows you time to review the list, and verify that the group names are indeed redundant before submitting them for deletion. It also prevents accidental execution of the list.

Once you are satisfied that you are ready to remove the redundant groups, change all DELGRX to DELGRP in a TSO edit session.

```
/*    REXX                                                    */
/************************************************************/
/* NAME: REDUNDGP                                           */
/* PURPOSE: THIS REXX WILL REPORT ON GROUPS THAT:           */
/* 1  HAVE NO USERS CONNECTED, AND                          */
/* 2  HAVE NO SUB-GROUPS                                    */
/************************************************************/
"EXECIO  *  DISKR  INDSNHLQ (STEM  INHLQ.  FINIS)";
"EXECIO  *  DISKR  INDBU100 (STEM  IN100.  FINIS)";
"EXECIO  *  DISKR  INDBU101 (STEM  IN101.  FINIS)";
"EXECIO  *  DISKR  INDBU102 (STEM  IN102.  FINIS)";
DO I =1 TO INHLQ.0
       PARSE VAR INHLQ.I  JUNK1  15  HLQGRP.I  23  JUNK2
END

DO I =1 TO IN100.0
       PARSE VAR  IN100.I  JUNK1  6  GRP100.I  14 JUNK2
END

DO I =1 TO IN101.0
       PARSE VAR  IN101.I  JUNK1  6  GRP101.I  15  ID101  23  JUNK2
END

DO I= 1 TO IN102.0
       PARSE VAR  IN102.I  JUNK1  6  GRP102.I  15  ID102  23 JUNK2
END
```

```
DO J = 1 TO  IN1ØØ.Ø
       DSNHLQ =  'NO'
       DO K = 1 TO INHLQ.Ø
              IF GRP1ØØ.J = fòLQGRP.K  THEN DO
                   DSNHLQ =   'YES'
                   K = INHLQ.Ø
              END

        END

        IF DSNHLQ =  'NO'  THEN DO
            DO L = 1 TO  IN1Ø2.Ø
                  CONNECT =  'NO'
                  IF GRP1ØØ.J  =  GRP1Ø2.L  THEN DO
                      CONNECT =  'YES'
                      L =  IN1Ø2.Ø
                  END

             END

      IF CONNECT =  'NO'  THEN DO
           DO M = 1  TO  IN1Ø1.Ø
                  SUBGRP =   'NO'
                   IF GRP1ØØ.J  =  GRP1Ø1.M  THEN DO
                        SUBGRP =  'YES'
                        M =  IN1Ø1.Ø
                   END

           END

      IF SUBGRP =  'NO'  THEN DO
          QUEUE 'DELGRX' GRP1ØØ.J
      END

      END
END
END
IF QUEUED()  =  Ø  THEN DO
    QUEUE 'NO GROUPS TO REPORT'
END
"EXECIO  *  DISKW  OUTGROUP (FINIS)";
EXIT
```

IN CONCLUSION

Removing redundant RACF groups is a useful exercise, necessitated by the fact that daily RACF administration will inevitably produce redundant groups.

Periodically running the REXX routine shown above is the answer to automatically cleaning these redundant groups.

This process can be placed in your RACF 'tool-kit' along with other aids to clean up other aspects of RACF redundancies.

Dinesh Dattani would welcome feedback, comments, and queries about this column. He can be contacted at dinesh123@rogers.com.

Dinesh Dattani
Mainframe Security Consultant
Toronto (Canada)

ICSF events reporting

INTRODUCTION

As the connectivity of computer networks and the quantity as well as the value of information processed by systems increases, concerns have grown about the threat of disclosure or modification of sensitive data, either accidentally or intentionally. In addition to that, because of the pervasive use of personal identification numbers at automated teller machines and point-of-sale terminals, and the increasing use of electronic funds transfer among banks and wholesale institutions, the financial industry has become more security conscious and has started to demand high-performance and high-security computer systems to support many types of financial transaction. As a result, the efficient economical protection of enterprise-critical information has become increasingly important in many diverse application environments.

Therefore it was not a surprise that computer users have demanded high-speed cryptographic functions for bulk encryption to provide network and database security. The

protection required to conduct commerce on the Internet, provide data confidentiality, and authenticate individuals can be provided only by cryptographic services and techniques. Cryptography is an effective method of protecting information while it is being transmitted through a communication link or while it is stored in a medium vulnerable to unauthorized access. Cryptographic operations can also be used for processing message authentication codes and personal identification numbers in a financial-transaction environment. The cryptographic feature is secure high-speed hardware that performs the actual cryptographic functions. The cryptographic feature available to your applications depends on the server or processor hardware. The cryptographic hardware currently available comes in two forms: cryptographic coprocessors and cryptographic accelerators.

PCI Cryptographic Coprocessor (PCICC) is a data encryption coprocessor card designed to provide data security functions. It is attached via an internal PCI (Peripheral Component Interconnect) bus to the system. The card operation is controlled by an on-board 486-type processor. It has special hardware for various cryptographic operations (Rivest-Shamir-Adleman (RSA) algorithm, DES, or Secure Hash Algorithm 1 (SHA-1)). Other cryptographic algorithms are carried out using the on-board processor. The card works in parallel and asynchronously to the zSeries processors. The PCICC card is aimed at high-security environments where the keys are encrypted when outside the protected card. It is mainly used for RSA public key algorithm operations including RSA key generation, special PIN operations, and special user cryptographic functions (using the User Defined Extension (UDX) capability of the card).

PCI Cryptographic Accelerator (PCICA) is a cryptographic hardware accelerator card that is attached via the PCI bus to the system. This card provides a competitive option to customers who do not need the high-security environment of the PCICC, but need high cryptographic performance for RSA public key algorithms that may be required in Web server

applications. The PCICA provides clear key RSA operations with a modulo length of 1024 bits or 2048 bits. Two algorithms are available, Modular Exponentiation (ME) and Chinese Remainder Theorem (CRT) for both key lengths. The card has five hardware engines that can work in parallel and asynchronously to the zSeries processors. There are always two cards packaged in one adapter that plugs into the system board. This allows for a scalable and cost-effective system performance in Web server applications.

The Integrated Cryptographic Service Facility (ICSF) is a software element of z/OS that works with the cryptographic hardware and the z/OS Security Server to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides many cryptographic services to protect your data. Some of these services are authentication, digital signatures, encryption, and key protection:

- Authentication is a process that calculates a Message Authentication Code (MAC) for some given data. The MAC provides a means for the receiver to verify that the data was not changed after it was authenticated. For example when a bank sends an Electronic Funds Transfer transaction to another bank, the MAC can be verified by the receiving bank to ensure that the account number and amount have not been changed in transit, whether by accident or mischief.
- Digital signatures are similar to authentication but differ from authentication in that you can guarantee the sender of the message. Algorithms supported for digital signatures are RSA and the DSS (Digital Signature Standard).
- Encryption is a process that scrambles sensitive data so unauthorized people cannot read it. Account numbers, personal medical information, and other private company or customer information are examples of typically encrypted information. Supported algorithms, in order of strength, include DES, Triple-DES, and AES.

ICSF also provides the APIs by which applications request the cryptographic services. Thus, ICSF shields the complexity of the hardware communication from the user. On the other hand, the ICSF provides administrative and application interfaces to allow the installation to generate, manage, and exchange cryptographic keys. ICSF can also help you to perform other functions related to data integrity and digital signatures. With ICSF, one can, for example, change the master encryption key and update other keys without disrupting service to applications currently using those keys.

MONITORING AND REPORTING

ICSF records certain ICSF events in the SMF dataset. ICSF also sends messages that are generated during processing to diagnostic datasets and consoles. The SMF recording and messages help you detect problems and track events. Let us now describe the events that ICSF records in the SMF record.

RMF reporting

While ICSF is running, one can use RMF and SMF to monitor certain events and operations. For example, RMF provides performance monitoring of installed cryptographic hardware in the Postprocessor Crypto Hardware Activity report. This report is based on SMF record type 70, subtype 2. In order to start gathering cryptographic hardware performance data, RMF Monitor I control statement CRYPTO should be specified. In order to evaluate the utilization of each configured crypto card, three kinds of metrics are provided: request rate (requests executed per second), exec time (the average amount of time (in ms) for a single request), and utilization % (busy time / interval time * 100). In addition to standard encrypting/ decrypting requests, key generation requests are measured separately. For PCICA cards, a drill-down into special types of operations is available with both 1024- and 2048-bit format ME and CRT. The above-mentioned report also separately provides the ICSF statistics about specific types of services.

This data is formatted on the bottom part of the report: using data encryption standard to encipher and decipher (DES), generating and verifying Message Authentication Codes (MAC), using public Hash functions (HASH) and translating and verifying Pins (PIN). See *z/OS RMF Report Analysis* (SC33-7991) for full descriptions of the fields in this report.

Starting with z/OS V1R2, RMF adds more transparency in the area of cryptographic processing by reporting Crypto Using and Delay values in the *Goals vs Actuals* section of the postprocessor WLMGL report. This was done by WLM's sampling address spaces or enclaves that are using or waiting for crypto-related hardware. In WLM goal mode, SMF type 72, subtype 3 reports on each service or report class in the active WLM policy. In the *Service/Report Class Period Data* section there are fields reporting cryptographic coprocessor usage and delays. Tasks that are found to be using or waiting for a crypto processor (synchronous or asynchronous) are reported on a service or report class period level.

There are three different types of delay/using samples:

- CAM Crypto samples (CAM) – a TCB was found executing on or waiting for a Cryptographic Asynchronous Message processor.
- Feature Queue sample (FQ) – a TCB was found executing on or waiting for a processor feature queue associated with a CPU.
- AP Crypto samples (AP) – a TCB was found executing on or waiting for a cryptographic Adjunct Processor.

While CAM and FQs are related to general purpose processors (CPs) that are assigned for crypto work, APs are related to PCI Crypto Coprocessors. Since the WLMGL report does not show the using and delay metrics on type granularity, the following overview criteria are introduced for more detailed reporting: CAPUSGP (Crypto CAP Using%), CAPDLYP (Crypto CAP Delay%), FQDLY (Crypto FQ Delay%), APUSGP (Crypto AP Using%), APDLYP (Crypto AP Delay%), CRYUSGP (Crypto Overall Using%), and CRYDLYP (Crypto Overall Delay%).

ICSF SMF records

As already stated, the main API to exploit the zSeries hardware cryptographic coprocessors is ICSF. ICSF dynamically routes the requests to crypto hardware, transparently to the application. Application designers cannot directly specify which coprocessor is to be used. ICSF makes the decision based on the required functions, which coprocessor type supports it, and performance considerations. The ICSF API is available to Assembler programs and to high-level languages as well. Note that the ICSF API is also accessible to REXX programs using a link 'stub'. The following IBM program products that call ICSF services imply the use of operational keys encrypted under a Master Key or a Key-Encrypting-Key. Therefore, they require at least one cryptographic coprocessor to be in operation for the z/OS image. In z/OS V1R4, the SSL-enabled servers (which therefore use System SSL) are IBM HTTP server, CICS Transaction System and CICS Transaction Gateway, TN3270 server, FTP server and client, LDAP server and client, WebSphere Application Server, and WebSphere MQ.

If one is eager to see who has been using ICSF services, SMF record type 30 (Common Address Space Work) field SMF30CSC in the Processor Accounting Section should be consulted because it contains the number of cryptographic instructions executed on behalf of the caller (within the caller's address space). In order to get that data, ICETOOL was used to select and display SMF type 30 records. Run this job to determine what offsets to use for the next job:

```
//*-----------------------------------------
//* OFFSET30 JCL: run this job to
//* find out what SMF record 30/4 offsets
//* to use for the next job.
//* Identification section (SMF30IOF)
//* Processor section      (SMF30COF)
//*-----------------------------------------
//TOOL1    EXEC PGM=ICETOOL
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=your.smf.weekly.ds,DISP=SHR
//SMFOFF   DD DSN=&&SMFOFF,DISP=(,PASS),SPACE=(CYL,(50,25)),
//         UNIT=SYSDA,DCB=(RECFM=VB,LRECL=32756,BLKSIZE=0)
```

```
//SRT1CNTL DD *
  OPTION DYNALLOC,VLSHRT,STOPAFT=1Ø,SPANINC=RC4
  INCLUDE COND=(6,1,BI,EQ,X'1E',AND,23,2,BI,EQ,X'ØØØ4')
  SORT FIELDS=(7,4,FI,A)
/*
//TOOLMSG DD SYSOUT=*
//REPORT  DD SYSOUT=*
//TOOLIN  DD *
  SORT FROM(RAWSMF) TO(SMFOFF) USING(SRT1)
  DISPLAY FROM(SMFOFF) LIST(REPORT)         -
  TITLE('SMF Record 3Ø/4 Offsets display')  -
  PAGE DATE TIME BLANK                      -
* Use values displayed for a, b in the next ICETOOL job
  HEADER('ID. Sec. (a)')      ON(33,4,FI)   -
  HEADER('Proc Sec.(b)')      ON(57,4,FI)
/*
//*
```

Output from this job looks like:

```
SMF Record 3Ø/4 Offsets display

ID. Sec.(a)   Proc Sec.(b)
-----------   -----------
        214           47Ø
```

Use a=214 and b=470 on the next ICETOOL job:

```
//SORT1 EXEC PGM=ICETOOL
//DFSMSG  DD SYSOUT=*
//SRT1IN  DD DSN=your.smf.weekly.ds,DISP=SHR
//SRT1INT DD DSN=&&SMF3ØOUT,DISP=(,PASS),SPACE=(CYL,(5Ø,25)),
//           UNIT=SYSDA,DCB=(RECFM=VB,LRECL=32756,BLKSIZE=Ø)
//SRT1CNTL DD *
  OPTION DYNALLOC,VLSHRT,SPANINC=RC4
* Run previous icetool job (OFFSET3Ø) to get values for a, b
* and substitute with the appropriate values
* SMF3ØLEN  (RDW)  Ø+1=1 start of record
* SMF3ØTME  (TIME) 6+1=7 start of record
* SMF3ØDTE  SMF record write date: 1Ø+1
*
* Identification Section offset (a): 214 + 1 = 215
*         off len filed
*  Ø+215: 215  8  SMF3ØJBN - Job or session name.
*  8+215: 223  8  SMF3ØPGM - Program name
* 32+215: 247  8  SMF3ØJNM - JES job identifier.
*
* Processor Accounting Section offset (b): 47Ø + 1 = 471
*         off len filed
*  4+471: 475  4  SMF3ØCPT - CPU time under the TCB
*  8+471: 479  4  SMF3ØCPS - CPU time under the SRB
```

```
* 56+471: 527  4  SMF30CSC - ICSF/MVS service count
*
   INCLUDE COND=(6,1,BI,EQ,X'1E',AND,23,2,BI,EQ,X'0004',
                 AND,527,4,BI,GT,X'00000000')
   SORT FIELDS=(11,4,FI,A,7,4,FI,A)
/*
//TOOLMSG DD SYSOUT=*
//REPORT  DD SYSOUT=*
//TOOLIN  DD *
   SORT FROM(SRT1IN) TO(SRT1INT) USING(SRT1)
   DISPLAY FROM(SRT1INT) LIST(REPORT)                    -
   TITLE('ICSF/MVS Service count')                       -
   PAGE DATE TIME BLANK                                  -
   HEADER('Date')          ON(11,4,DT1,E'9999/99/99')    -
   HEADER('Time')          ON(07,4,TM1,E'99:99:99')      -
   HEADER('JES2 #')        ON(247,8,CH)                  -
   HEADER('Job name')      ON(215,8,CH)                  -
   HEADER('Program')       ON(223,8,CH)                  -
   HEADER('ICSF count')    ON(527,4,FI)                  -
   HEADER('TCB time')      ON(475,4,FI,F1)               -
   HEADER('SRB time')      ON(479,4,FI,F1)
/*
```

This simple ICETOOL report (below) provides only a summary list of jobs/tasks using ICSF services. However, please note that each time ICSF issues a hardware instruction, this count is incremented. This means that in some cases, like bulk encryption, the count would be incremented by more than 1, and for other operations, like a PIN verification, the count would increment by 1. There is no correlation between the 'number of cryptographic instructions' and the number of service calls.

Date	Time	JES2 #	Job name	Pgm	ICSF count	TCB time	SRB time
2003/07/16	09:14:32	STC06982	CSFSTART	CSFMMAIN	3575	0.37	0.07
2003/07/28	09:17:09	STC07333	PKISERVD	IKYPKID	133	506.25	5.52
2003/07/29	12:40:57	STC07892	IMWEBSRV	IMWHTTPD	1572	69.41	5.53
2003/09/12	08:33:57	STC00256	IMWEBSRV	IMWHTTPD	1684	66.96	5.37
2003/10/05	13:38:49	STC08103	BBODMNB	BBODAEMN	9	0.53	0.03
2003/10/10	15:49:29	STC08285	BBOS001	BBOCTL	29	53.17	0.09
2003/10/12	09:39:55	STC08341	BBOS001	BBOCTL	2528	153.05	0.27
2003/10/12	10:22:39	STC08360	BBOS001	BBOCTL	987	109.73	0.12
2003/10/12	11:03:39	STC08339	IMWEBSRV	IMWHTTPD	311	74.35	5.95
2003/10/19	13:56:15	STC08419	IMWEBSRV	IMWHTTPD	2626	638.27	51.67
2003/10/19	13:56:57	STC08530	BBODMNB	BBODAEMN	9	1.09	0.22
2003/10/19	13:57:02	STC08529	BBOS001	BBOCTL	8443	261.13	0.61

ICSF uses SMF record type 82 to record certain ICSF events and operations. Record type 82 contains a fixed header section and subtypes. Each subtype contains information about the event/operation that caused ICSF to write to the SMF record:

- Subtype 1 (ICSF Initialization) – when ICSF starts, ICSF writes to subtype 1 after initialization is completed. This subtype describes the values of installation options that are specified in the installation options dataset.
- Subtype 3 (ICSF Status Change) – ICSF writes to subtype 3 when processors are verified at initialization, after a master key is set or changed, when ICSF switches from stand-by mode to normal mode, or when a processor comes on-line or off-line. When processor status changes, subtype 3 gives the status of the processors still on-line.
- Subtype 4 (Error Handling for Cryptographic Coprocessor Feature) – ICSF writes to subtype 4 when the Coprocessor is in standby mode or when the Cryptographic Coprocessor Feature detects tampering. The error conditions for cryptographic feature failure is CC3, Reason Code 1, and for cryptographic tampering is CC3 Reason Code 3.
- Subtype 5 (Special Secure Mode Change) – ICSF writes to subtype 5 when the status of the special secure mode changes. ICSF also updates subtype 5 when the Cryptographic Coprocessor Feature indicates that special secure mode was required for an instruction, but was not enabled.
- Subtype 6 (Master Key Part Entry) – ICSF writes to subtype 6 when master key parts are entered using a TKE workstation and are processed using the TKE master key entry ICSF panels.
- Subtype 7 (Operation Key Part Entry) – ICSF writes to subtype 7 when key parts are entered using a TKE workstation and are processed using the operational key entry ICSF panels.

- Subtype 8 (CKDS Refresh) – ICSF writes to subtype 8 when the in-storage CKDS is successfully refreshed. ICSF refreshes the in-storage CKDS by reading a disk copy of a CKDS into storage.
- Subtype 9 (Dynamic CKDS Update) – ICSF writes to subtype 9 when an application uses the dynamic CKDS update services to write to the CKDS.
- Subtype 10 (PKA Key Part Entry) – ICSF writes to subtype 10 when a clear key part is entered for one of the PKA master keys, ie when you use the ICSF panels to enter PKA master key parts.
- Subtype 11 (Clear New Master Key Part Entry) – ICSF writes to subtype 11 when a clear key part is entered for the DES master key, ie when you use the ICSF panels to enter new master key parts.
- Subtype 12 (PKSC Commands) – ICSF writes to subtype 12 for each request and reply from calls to the CSFSPKSC service by TKE.
- Subtype 13 (Dynamic PKDS Update) – ICSF writes to subtype 13 whenever the PKDS is updated by a dynamic PKDS update service, ie when an application uses the dynamic PKDS update services to change the PKDS.
- Subtype 14 (PCI Cryptographic Coprocessor Clear Master Key Entry) – ICSF writes to subtype 14 when a clear part is entered for any of the PCI Cryptographic Coprocessor master keys, ie whenever you use ICSF panels to update SYM-MK and ASYM-MK in the new master key register in a PCI Cryptographic Coprocessor.
- Subtype 15 (PCI Cryptographic Coprocessor Retained Key Create or Delete) – ICSF writes to subtype 15 whenever you create or delete a retained private key in a PCI Cryptographic Coprocessor.
- Subtype 16 (PCI Cryptographic Coprocessor TKE Command Request or Reply) – ICSF writes to subtype 16

whenever a TKE workstation either issues a command request to a PCI Cryptographic Coprocessor or receives a reply response from a PCI Cryptographic Coprocessor.

- Subtype 17 (PCI Cryptographic Coprocessor Timing) – ICSF periodically records processing times for PCI Cryptographic Coprocessor operations in subtype 17 to provide some indication of PCI Cryptographic Coprocessor performance.
- Subtype 18 (PCI Cryptographic Coprocessor Configuration) – ICSF writes subtype 18 when a PCI Cryptographic Coprocessor, PCI Cryptographic Accelerator, or PCI X Cryptographic Coprocessor is brought on-line or taken off-line.
- Subtype 19 (PCI X Cryptographic Coprocessor Timing) – ICSF periodically records processing times for PCI X Cryptographic Coprocessor operations (when a PCI X Cryptographic Coprocessor operation begins or ends) in subtype 17 to provide some indication of PCI X Cryptographic Coprocessor performance.

As already mentioned, the subtype 17 and 19 records have timestamps that can be used to give an indication of how a particular coprocessor is performing. By extracting the timestamps from the records and calculating the elapsed time for the card activity and the time for the data to get back to the caller's address space, a picture can be drawn of how the cards have processed over time.

I have extracted the following fields:

- SMF82CTN – the timestamp just before queueing the request to the coprocessor.
- SMF82CTD – the timestamp just after dequeueing the result from the coprocessor.
- SMF82CTW – the timestamp just after returning the result to the caller's address space

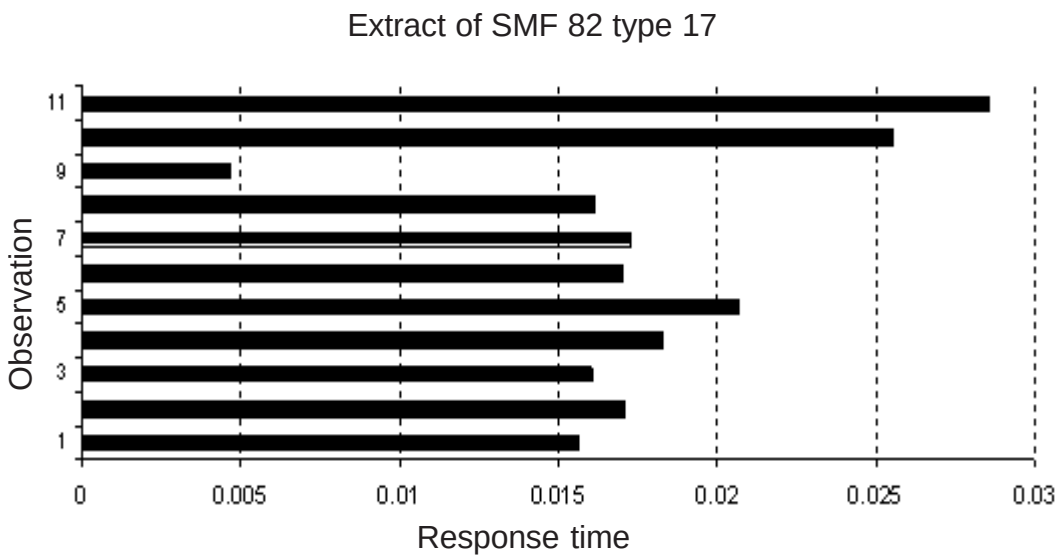

Figure 1: Coprocessor performance

Then I have calculated the following values:

- SMF82CTD - SMF82CTN – this approximates to the response time from the card.
- SMF82CTW - SMF82CTD – this approximates to the time taken to transfer the data back to the caller.

I have loaded the times produced into a spreadsheet, and graphed the results (see Figure 1).

In this way, it is easier to spot trends. This graph is an extract of data produced during test runs, and contains the card response time data from 40 SMF type 82 records produced over a period of 40 seconds. This shows that a lot of records can be produced when the cards are being heavily used, though the records themselves are quite small. The SMF records contain the card serial number, function code, etc, so further filtering is possible. However, it is important to remember that these records are only periodic and not necessarily regular, so the records do not provide an accurate time series and are only representative of the overall activity. They may be useful when trying to track delays, but this is not guaranteed.

Also, note that different function codes are recorded in these records, each with different characteristics. As can seen from the graph, some of them have much longer elapsed times than others, in normal operation, so you need to ensure that you are comparing the response times of the same functions. The amounts of data being processed per operation will also affect the elapsed times, but this is not recorded in the data. In particular, since the RMF records are regularly spaced in time, unlike the ICSF records, correlating the two types of data will require some effort. One approach to correlating the data involves grouping the ICSF records into the same time intervals as reported in RMF, eg every five minutes, and averaging the response times by card and function. In this way the variable numbers of records produced can be reduced to fewer observations. Any intervals without records can be assigned zero values. This should then produce consolidated time-sequenced values, which can then be compared against RMF reports.

CODE

Based on the ICSF type 82 record description obtained from its mapping macro (CSFSMF82, which resides in SYS1.MACLIB), a sample ICSF event report writer was written. Please note that there are/were several assembly errors in CSFSMF82 macro (see APAR OW47627, OA02792, and OW46414) so I would advise you to check the PTF database for availability of the appropriate fix. In order to extract ICSF event information from SMF data, I have constructed a three-part job stream. In the first part (DUMP82), SMF records 82 are extracted from an SMF weekly dataset to a file that can be used as a base of archived records. Please note that sorting of SMF data may issue an error message (ICE204A), set a return code of 16 and terminate if it detects an incomplete spanned record. In order to overcome this potential obstacle, DFSORTs SPANINC=RC4 option was used to remove the incomplete spanned records. It should be noted that SPANINC=RC0 tells DFSORT (Release 14) to issue a warning

message, set a return code of 0 and eliminate all incomplete spanned records it detects. Valid records (that is, complete spanned records) are recovered and written to the output data set, while SPANINC=RC4 does the same thing as SPANINC=RC0, but with a return code of 4 instead of 0. The shipped default is SPANINC=RC16. In the second step (COPY82) previously extracted records (selection being defined by INCLUDE's condition) are sorted and copied to a file, which is then input to the analysis and reporting ICSF REXX EXEC invoked in ICSF82 step.

Only one report is generated by this REXX EXEC – the ICSF/MVS event report. The report provides information about the event/operation that caused ICSF to write to the SMF record. The REXX routine was modified to parse basic SMF type 82 subtype 17 data into a CSV format file for downloading to a spreadsheet or database – perhaps the most amenable medium for analysis. This dataset can then be transferred to a PC using the transfer mechanism of your choice, for example FTP or IBM PC 3270 file transfer. Ensure that the data is transferred in ASCII mode. The data can then be imported into the spreadsheet of your choice and manipulated as required.

Sample JCL to execute SMF type 82 data extract and ICSF/MVS event reporting:

```
//*-------------------------------------------------------------------*
//*   UNLOAD SMF 82 RECORDS FROM VSAM OR VBS TO VB                    *
//*   Note: change the DUMPIN DSN=your.smfdata to be the name of      *
//*   the dataset where you currently have SMF data being             *
//*   recorded. It may be either SMF weekly dataset or an active      *
//*   dump dataset. If you choose the latter, then prior to           *
//*   executing this job, you need to terminate SMF recording         *
//*   of the currently active dump dataset to allow the               *
//*   unloading of SMF records.                                       *
//*   Also, change the DCB reference to match the name of your        *
//*   weekly SMF dump dataset.                                        *
//*-------------------------------------------------------------------*
//DUMP82   EXEC  PGM=IFASMFDP
//DUMPIN   DD    DISP=SHR,DSN=your.smfdata
//DUMPOUT  DD    DISP=(NEW,PASS),DSN=&&SMF82OUT,UNIT=SYSDA,
//         SPACE=(CYL,(2,2)),DCB=(your.smfweekly.dataset)
//SYSPRINT DD    SYSOUT=*
```

```
//SYSIN    DD    *
    INDD(DUMPIN,OPTIONS(DUMP))
    OUTDD(DUMPOUT,TYPE(82))
/*
//*-------------------------------------------------------------------*
//* COPY VBS TO VB, DROP HEADER/TRAILER RECORDS, SORT ON DATE/TIME   *
//* Note: change the SMF82 DSN=hlq.SMF82.DATA to the name of         *
//* the dataset you'll use in the last step.                         *
//*-------------------------------------------------------------------*
//COPY82   EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=&&SMF82OUT,DISP=SHR
//SMF82    DD DSN=hlq.SMF82.DATA,SPACE=(CYL,(x,y)),UNIT=SYSDA,
//         DISP=(NEW,CATLG,KEEP),
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=32760)
//TOOLIN   DD *
   SORT FROM(RAWSMF) TO(SMF82) USING(SMFI)
//SMFICNTL DD *
   OPTION SPANINC=RC4,VLSHRT
   INCLUDE COND=(6,1,BI,EQ,82)
   SORT FIELDS=(11,4,PD,A,7,4,BI,A)
/*
//*-------------------------------------------------------------------*
//*   FORMAT ICSF/MVS TYPE 82 RECORDS                                 *
//*   Note: change the SYSEXEC DSN=your.rexx.library to the name      *
//*   of the dataset where you have placed the ICSFF REXX EXEC.       *
//*   Also, change the ICSF DSN=hlq.SMF82.DATA to the name of         *
//*   the dataset you have created in the previous step.              *
//*-------------------------------------------------------------------*
//ICSF82   EXEC PGM=IKJEFT01,REGION=0M
//SYSEXEC  DD DISP=SHR,DSN=your.rexx.library
//ICSF     DD DISP=SHR,DSN=hlq.SMF82.DATA
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
prof nopref
%ICSF
/*
```

ICSFE reporting EXEC:

```
/* REXX EXEC to read and format ICSF/MVS records */
 Numeric digits 16
 userid=SYSVAR(SYSUID)
 r82icsf = userid||'.icsf.rep'             /* ICSF/MVS report file */
 r82csv  = userid||'.icsfcsv.rep'          /* ICSF/MVS CSV file    */

 If SYSDSN(r82icsf) = 'OK'
 Then "DELETE "r82icsf" PURGE"
   "ALLOC FILE(REPORT) DA("r82icsf")",
```

```
   "UNIT(SYSALLDA) NEW TRACKS SPACE(90,5) CATALOG",
   "REUSE LRECL(180) RECFM(F B) "

 If SYSDSN(r82csv) = 'OK'
 Then "DELETE "r82csv" PURGE"
   "ALLOC FILE(S82CSV) DA("r82csv")",
   "UNIT(SYSALLDA) NEW TRACKS SPACE(1,1) CATALOG",
   "REUSE LRECL(40) RECFM(F B) "
 /*-------------------------------------------------------*/
 /* Short subtypes label                                  */
 /*-------------------------------------------------------*/
label.1 = "ICSF Initialization"
label.3 = "Processors status change"
label.4 = "Error handling for CCP feature"
label.5 = "Special Security mode changed"
label.6 = "Master Key Part Entry"
label.7 = "Operation Key Part Entry"
label.8 = "Cryptographic Key Data Set Refreshed"
label.9 = "Dynamic CKDS Update"
label.10= "Clear key part for the PKA"
label.11= "Clear New Master Key Part Entry"
label.12= "PKSC Commands"
label.13= "Dynamic Public KDS Update"
label.14= "PCI Crypto Coprocessor Clear Master Key"
label.15= "Retained key create/delete"
label.16= "PCI Interface"
label.17= "PCI Crypto Coprocessor Timing"
label.18= "PCI Crypto Coprocessor Configuration"
 /*-------------------------------------------------------*/
 /*  Print report header                                  */
 /*-------------------------------------------------------*/
 hdr.1 = left('ICSF/MVS Event report',50)
 hdr.2 = left(' ',1,' ')
 hdr.3 = left('Report produced on',18),
         ||left(' ',1,' ')||left(date(),11),
         ||left('at',3,' ')||left(time(),10)
 hdr.4 = left(' ',1,' ')
 hdr.5 = left('Date and time',13)||left(' ',11,' '),
         ||left('ICSF Event',11)||left(' ',3,' '),
         ||left('Record subtype',15)
 hdr.6 = left('-',60,'-')
 "EXECIO * DISKW REPORT (STEM hdr.)"

 ccr.1  ='ICSF/MVS response timing CSV file'
 ccr.2  = 'Response time'||";"||'Transfer time'
 "EXECIO * DISKW S82CSV (STEM ccr.)"

 Totrec = 0; subtype_count.= 0; indent = "  "
 sep = left('.',60,'.')
 /*-------------------------------------------------------*/
```

```
 /*                 Main processing loop                      */
  /*-----------------------------------------------------------*/
 DO FOREVER
   "EXECIO 1 DISKR ICSF"
    IF RC ¬= Ø THEN call End_of_file
 else do
    PARSE PULL record
    Totrec = Totrec + 1         /* Count the total records processed */
    PARSE VAR record header 15 rest
    smf82rty  = c2d(substr(header,2,1))             /* Record type  */

    call SMF82_header
  end
 End
  /*-----------------------------------------------------------*/
 /* Print out record count at end of input file               */
  /*-----------------------------------------------------------*/
End_of_file:
     call printline " "
  call printline "Total SMF 82 (ICSF/MVS) records read: " left(Totrec,4)
     call printline " "
     call printline " sub  "
     call printline " type count percent Subtype label"
 call printline " ---------------------------------------------------"
     do j=1 to 25
      if subtype_count.j > Ø then
     call printline right(j,3) format(subtype_count.j,6),
                    format((subtype_count.j/Totrec)*1ØØ,5,2),
                    label.j
     end
 /* Close & free all allocated files  */
 "EXECIO Ø DISKR ICSF (FINIS"
 "EXECIO Ø DISKW REPORT (FINIS"
 "EXECIO Ø DISKW S82CSV (FINIS"
  say "ICSF/MVS report ......."r82icsf
  say "ICSF/MVS csv file ....."r82csv
 "FREE FILE(ICSF REPORT S82CSV)"
 EXIT Ø
  /*-----------------------------------------------------------*/
 /*        Basic section of the SMF82                         */
  /*-----------------------------------------------------------*/
SMF82_header:
smf82flg = x2b(c2x(substr(header,Ø1,Ø1)))        /* System indicator */
smf82rty = c2d(substr(header,Ø2,1))              /* Record type      */
smf82tme = smf(c2d(substr(header,Ø3,Ø4)))   /*Time record was written*/
smf82dte = substr(c2x(substr(header,Ø7,Ø4)),3,5)
                                          /* Date record was written */
smf82sid = substr(header,11,Ø4)             /* System Identification */
smf82ssi = substr(rest,1,Ø)                      /* Subsystem ID     */
smf82sty = c2d(substr(rest,5,2))                 /* Record subtype   */
```

```
subtype_count.smf82sty = subtype_count.smf82sty + 1
Sub_line = left(Date('N',smf82dte,'J'),11)
Sub_line = Sub_line smf82tme              /* SMF time in hh:mm:ss.th */
call SMF82rec                             /* Process SMF82rec record */
return
 /*-------------------------------------------------------------*/
/*    SMF82rec: get subtype info                                 */
 /*-------------------------------------------------------------*/
SMF82rec:
    PARSE var rest first 7 subtype
SMF82rec_subtype:
    Select
      when smf82sty = "Ø1" then call subtypeØ1
      when smf82sty = "Ø3" then call subtypeØ3
      when smf82sty = "Ø4" then call subtypeØ4
      when smf82sty = "Ø5" then call subtypeØ5
      when smf82sty = "Ø6" then call subtypeØ6
      when smf82sty = "Ø7" then call subtypeØ7
      when smf82sty = "Ø8" then call subtypeØ8
      when smf82sty = "Ø9" then call subtypeØ9
      when smf82sty = "1Ø" then call subtype1Ø
      when smf82sty = "11" then call subtype11
      when smf82sty = "12" then call subtype12
      when smf82sty = "13" then call subtype13
      when smf82sty = "14" then call subtype14
      when smf82sty = "15" then call subtype15
      when smf82sty = "16" then call subtype16
      when smf82sty = "17" then call subtype17
      when smf82sty = "18" then call subtype18
      when smf82sty = "19" then call subtype19
      otherwise nop
    End                                       /* of select */
return
subtypeØ1:
 /*-------------------------------------------------------------*/
/* ICSF Initialization (Subtype 1) - written whenever          */
/* ICSF/MVS is started                                         */
 /*-------------------------------------------------------------*/
 call printline Sub_line "INITIALIZATION (Subtype:" ||smf82sty||")"
 smf82ves =     substr(subtype,Ø1,4)       /* CCVE status bits       */
 smf82vts = c2x(substr(subtype,Ø5,1))      /* CCVT status bits       */
 smf82ido = c2d(substr(subtype,Ø6,1))  /*Current crypto domain index*/
 smf82ite = c2d(substr(subtype,Ø9,4))      /*Number of trace entries */
 smf82ckd =     substr(subtype,13,44)      /* CKDS name              */
 smf82iml = c2d(substr(subtype,57,4))      /*Maximum length for data */
 smf82usr =     substr(subtype,61,8)      /* Userparm from options ds*/
 smf82pkd =     substr(subtype,69,44)      /* PKDS name              */
 indenØ = left(' ',23,' ');          inden3="Crypto comm. status:   "
 inden4="Crypto domain index:   ";  inden5="Num. of trace entries: "
 inden6="Crypto key data set:   ";  inden7="Max length for data:   "
```

```
 inden8="Userparm options ds:   ";  inden9="PKDS name:              "
 inden2 = "    "
  /*-------------------------------------------------------*/
 /* Decode and print Cryptographic Communication Vector     */
 /* table Extension (CCVE) status bits.                     */
  /*-------------------------------------------------------*/
   smf82sma = '80000000'x  /* Special Security Mode allowed */
   smf82sme = '40000000'x  /* Special Security Mode enabled */
   smf82ka  = '10000000'x  /* Key authentication            */
   smf82rac = '02000000'x  /* RACF checking                 */
   smf82enc = '00040000'x  /* Compenc from options ds       */

  if bitand(smf82ves,smf82sma) = smf82sma then
     inden2="Crypto comm. extension:"
     call printline inden2 "Special security mode allowed"
  if bitand(smf82ves,smf82sme) = smf82sme then
     call printline inden0 "Special security mode enabled"
  if bitand(smf82ves,smf82ka) = smf82ka then
     call printline inden0 "Key authentication"
  if bitand(smf82ves,smf82rac) = smf82rac then
     call printline inden0 "RACF checking of supervisor-state callers"
  if bitand(smf82ves,smf82enc) = smf82enc then
     call printline inden0 "Encryption algorithm in the installation"
  /*-------------------------------------------------------*/
 /* Decode and print Cryptographic Communication Vector     */
 /* Table (CCVT) status bits.                               */
  /*-------------------------------------------------------*/
   smf82cpa = '08000000'x          /* Compatibility mode bit*/
   if bitand(smf82vts,smf82cpa) = smf82cpa then
      call printline inden3 "Compatible with CUSP and PCF"
    msg = inden3 smf82vts; call printline msg
    msg = inden4 smf82ido; call printline msg
    msg = inden5 smf82ite; call printline msg
    msg = inden6 smf82ckd; call printline msg
    msg = inden7 smf82iml; call printline msg
    msg = inden8 smf82usr; call printline msg
    msg = inden9 smf82pkd; call printline msg
    call printline sep
return
subtype03:
  /*-------------------------------------------------------*/
 /* Status Change (Subtype 3) - written whenever there is   */
 /* a change to the number of available processors with     */
 /* the cryptographic feature                               */
  /*-------------------------------------------------------*/
 seclen = 56
 call printline Sub_line "STATUS CHANGE (Subtype:" ||smf82sty||")"
 stibit ="Master key verification:        "
 PARSE var subtype smf82sns 5 section; smf82sns = c2d(smf82sns)
  /*-------------------------------------------------------*/
```

```
 /*  Status Change Sections:                                    */
  /*-----------------------------------------------------------*/
 do j = Ø to smf82sns -1
 secoff = substr(section,(j*seclen)+1,seclen)
 smf82spr = c2d(substr(secoff,Ø1,4))     /* Processor number         */
 smf82ksu = c2d(substr(secoff,Ø5,4))  /*Key storage unit (KSU) number*/
 smf82sdx = c2d(substr(secoff,Ø9,4))    /*Current crypto domain index*/
 smf82ver = c2d(substr(secoff,13,4))     /*Current master key version*/
 smf82ssw = c2d(substr(secoff,17,4)) /*Ø if no error condition exists*/
                                  /* with the processor. Otherwise,  */
 smf82sti =     substr(secoff,21,4)      /* the ICSF status word     */
 smf82mkf = c2x(substr(secoff,21,4))     /* Master key flag          */
 smf82cvp = c2x(substr(secoff,25,8))
                                   /*Current master key verfy pattern*/
 smf82nap = c2x(substr(secoff,33,8))    /*New master key auth pattern*/
 smf82nvp = c2x(substr(secoff,41,8))   /*New master key verfy pattern*/
 smf82ovp = c2x(substr(secoff,49,8))   /*Old Master key verfy pattern*/
 msg = "Processor number:                 "smf82spr; call printline msg
 msg = "Key storage unit number:          "smf82ksu; call printline msg
 msg = "Current crypto domain index:      "smf82sdx; call printline msg
 msg = "Current master key version:       "smf82ver; call printline msg
 msg = "Processor error condition:        "smf82ssw; call printline msg
 msg = "Master key bit flag:              "smf82mkf; call printline msg
 msg = "Current master key verfy pattern:"smf82cvp; call printline msg
 msg = "New master key auth pattern:      "smf82nap; call printline msg
  /*-----------------------------------------------------------*/
 /* Decode and print Master key bit flag                       */
  /*-----------------------------------------------------------*/
   smf82cv =  '8Ø000000'x  /* Current master key ver pattern */
   smf82na =  '4Ø000000'x  /* New master key auth pattern    */
   smf82nv =  '2Ø000000'x  /* New master key ver pattern     */
   smf82ov =  '1Ø000000'x  /* Old master key ver pattern     */
 if bitand(smf82sti,smf82cv) = smf82cv then
   call printline stibit "Current master key verification pattern valid"
 if bitand(smf82sti,smf82na) = smf82na then
    call printline indenØ "New master key authentication pattern valid"
 if bitand(smf82sti,smf82nv) = smf82nv then
    call printline indenØ "New master key verification pattern valid"
 if bitand(smf82sti,smf82ov) = smf82ov then
    call printline indenØ "Old master key verification pattern valid"
 msg = "New master key verfy pattern:    "smf82nvp;call printline msg
 msg = "Old Master key verfy pattern:    "smf82ovp;call printline msg
 msg = "             ";                          call printline msg
 end
    call printline sep
return
subtypeØ4:
  /*-----------------------------------------------------------*/
 /* Condition Code Three Section (Subtype 4) - written      */
 /* whenever ICSF/MVS handles error conditions for          */
```

```
 /* Cryptographic Coprocessor feature failure (CC3, Reason  */
 /* Code 1) or cryptographic tampering CC3 Reason Code 3)   */
  /*-------------------------------------------------------*/
 call printline Sub_line "CONDITION CODE (Subtype:" ||smf82sty||")"
  smf823sw = c2d(substr(subtype,Ø1,4))        /* Status word from CC3*/
  smf823pr = c2d(substr(subtype,Ø5,1))         /* CPU number         */
  smf823dx = c2d(substr(subtype,Ø6,1))  /*Current Crypto Domain Index*/
  msg = "Status word from condition code: "smf823sw; call printline msg
  msg = "Processor number:                "smf823pr; call printline msg
  msg = "Current crypto domain index:     "smf823dx; call printline msg
  call printline sep
return
subtypeØ5:
  /*-------------------------------------------------------*/
 /* Special Secure Mode Change (Subtype 5) - is written     */
 /* whenever a change to special security  mode is detected */
  /*-------------------------------------------------------*/
 ssmtxt ="Special security mode: "
 call printline Sub_line "SPECIAL SECURITY MODE (Subtype:"
||smf82sty||")"
  smf82ssb = substr(subtype,Ø1,8)  /*Special security mode (SSM) bits*/
  smf82ssa = '8ØØØØØØØØØØØØØØØ'x   /* SSM mode is enabled:           */
                                   /* ON=SSM mode went from OFF to ON*/
                                  /* OFF=SSM mode went from ON to OFF*/
  /*-------------------------------------------------------*/
 /* Decode and print Special security mode bits             */
  /*-------------------------------------------------------*/
 if bitand(smf82ssb,smf82ssa) = smf82ssa then
    call printline ssmtxt "SSM mode is enabled"
    call printline sep
return
subtypeØ6:
  /*-------------------------------------------------------*/
 /* Master Key Part Entry (Subtype 6) - written whenever    */
 /* a master key parts are entered using TKE workstation    */
 /* and are processed using the TKE master key entry ICSF   */
 /* ICSF panels.                                            */
  /*-------------------------------------------------------*/
 mkptxt ="Master Key part entry:                "
 call printline Sub_line "MASTER KEY PART ENTRY (Subtype:"
||smf82sty||")"
  smf82mkb = substr(subtype,Ø1,4)      /*Master key part (MKPART) bits*/
  smf82mkx = c2x(substr(subtype,Ø1,4)) /*Master key part (MKPART) bits*/
  smf82nmv = c2x(substr(subtype,Ø5,8)) /*New master key ver pattern   */
  smf82omv = c2x(substr(subtype,13,8))    /* Ver pattern for key part */
  smf82mks = c2d(substr(subtype,21,1))    /* KSU number               */
  smf82mdx = c2d(substr(subtype,22,1))  /*Current crypto domain index */

  msg = "Master key part bits (MKPART):         "smf82mkx; call
printline msg
```

```
  msg = "New master key verification pattern:   "smf82nmv; call
printline msg
  msg = "Verification pattern for the key part: "smf82omv; call
printline msg
  msg = "Key storage unit (KSU) number:         "smf82mks; call
printline msg
  msg = "Current crypto domain index:           "smf82mdx; call
printline msg
 /*-------------------------------------------------------------*/
 /* Decode and print Master key part (MKPART) bits              */
 /*-------------------------------------------------------------*/
   smf82nvv = '80000000'x        /* New master key vp valid */
   smf82ovv = '40000000'x        /* Old master key vp valid */
    if bitand(smf82mkb,smf82nvv) = smf82nvv then
       call printline mkptxt "New master key verification pattern valid"
    if bitand(smf82mkb,smf82ovv) = smf82ovv then
       call printline mkptxt "Old master key verification pattern valid"
       call printline sep
return
subtypeØ7:
 /*-------------------------------------------------------------*/
 /* Operation Key Part Entry (Subtype 7) - written whenever */
 /* a key parts are entered using the TKE workstation       */
 /* and are processed using the operational key entry ICSF  */
 /* panels.                                                 */
 /*-------------------------------------------------------------*/
 okptxt ="Operation Key Part Entry:  "
 call printline Sub_line "OPERATION KEY PART ENTRY (Subtype:"
||smf82sty||")"
 smf82kpb =      substr(subtype,Ø1,4)    /* Key part (KPART) bits    */
 smf82kpx = c2x(substr(subtype,Ø1,4))       /* KPART bits            */
 smf82kv  = c2x(substr(subtype,Ø5,8)) /*Key part verification pattern*/
 smf82kks = c2d(substr(subtype,13,1))       /* KSU number            */
 smf82kdx = c2d(substr(subtype,14,1))    /*Current crypto domain index*/
 smf82kck =     substr(subtype,17,44)       /* CKDS name             */
 smf82kcl = c2d(substr(subtype,61,72))    /*CKDS entry being modified*/
 msg = "Key part (KPART) bits:       "smf82kpx; call printline msg
 msg = "Key part verify pattern:     "smf82kv ; call printline msg
 msg = "Key storage unit number:     "smf82kks; call printline msg
 msg = "Current crypto domain index: "smf82kdx; call printline msg
 /*-------------------------------------------------------------*/
 /* Decode and print Key part (KPART) bits                      */
 /*-------------------------------------------------------------*/
   smf82kvv =  '80000000'x  /* Key part verify pattern valid */
 if bitand(smf82kpb,smf82kvv) = smf82kvv then
  call printline okptxt "Key part verification pattern valid"
  msg = "CKDS name:                   "smf82kck;call printline msg
  msg = "CKDS entry being modified:   "smf82kcl;call printline msg
  call printline sep
return
```

```
subtypeØ8:
 /*-------------------------------------------------------------*/
 /* Cryptographic Key Data Set Refresh Section (Subtype 8)  */
 /* - is written whenever the in-storage copy of the CKDS   */
 /* is refreshed.                                           */
 /*-------------------------------------------------------------*/
 call printline Sub_line "CKDS REFRESH (Subtype:" ||smf82sty||")"
 smf82roc = substr(subtype,Ø1,44)    /*Name of the CKDS being replaced*/
 smf82rnc = substr(subtype,45,44)    /*Name of the CKDS to replace the*/
 ynden0="Old CKDS name:         "        /* current CKDS              */
 ynden1="New CKDS name:         "
    msg = ynden0 smf82roc; call printline msg
    msg = ynden1 smf82rnc; call printline msg
    call printline sep
return
subtypeØ9:
 /*-------------------------------------------------------------*/
 /* Dynamic CKDS Update (Subtype 9) - written whenever an   */
 /* application uses the dynamic CKDS update services to    */
 /* write to the CKDS                                       */
 /*-------------------------------------------------------------*/
 dyn = "Dynamic CKDS Update event: "
 call printline Sub_line "DYNAMIC CKDS UPDATE (Subtype:" ||smf82sty||")"
 smf82ucb =     substr(subtype,Ø1,Ø4)          /* Update CKDS bits   */
 smf82ucx = c2x(substr(subtype,Ø1,Ø4))         /* Update CKDS bits   */
 smf82ucn =     substr(subtype,Ø5,44)          /* CKDS name          */
 smf82ucl =     substr(subtype,49,72)     /*CKDS entry being modified*/
 msg = "Dynamic CKDS Update bits:  "smf82ucx; call printline msg
 /*-------------------------------------------------------------*/
 /* Decode and print Dynamic CKDS Update bits               */
 /*-------------------------------------------------------------*/
   smf82uca = '8Ø000000'x          /* CKDS record added     */
   smf82ucc = '4Ø000000'x          /* CKDS record changed   */
   smf82ucd = '2Ø000000'x          /* CKDS record deleted   */
 if bitand(smf82ucb,smf82uca) = smf82uca then
    call printline dyn "CKDS record added"
 if bitand(smf82ucb,smf82ucc) =  smf82ucc then
    call printline dynn "CKDS record changes"
 if bitand(smf82ucb,smf82ucd) = smf82ucd then
    call printline dyn "CKDS record deleted"
 msg = "Name of the CK data set:   "smf82ucn; call printline msg
 msg = "CKDS entry being modified: "smf82ucl; call printline msg
 call printline sep
return
subtype1Ø:
 /*-------------------------------------------------------------*/
 /* PKA Key Part Entry (Subtype 1Ø) - is written            */
 /* when a clear key part is entered for one of the PKA     */
 /* master keys, ie when you use the ICSF panels to enter   */
 /* PKA master key parts                                    */
```

```
 /*-------------------------------------------------------*/
 pkat = "PKA Key Part Entry:     "
 pkkt = "                        "
 call printline Sub_line "PKA KEY PART ENTRY (Subtype:" ||smf82sty||")"
  smf82pkb =     substr(subtype,Ø1,Ø4)     /* PKA part bits          */
  smf82pkz = c2x(substr(subtype,Ø1,Ø4))    /* PKA part bits (hex)    */
  smf82php = c2x(substr(subtype,Ø5,16))    /* Master key hash pattern*/
  smf82kph = c2x(substr(subtype,21,16))    /* Key part hash pattern  */
  smf82pks = c2d(substr(subtype,37,Ø1))    /* KSU number             */
  smf82pkx = c2d(substr(subtype,38,Ø1))    /*Current crypto domain ix*/
  msg = "Master key hash pattern: "smf82php; call printline msg
  msg = "Key part hash pattern:   "smf82kph; call printline msg
  msg = "PKA Key Part Entry bits: "smf82pkz; call printline msg
 /*-------------------------------------------------------*/
 /* Decode and print PKA Key Part Entry bits               */
 /*-------------------------------------------------------*/
  smf82rhv = '8Ø000000'x           /* KMMK processed        */
  smf82shv = '4Ø000000'x           /* SMK processed         */
  smf82khv = '2Ø000000'x           /* KP hash pattern valid */
  smf82mhv = '1Ø000000'x           /* MK hash pattern valid */
  if bitand(smf82pkb,smf82rhv) = smf82rhv then
     call printline pkat "Key management master key processed"
  if bitand(smf82pkb,smf82shv) = smf82shv then
     call printline pkat "Signature master key processed"
  if bitand(smf82pkb,smf82khv) = smf82khv then
     call printline pkkt "Key part hash pattern valid"
  if bitand(smf82pkb,smf82mhv) = smf82mhv then
     call printline pkkt "Master key hash pattern valid"
  msg = "Key storage unit no.:    "smf82pks; call printline msg
  msg = "Current crypto domain ix:"smf82pkx; call printline msg
  call printline sep
return
subtype11:
 /*-------------------------------------------------------*/
 /* Clear New Master Key Part Entry (Subtype 11) - written  */
 /* when a clear key part is entered for the DES master     */
 /* key i.e when you use the ICSF panels to enter new       */
 /* master key parts.                                       */
 /*-------------------------------------------------------*/
 nme = "Clear New Master key event:"
 call printline Sub_line "CLEAR NEW MASTER KEY PART (Subtype:"
||smf82sty||")"
 smf82cmb =     substr(subtype,Ø1,4)     /*Clear new master key bits */
 smf82cmx = c2x(substr(subtype,Ø1,4))
                                   /*Clear new master key bits (hex)*/
 smf82chp = c2x(substr(subtype,Ø5,16))
                                  /* Clear new master key hash patt. */
 smf82cnp = c2x(substr(subtype,21,8))
                                  /* Clear new master key ver. patt. */
 smf82cpw = c2x(substr(subtype,29,16))   /* Key part hash pattern    */
```

```
 smf82cpq = c2x(substr(subtype,45,8))
                                  /* Key part verification pattern   */
 smf82cks = c2d(substr(subtype,53,1))    /* KSU number                */
 smf82cdx = c2d(substr(subtype,54,1))   /*Current crypto domain index*/
 msg = "Clear NM key hash pattern:  "smf82chp; call printline msg
 msg = "Clear NM key ver pattern:   "smf82cnp; call printline msg
 msg = "Key part hash patter:       "smf82cpw; call printline msg
 msg = "Key part verification pat:  "smf82cpq; call printline msg
 msg = "Clear NM key bits:          "smf82cmx; call printline msg
  /*-------------------------------------------------------------*/
 /* Decode and print Clear new master key bits                  */
  /*-------------------------------------------------------------*/
   smf82cmh = '80000000'x          /* Clear NMK HP valid    */
   smf82cmv = '40000000'x          /* Clear NMK VP valid    */
   smf82cph = '20000000'x          /* Clear NMK KP HP valid */
   smf82cpv = '10000000'x          /* Clear NMK KP VP valid */
 if bitand(smf82cmb,smf82cmh) = smf82cmh then
    call printline nme "Clear NM key hash pattern valid"
 if bitand(smf82cmb,smf82cmv) = smf82cmv then
    call printline nme "Clear NM key"
 if bitand(smf82cmb,smf82cph) = smf82cph then
    call printline nme "Clear NM key part hash pattern valid"
 if bitand(smf82cmb,smf82cpv) = smf82cpv then
    call printline nme "Clear NM verification pattern valid"
 msg = "Key storage unit number:    "smf82cks; call printline msg
 msg = "Current crypto domain ix:   "smf82cdx; call printline msg
 call printline sep
return
subtype12:
  /*-------------------------------------------------------------*/
 /* PKSC Commands (Subtype 12) - written for each request       */
 /* and reply from calls to the CSFSPKSC service by TKE.        */
  /*-------------------------------------------------------------*/
 call printline "Subtype=" || smf82sty "PKSC Commands "
 smf82psq =     substr(subtype,Ø1,1Ø24)   /*The complete PKSC request*/
 smf82pr1 =     substr(subtype,Ø1,Ø1)      /* Reserved (X'ØØ')       */
 smf82prt =     substr(subtype,Ø2,Ø1)      /* Request message type   */
 xmf82prt = c2x(substr(subtype,Ø2,Ø1))    *Request message type (hex)*/
 smf82prl = c2d(substr(subtype,Ø3,Ø2))     /* Message length         */
 smf82prc =     substr(subtype,Ø5,Ø1)      /* PKSC command code      */
 xmf82prc = c2x(substr(subtype,Ø5,Ø1))     /*PKSC command code (hex) */
 smf82par = c2d(substr(subtype,Ø6,Ø1))     /*Number of authorization */
                                           /* register to be used to */
                                           /* verify signature on a  */
                                        /* signed command (X'ØØ' for */
                                        /* unsigned commands, X'FF'  */
                                        /* for self-signature        */
                                        /* initialization commands)  */
 smf82pda = c2d(substr(subtype,Ø7,Ø1))     /* Domain index or        */
                                           /* Authorization register */
```

```
                                          /* number                */
smf82pkt = c2d(substr(subtype,07,01))     /* Master key type for   */
                                        /* extract and encrypt mstr. */
                                        /* key, key part type for    */
                                        /* load key part,            */
                                        /* X'00' otherwise           */
smf82pid = c2x(substr(subtype,08,16))    /*Query ID for request type*/
                                        /* UCQ, Crypto Module ID for */
                                        /* request type SCB          */
smf82pts = c2x(substr(subtype,26,16))     /* Transaction SN for    */
                                          /* request type SCB      */
smf82pr2 = c2x(substr(subtype,43,984))    /*From here to the end of */
                                        /* the request is available  */
                                        /* for IBM diagnosis only    */
msg = "No.of authorization register:  "smf82par; call printline msg
msg = "Authorization register no.:    "smf82pda; call printline msg
msg = "Master key type/key part type: "smf82pkt; call printline msg
 /*-----------------------------------------------------------*/
/* Decode and print Request message type bits                 */
 /*-----------------------------------------------------------*/
reqtyp = "Request type:     "
reqbla = "Query ID:         "
reqmod = "Crypto Module ID: "
reqmdd = "Transaction SN:   "
komma  = "PKSC command:     "
repkom = "Requesr reply:    "
smf82puc = '02000000'x      /* Type is UCQ (Query request)  */
smf82pcr = '03000000'x      /* Type is SCB (Command request)*/
if bitand(smf82prt,smf82puc) = smf82puc then do
   call printline reqtyp "Query request"
   call printline reqbla smf82pid
   end
if bitand(smf82prt,smf82pcr) = smf82pcr then do
   call printline reqtyp "Command request"
   call printline reqmod smf82pid
   call printline reqmdd smf82pts
   end
 /*-----------------------------------------------------------*/
/* Decode and print Query/Command request                     */
 /*-----------------------------------------------------------*/
     smf82qmi =  '00000000'x;  smf82qar =  '01000000'x
     smf82qcb =  '02000000'x;  smf82qpc =  '03000000'x
     smf82qhm =  '04000000'x;  smf82qhg =  '05000000'x
     smf82qhf =  '06000000'x;  smf82qbt =  '07000000'x
     smf82qdi =  '08000000'x;  smf82qpt =  '09000000'x
     smf82cos =  '30000000'x;  smf82lhm =  '32000000'x
     smf82lhg =  '33000000'x;  smf82chk =  '34000000'x
     smf82lap =  '70000000'x;  smf82lcb =  '71000000'x
     smf82zd  =  '72000000'x;  smf82lec =  '73000000'x
     smf82xem =  '74000000'x;  smf82lkp =  '75000000'x
```

```
     smf82xes =  '76000000'x;  smf82xer =  '76000000'x
     smf82lcs =  '77000000'x;  smf82lcr =  '77000000'x
   tmpq = "Query module information"
if bitand(smf82prc,smf82qmi) = smf82qmi then do
   call printline komma tmpq
   tmpq = "Query authorization register"
if bitand(smf82prc,smf82qar) = smf82qar then do
   call printline komma tmpq
   tmpq = "Query pksc control block"
if bitand(smf82prc,smf82qcb) = smf82qcb then do
   call printline komma tmpq
   tmpq = "Query pending-command register"
if bitand(smf82prc,smf82qpc) = smf82qpc then do
   call printline komma tmpq
   tmpq = "Query dh modulus"
if bitand(smf82prc,smf82qhm) = smf82qhm then do
   call printline komma tmpq
   tmpq = "Query dh generator"
if bitand(smf82prc,smf82qhg) = smf82qhg then do
   call printline komma tmpq
   tmpq = "Query dh first key part"
if bitand(smf82prc,smf82qhf) = smf82qhf then do
   call printline komma tmpq
   tmpq = "Query basic transport"
if bitand(smf82prc,smf82qbt) = smf82qbt then do
   call printline komma tmpq
   tmpq = "Query domain information"
if bitand(smf82prc,smf82qdi) = smf82qdi then do
   call printline komma tmpq
   tmpq = "Query pka transport"
if bitand(smf82prc,smf82qpt) = smf82qpt then do
   call printline komma tmpq
   tmpq = "co_sign"
if bitand(smf82prc,smf82cos) = smf82cos then do
   call printline komma tmpq
   tmpq = "Load dh modulus"
if bitand(smf82prc,smf82lhm) = smf82lhm then do
   call printline komma tmpq
   tmpq = "Load g and generate dh first part"
if bitand(smf82prc,smf82lhg) = smf82lhg then do
   call printline komma tmpq
   tmpq = "Combine dh key parts"
if bitand(smf82prc,smf82chk) = smf82chk then do
   call printline komma tmpq
   tmpq = "Load authorization public modulus"
if bitand(smf82prc,smf82lap) = smf82lap then do
   call printline komma tmpq
   tmpq = "Load pksc control block"
if bitand(smf82prc,smf82lcb) = smf82lcb then do
   call printline komma tmpq
```

```
   tmpq = "Zeroize domain"
if bitand(smf82prc,smf82zd ) = smf82zd  then do
   call printline komma tmpq
   tmpq = "Load environment-control mask"
if bitand(smf82prc,smf82lec) = smf82lec then do
   call printline komma tmpq
   tmpq = "Extract and encrypt master key"
if bitand(smf82prc,smf82xem) = smf82xem then do
   call printline komma tmpq
   tmpq = "Load key part"
if bitand(smf82prc,smf82lkp) = smf82lkp then do
   call printline komma tmpq
   tmpq = "Extract and encrypt smk"
if bitand(smf82prc,smf82xes) = smf82xes then do
   call printline komma tmpq
   tmpq = "Extract and encrypt rmk"
if bitand(smf82prc,smf82xer) = smf82xer then do
   call printline komma tmpq
   tmpq = "Load and combine smk"
if bitand(smf82prc,smf82lcs) = smf82lcs then do
   call printline komma tmpq
   tmpq = "Load and combine rmk"
if bitand(smf82prc,smf82lcr) = smf82lcr then do
   call printline komma tmpq
 /*-------------------------------------------------------------*/
/* Decode and print PKSC response                              */
 /*-------------------------------------------------------------*/
smf82psp = substr(subtype,1025,1024)
                                /* The corresponding PKSC response */
smf82rr1 = c2d(substr(subtype,1025,01))    /* Reserved (X'00')      */
smf82rrt =     substr(subtype,1026,01)     /* Reply type            */
xmf82rrt = c2x(substr(subtype,1026,01))    /* Reply type (hex)      */
 /*-------------------------------------------------------------*/
/* Decode and print Reply message type bits                    */
 /*-------------------------------------------------------------*/
  smf82rpu = '82000000'x   /* Unsigned reply (type URN or UNI) */
  smf82rps = '83000000'x   /* Signed reply (type SRN or SRQ)   */
smf82rpl = c2d(substr(subtype,1027,02))    /* Message length        */
smf82rrc = substr(subtype,1029,01)         /* Reply code            */
smf82rr2 = substr(subtype,1030,01)         /* Reserved (X'00')      */
smf82rrp = substr(subtype,1031,02))        /* For type URN: X'0000' */
                                  /* For type UNI: 7-bit PRNIC, */
                                  /* right-justified            */
smf82ryp = substr(subtype,1033,02)         /* For type SRN and SRQ  */
 /*-------------------------------------------------------------*/
/* Decode and print Reply type                                 */
 /*-------------------------------------------------------------*/
  smf82urn = '0000000000000000'      /* Unsigned reply URN */
  smf82srn = '0001000000000000'      /* Reply type is SRN  */
  smf82srq = '0002000000000000'      /* Reply type is SRQ  */
```

```
if bitand(smf82ryp,smf82srn) = smf82srn then
   sign = "Reply type is SRN"
if bitand(smf82ryp,smf82srq) = smf82srq then
   sign = "Reply type is SRQ"
Select
when bitand(smf82rrp,smf82urn) = smf82urn then,
          unsign = "Unsigned type is URN"
otherwise unsign = "Unsigned type is UNI"
End
smf82rcm =     substr(subtype,1035,16)     /* Crypto Module ID      */
smf82rsn = c2x(substr(subtype,1051,16))    /*Crypto Module Signature*/
                                           /* Sequence Number       */
smf82rr3 = c2x(substr(subtype,1067,152))   /* Available for IBM     */
                                           /* diagnosis only        */
smf82rts = c2x(substr(subtype,1219,16))    /* Transaction sequence  */
                                           /* number for QUERY      */
                                           /* AUTHORIZATION REGISTER*/
smf82rr4 = c2(substr(subtype,1235,816))    /*From here to the end of*/
                                     /* the reply is available    */
                                     /* for IBM diagnosis only    */
msg = "Crypto Module ID:             "smf82rcm; call printline msg
msg = "Crypto Module Signature Seq.: "smf82rsn; call printline msg
rpl = "Reply type:                   "
rpls= "Signed reply type:            "
upls= "Unsigned reply type:          "
if bitand(smf82rrt,smf82rpu) = smf82rpu then do
   call printline rpl "Unsigned reply (type URN or UNI)"
   call printline upls unsign
   end
if bitand(smf82rrt,smf82rps) = smf82rps then do
   call printline rpl "Signned reply (type SRN or SRQ)"
   call printline rpls sign
   end
 /*-------------------------------------------------------------*/
/* Decode and print Query/Command request reply                 */
 /*-------------------------------------------------------------*/
     smf82r00 =  '00000000'x;   smf82r10 =  '10000000'x
     smf82r21 =  '21000000'x;   smf82r22 =  '22000000'x
     smf82r23 =  '23000000'x;   smf82r24 =  '24000000'x
     smf82r25 =  '25000000'x;   smf82r26 =  '26000000'x
     smf82r27 =  '27000000'x;   smf82r28 =  '28000000'x
     smf82r31 =  '31000000'x;   smf82r33 =  '33000000'x
     smf82r41 =  '41000000'x;   smf82r42 =  '42000000'x
     smf82r51 =  '51000000'x;   smf82r61 =  '61000000'x
     smf82r62 =  '62000000'x;   smf82r63 =  '63000000'x
     smf82r64 =  '64000000'x;   smf82r65 =  '65000000'x
     smf82r71 =  '71000000'x;   smf82r72 =  '72000000'x
     smf82r73 =  '73000000'x;   smf82r81 =  '81000000'x
     smf82r82 =  '82000000'x;   smf82r83 =  '83000000'x
     smf82r91 =  '91000000'x
```

```
   tmpr = "Normal completion"
if bitand(smf82rrc,smf82r00) = smf82r00 then do
   call printline repkom tmpr
   tmpr = "Machine failure"
if bitand(smf82rrc,smf82r10) = smf82r10 then do
   call printline repkom tmpr
   tmpr = "Invalid command"
if bitand(smf82rrc,smf82r21) = smf82r21 then do
   call printline repkom tmpr
   tmpr = "Command disabled"
if bitand(smf82rrc,smf82r22) = smf82r22 then do
   call printline repkom tmpr
   tmpr = "Request message length"
if bitand(smf82rrc,smf82r23) = smf82r23 then do
   call printline repkom tmpr
   tmpr = "Reserved field"
if bitand(smf82rrc,smf82r24) = smf82r24 then do
   call printline repkom tmpr
   tmpr = "Signature index"
if bitand(smf82rrc,smf82r25) = smf82r25 then do
   call printline repkom tmpr
   tmpr = "Authorization index"
if bitand(smf82rrc,smf82r26) = smf82r26 then do
   call printline repkom tmpr
   tmpr = "Domain index"
if bitand(smf82rrc,smf82r27) = smf82r27 then do
   call printline repkom tmpr
   tmpr = "Command extension"
if bitand(smf82rrc,smf82r28) = smf82r28 then do
   call printline repkom tmpr
   tmpr = "Facility disabled"
if bitand(smf82rrc,smf82r31) = smf82r31 then do
   call printline repkom tmpr
   tmpr = "PKSC domain disabled"
if bitand(smf82rrc,smf82r33) = smf82r33 then do
   call printline repkom tmpr
   tmpr = "CM not initialized"
if bitand(smf82rrc,smf82r41) = smf82r41 then do
   call printline repkom tmpr
   tmpr = "PRN not initialized"
if bitand(smf82rrc,smf82r42) = smf82r42 then do
   call printline repkom tmpr
   tmpr = "Configured command"
if bitand(smf82rrc,smf82r51) = smf82r51 then do
   call printline repkom tmpr
   tmpr = "Incorrect CMID"
if bitand(smf82rrc,smf82r61) = smf82r61 then do
   call printline repkom tmpr
   tmpr = "ASM violation"
if bitand(smf82rrc,smf82r62) = smf82r62 then do
```

```
    call printline repkom tmpr
    tmpr = "Incorrect TSN"
 if bitand(smf82rrc,smf82r63) = smf82r63 then do
    call printline repkom tmpr
    tmpr = "Authority modulus"
 if bitand(smf82rrc,smf82r64) = smf82r64 then do
    call printline repkom tmpr
    tmpr = "Signature verification"
 if bitand(smf82rrc,smf82r65) = smf82r65 then do
    call printline repkom tmpr
    tmpr = "ACM violation"
 if bitand(smf82rrc,smf82r71) = smf82r71 then do
    call printline repkom tmpr
    tmpr = "DXM violation"
 if bitand(smf82rrc,smf82r72) = smf82r72 then do
    call printline repkom tmpr
    tmpr = "DCM violation"
 if bitand(smf82rrc,smf82r73) = smf82r73 then do
    call printline repkom tmpr
    tmpr = "Pattern mismatch"
 if bitand(smf82rrc,smf82r81) = smf82r81 then do
    call printline repkom tmpr
    tmpr = "DH operand"
 if bitand(smf82rrc,smf82r82) = smf82r82 then do
    call printline repkom tmpr
    tmpr = "Configured modulus"
 if bitand(smf82rrc,smf82r83) = smf82r83 then do
    call printline repkom tmpr
    tmpr = "Key-part-queue full"
 if bitand(smf82rrc,smf82r91) = smf82r91 then do
    call printline repkom tmpr
    call printline sep
return
subtype13:
 /*-------------------------------------------------------*/
 /* Dynamic PKDS Update (Subtype 13) - written whenever   */
 /* the PKDS is updated by a dynamic PKDS update service  */
 /* (when an application uses the dynamic PKDS update     */
 /* services to change the PKDS)                          */
 /*-------------------------------------------------------*/
 upq = "PKDS update event:       "
 call printline Sub_line "DYNAMIC PKDS UPDATE (Subtype:" ||smf82sty||")"
 smf_pkds_bits      =     substr(subtype,Ø1,Ø4)    /* PKDS bits      */
 smf_pkds_bitsx     = c2x(substr(subtype,Ø1,Ø4))   /* PKDS bits (hex)*/
 smf_pkds_name      =     substr(subtype,Ø5,44)    /* PKDS name      */
 smf_pkds_key_label = c2x(substr(subtype,49,72))
                                                 /* PKDS being modified */
 msg = "Dynamic PKDS Update bits: "smf_pkds_bitsx; call printline msg
 /*-------------------------------------------------------*/
 /* Decode and print Dynamic PKDS Update bits             */
```

```
 /*-------------------------------------------------------*/
   smf_pkds_added   = '80000000'x    /* PKDS record added   */
   smf_pkds_updated = '40000000'x    /* PKDS record changed */
   smf_pkds_deleted = '20000000'x    /* PKDS record deleted */
 if bitand(smf_pkds_bits,smf_pkds_added) = smf_pkds_added     then
    call printline upq "PKDS record added"
 if bitand(smf_pkds_bits,smf_pkds_updated) = smf_pkds_updated then
    call printline upq "PKDS record changed"
 if bitand(smf_pkds_bits,smf_pkds_deleted) = smf_pkds_deleted then
    call printline upq "PKDS record deleted"
 msg = "PKDS name:                "smf_pkds_name;     call printline msg
 msg = "PKDS entry being modified: "smf_pkds_key_label;call printline
msg
 call printline sep
return
subtype14:
 /*-------------------------------------------------------*/
 /* PCI Cryptographic Coprocessor Clear Master Key        */
 /* Entry (Subtype 14) - written when a clear part is     */
 /* entered for any of the PCI Cryptographic Coprocessor  */
 /* master keys (ie whenever you use ICSF panels to       */
 /* update SYM-MK and ASYM-MK in the new master key       */
 /* register in a PCI Cryptographic Coprocessor)          */
 /*-------------------------------------------------------*/
 pci = "PCI Crypto Processor event:     "
 tit = "PCI CRYPTOGRAPHIC COPROCESSOR CLEAR MK"
 tit1 = "(Subtype:" ||smf82sty||")"
 call printline Sub_line tit tit1
 smf82aab =     substr(subtype,01,04)    /* Flag bytes              */
 smf82aax = c2x(substr(subtype,01,04))   /* Flag bytes (hex)        */
 smf82anv = c2x(substr(subtype,05,16))   /*New master key register VP*/
 smf82akv = c2x(substr(subtype,21,16)) /*Key part verification pattern*/
 smf82apn = c2d(substr(subtype,37,01))   /* PCI Crypto Processor no  */
 smf82asn =     substr(subtype,38,08) /*PCI Crypto Processor serial no*/
 smf82adm = c2d(substr(subtype,46,01)) /*PCI Crypto Coprocessor domain*/
 msg = "New master key register VP:      "smf82anv; call printline msg
 msg = "Key part verification pattern:   "smf82akv; call printline msg
 msg = "PCI Crypto Processor flag bytes: "smf82aax; call printline msg
 /*-------------------------------------------------------*/
 /* Decode and print PCI Crypto Processor flag bytes      */
 /*-------------------------------------------------------*/
   smf82asv = '80000000'x     /* Sym-Key NMK VP valid       */
   smf82aav = '40000000'x     /* Asym-Key NMK VP valid      */
   smf82ask = '20000000'x     /* Sym-Key Key part VP valid  */
   smf82aak = '10000000'x     /* Asym-Key Key part VP valid */
 fb80 = "Symmetric-Key NMK verification pattern is valid"
 fb40 = "Asymmetric-Key NMK verification pattern is valid"
 fb20 = "Symmetric-Key Key part verification pattern is valid"
 fb10 = "Asymmetric-Key Key part verification pattern is valid"
 if bitand(smf82aab,smf82asv) = smf82asv then
```

```
    call printline pci fb80
 if bitand(smf82aab,smf82aav) = smf82aav then
    call printline pci fb40
 if bitand(smf82aab,smf82ask) = smf82ask then
    call printline pci fb20
 if bitand(smf82aab,smf82aak) = smf82aak then
    call printline pci fb10
 msg = "PCI Crypto Processor number:     "smf82apn;  call printline msg
 msg = "PCI Crypto Processor serial num.:"smf82asn;  call printline msg
 msg = "PCI Crypto Coprocessor domain:   "smf82adm;  call printline msg
 call printline sep
return
subtype15:
 /*-------------------------------------------------------*/
 /* PCI Cryptographic Coprocessor Retained Key            */
 /* Create or Delete (Subtype 15) - written whenever you  */
 /* create or delete a retained private key in a PCI      */
 /* Cryptographic Coprocessor.                            */
 /*-------------------------------------------------------*/
 call printline Sub_line "RETAINED KEY CREATE/DELETE (Subtype:"
||smf82sty||")"
 smf82rkf =     substr(subtype,01,04)      /* Retained key flag bits */
 smf82rkx = c2x(substr(subtype,01,04))     /* Retained key flag (hex)*/
 smf82rkn = c2x(substr(subtype,05,64))     /* Label of retained key  */
 smf82rkp = c2x(substr(subtype,69,01))     /* Processor number       */
 smf82rks = c2x(substr(subtype,70,08))     /* Serial number          */
 smf82rdm = c2x(substr(subtype,78,01))     /* Current crypto domain  */
 msg = "Retained key flag bits: "smf82rkx; call printline msg
 /*-------------------------------------------------------*/
 /* Decode and print Retained key flag bits               */
 /*-------------------------------------------------------*/
   smf82rkc = '80000000'x           /* Retained key created */
   smf82rkd = '40000000'x           /* Retained key deleted */
   rke      = "Retained key event           : "
   s1580    = "Retained key created"
   s1540    = "Retained key deleted"
     if bitand(smf82rkf,smf82rkc) = smf82rkc then
        call printline rke s1580
     if bitand(smf82rkf,smf82rkd) = smf82rkd then
        call printline rke s1540
 msg = "Label of Retained private key: "smf82rkn; call printline msg
 msg = "PCI Crypto Coprocessor number: "smf82rkp; call printline msg
 msg = "PCI Crypto Coprocessor serial: "smf82rks; call printline msg
 msg = "PCI Crypto Coprocessor domain: "smf82rdm; call printline msg
 call printline sep
return
subtype16:
 /*-------------------------------------------------------*/
 /* PCI Cryptographic Coprocessor TKE Command Request or  */
 /* Reply (Subtype 16) - written whenever a TKE workstation */
```

```
 /* either issues a command request to a PCI Cryptographic  */
 /* or receives a reply response from a PCI Cryptographic   */
 /* Coprocessor.                                            */
  /*---------------------------------------------------------*/
 call printline Sub_line "PCI INTERFACE (Subtype:" ||smf82sty||")"
 smf82pfl =     substr(subtype,Ø1,Ø4)    /* Bits to indicate request/r*/
 smf82pfx = c2x(substr(subtype,Ø1,Ø4))   /* Bits to indicate request/r*/
 smf82ppn = c2d(substr(subtype,Ø5,Ø1))   /* Receiving PCI Index       */
 smf82psn =     substr(subtype,Ø6,Ø8)    /* Receiving PCI Serial No   */
 smf82pdm = c2d(substr(subtype,14,Ø1))   /* Receiving PCI Domain      */
 smf82pbl = c2d(substr(subtype,17,Ø4))   /*Parameter Block Len ("xxx")*/
 smf82pdl = c2d(substr(subtype,21,Ø4))   /* Parameter Data Block Len  */
                                            /* ("yyy")               */
 /*smf82pil                                Length of basic section    */
 smf82pbk = substr(subtype,25,smf82pbl+;   /* Parameter data blocks:  */
                  smf82pdl)                /*parameter block of length*/
                                           /* "xxx"(smf82pbl) followed*/
                                           /* by parameter data block */
                                           /*of length "yyy"(smf82pdl)*/
  msg = "Request / Reply bits: "smf82pfx; call printline msg
  msg = "Receiving PCI Index:  "smf82ppn; call printline msg
  msg = "Receiving PCI Serial: "smf82psn; call printline msg
  msg = "Receiving PCI Domain: "smf82pdm; call printline msg
  msg = "Parameter Block Len:  "smf82pbl; call printline msg
  msg = "Parameter Data Block: "smf82pdl; call printline msg
 /*---------------------------------------------------------*/
 /* Decode and print request/reply bits                     */
 /*---------------------------------------------------------*/
   s168Ø = "Request command"
   s164Ø = "Reply response"
   s16   = "Request / Reply: "
   smf82prq =  '8Ø000000'x       /* Processing Request CPRB */
   smf82prp =  '4Ø000000'x       /* Processing Reply CPRB   */
     if bitand(smf82pfl,smf82prq) = smf82prq then
        call printline s16 s168Ø
     if bitand(smf82pfl,smf82prp) = smf82prp then
        call printline s16 s164Ø
    msg = "Parameter data blocks: "smf82pbk
        call printline msg
        call printline sep
return
subtype17:
 /*---------------------------------------------------------*/
 /* PCI Cryptographic Coprocessor Timing (Subtype 17) -     */
 /* written periodically to provide some indication of PCI  */
 /* Cryptographic Coprocessor usage                         */
 /*---------------------------------------------------------*/
 call printline Sub_line "PCI TIMINGS (Subtype:" ||smf82sty||")"
 tit17 = "PCI Cryptographic Coprocessor Timing: "; call printline tit17
 smf82ctn = substr(st(c2x(SUBSTR(subtype,Ø1,Ø8))),12,32)
```

```
                                      /* CCP time before NQAP:       */
                                      /* time just before the PCI    */
                                      /* Cryptographic Coprocessor   */
                                      /* operation begins            */
 smf82ctd = substr(st(c2x(substr(subtype,Ø9,Ø8))),12,32)
                                      /* CCP time after DQAP:        */
                                      /* time just after PCI         */
                                      /* Cryptographic Coprocessor   */
                                      /* operation ends              */
 smf82ctw = substr(st(c2x(substr(subtype,17,Ø8))),12,32)
                                      /* CCP time after WAIT:        */
                                      /* time just after results     */
                                      /* have been communicated back */
                                      /* to caller address space     */
 smf82ctq = c2d(substr(subtype,25,Ø4))   /* CCP queue length:        */
                                      /* number of processes waiting */
                                      /* to submit work to the       */
                                      /* same PCI Cryptographic      */
                                      /* Coprocessor and domain,     */
                                      /* using the same reference    */
                                      /* number                      */
 smf82ctf = c2d(substr(subtype,29,Ø2))   /* Function code of service  */
 smf82ctx = c2d(substr(subtype,31,Ø1))   /* CCP Index                 */
 smf82cts =     substr(subtype,32,Ø8)    /* CCP Serial Number         */
 smf82ctm = c2d(substr(subtype,4Ø,Ø1))   /* Domain                    */
 smf82ctr = c2d(substr(subtype,41,Ø1))   /* Reference Number          */
 Resptime = dif(smf82ctn,smf82ctd)       /* CCP card response time    */
 Xfertime = dif(smf82ctd,smf82ctw)       /* CCP transfer time         */
 msg = "CCP time before NQAP:   "smf82ctn; call printline msg
 msg = "CCP time after DQAP:    "smf82ctd; call printline msg
 msg = "CCP card response time: "Resptime; call printline msg
 msg = "CCP time after WAIT:    "smf82ctw; call printline msg
 msg = "CCP transfer time:      "Xfertime; call printline msg
 msg = "CCP queue length:       "smf82ctq; call printline msg
 msg = "Service function code:  "smf82ctf; call printline msg
 msg = "CCP Index:              "smf82ctx; call printline msg
 msg = "CCP Serial Number:      "smf82cts; call printline msg
 msg = "CCP Domain:             "smf82ctm; call printline msg
 msg = "CCP Reference Number:   "smf82ctr; call printline msg
 call printline sep
 r82time.1 =  Resptime ||";"|| Xfertime
 "EXECIO 1 DISKW S82CSV(stem r82time.)"
  drop r82time.
return
subtype18:
  /*-------------------------------------------------------------*/
 /* PCI Cryptographic Coprocessor Configuration (Subtype 18): */
 /* written when a PCI Cryptographic Coprocessor is brought   */
 /* online or taken offline.                                  */
  /*-------------------------------------------------------------*/
```

```
 call printline Sub_line "CCP CONFIGURATION (Subtype:" ||smf82sty||")"
 smf82cgb = substr(subtype,Ø1,Ø4)       /* CCP configuration bits */
 smf82cxx = c2x(substr(subtype,Ø1,Ø4))  /* CCP configuration bits */
 smf82cgx = c2d(substr(subtype,Ø5,Ø1))  /* CCP index              */
 smf82cgs =     substr(subtype,Ø6,Ø8)   /* CCP serial number      */
 s181 ="Crypto Coprocessor:    "
 s182 ="CCP index:             "
 s183 ="CCP serial number:     "
 s184 ="CCP configuration bits:"
 msg  = s183 smf82cgs; call printline msg
 msg  = s184 smf82cxx; call printline msg
  /*-------------------------------------------------------------*/
 /* Decode and print CCP configuration bits                     */
  /*-------------------------------------------------------------*/
   smf82cgn = '80000000'x               /* CCP brought online */
   smf82cgf = '40000000'x               /* CCP taken offline  */
    if bitand(smf82cgb,smf82cgn  ) = smf82cgn then
      call printline s181 "PCI CC brought online"
    if bitand(smf82cgb,smf82cgf   ) = smf82cgf then
      call printline s181 "PCI CC brought offline"
      msg = s182 smf82cgx; call printline msg
      call printline sep
return
subtype19:
  /*-------------------------------------------------------------*/
 /* PCI X Cryptographic Coprocessor Timing (Subtype 19) -       */
 /* written  when a PCI X Cryptographic Coprocessor             */
 /* operation begins or ends.                                   */
  /*-------------------------------------------------------------*/
 call printline Sub_line "PCI X TIMINGS (Subtype:" ||smf82sty||")"
 tit17 = "PCI X Cryptographic Coprocessor Timing: "; call printline
tit17
 smf82xtn = substr(st(c2x(SUBSTR(subtype,Ø1,Ø8))),12,32)
                                      /* X CCP time before NQAP:     */
                                      /* time just before the PCI X  */
                                      /* Cryptographic Coprocessor   */
                                      /* operation begins            */
 smf82xtd = substr(st(c2x(substr(subtype,Ø9,Ø8))),12,32)
                                      /* CCP X time after DQAP:      */
                                      /* time just after PCI X       */
                                      /* Cryptographic Coprocessor   */
                                      /* operation ends              */
 smf82xtw = substr(st(c2x(substr(subtype,17,Ø8))),12,32)
                                      /* CCP X time after WAIT:      */
                                      /* time just after results     */
                                      /* have been communicated back */
                                      /* to caller address space     */
 smf82xtq = c2d(substr(subtype,25,Ø4))   /* CCP X queue length:      */
                                      /* number of processes waiting */
                                      /* to submit work to the       */
```

```
                                        /* same PCI Cryptographic      */
                                        /* Coprocessor and domain,     */
                                        /* using the same reference    */
                                        /* number                      */
 smf82xtf = c2d(substr(subtype,29,Ø2))   /* Function code of service  */
 smf82xtx = c2d(substr(subtype,31,Ø1))   /* PCI X CCP Index           */
 smf82xts =     substr(subtype,32,Ø8)    /* PCI X CCP Number          */
 smf82xtm = c2d(substr(subtype,4Ø,Ø1))   /* PCI X CCP domain          */
 smf82xtr = c2d(substr(subtype,41,Ø1))   /* PCI X CCP reference number*/
 Xresptime= dif(smf82xtn,smf82xtd)     /* PCI X CCP card response time*/
 Xxfertime= dif(smf82xtd,smf82xtw)     /* PCI X CCP transfer time     */
 msg = "CCP time before NQAP:   "smf82xtn;  call printline msg
 msg = "CCP time after DQAP:    "smf82xtd;  call printline msg
 msg = "CCP card response time: "Xresptime; call printline msg
 msg = "CCP time after WAIT:    "smf82xtw;  call printline msg
 msg = "CCP transfer time:      "Xxfertime; call printline msg
 msg = "CCP queue length:       "smf82xtq;  call printline msg
 msg = "Service function code:  "smf82xtf;  call printline msg
 msg = "CCP Index:              "smf82xtx;  call printline msg
 msg = "CCP Serial Number:      "smf82xts;  call printline msg
 msg = "CCP Domain:             "smf82xtm;  call printline msg
 msg = "CCP Reference Number:   "smf82xtr;  call printline msg
 call printline sep
return
Printline:
 /*-------------------------------------------------------------*/
 /*    Print each report line                                   */
 /*-------------------------------------------------------------*/
  PARSE arg lineout1
   "EXECIO 1 DISKW REPORT (STEM lineout)"
   if rc ¬= Ø then
     do
       say "printline RC =" RC
       exit rc
     end                                  /* end of printline */
Return
SMF: procedure
 /*-------------------------------------------------------------*/
 /* REXX - convert a SMF time to hh:mm:ss:hd format             */
 /*-------------------------------------------------------------*/
arg time
time1 = time % 1ØØ
 hh = time1 % 36ØØ;                    hh = right("Ø"||hh,2)
 mm = (time1 % 6Ø) - (hh * 6Ø);        mm = right("Ø"||mm,2)
 ss = time1 - (hh * 36ØØ) - (mm * 6Ø); ss = right("Ø"||ss,2)
 fr = time //  1ØØØ;                   fr = right("Ø"||fr,2)
rtime = hh||":"||mm||":"||ss||":"||fr
return rtime
ST:
 /*-------------------------------------------------------------*/
 /* TOD64 timestamp format converted. The BLSUXTOD proc is      */
```

```
 /* described in "z/OS V1R3 MVS IPCS Customization".          */
  /*-------------------------------------------------------------*/
arg todtime
If todtime    <> '0000000000000000000000' Then
  Do
    TOD_Value = X2C(todtime)
    Returned_Date = '-------------------------'
    address LINKPGM "BLSUXTOD TOD_Value Returned_Date"
  End
Else
  Returned_Date = '00000000000000000000'
Return Returned_Date
DIF:
  /*-------------------------------------------------------------*/
 /* Dif: REXX subroutine to find the difference between two   */
 /* timestamps                                                */
  /*-------------------------------------------------------------*/
 arg t.1,t.2
  do j=1 to 2
   _=arg(j)':0:0:0:'
    parse var _ h ":" m ":" s ":"
    s.j= h * 3600 + m * 60 + s
  end
diff=s.2-s.1+(s.1>s.2)*86400
dtime =right(diff%3600,2,0)':'|| ,
       right(diff//3600%60,2,0)":"|| ,
       translate(format(diff//3600//60,2),0,' ')
return diff
```

CONCLUDING REMARK

The ICSF SMF data reports are useful, but should be used with care. The data is basically only a sample of the activity against the crypto cards. They should be used in conjunction with:

- An understanding of the RMF reports for the same time intervals.
- The crypto hardware installed and the crypto functions they support.
- The applications using cryptographic functions, the functions and data sizes they use, and the hardware they exploit.

Mile Pekic
Systems Programmer (Serbia and Montenegro)

RACF 101 – a RACF quiz

RACF 101 is a regular column for newcomers to the RACF world. It presents basic RACF topics in a tutorial format. In this issue we will advance our RACF knowledge by taking a quiz designed to make you think like RACF does.

Just as a programmer, when writing a program, thinks like the computer does, so a RACF administrator should think like RACF does in determining the outcomes of various security checks.

The following quiz will help the beginner review some of the material covered in past issues, by looking at real-life RACF security scenarios.

Answers are given at the end. Give yourself one point for each correct answer. Anything more than five points is good.

Question 1

Consider the following access list for a dataset profile:

```
Userid/Group      Access
ABC               UPDATE
DEF               ALTER
```

If ABC and DEF are groups, and userid USER76 is a member of both of these groups, what effective access does USER76 have for this profile?

Question 2

Regarding Question 1, if, in addition to being connected to groups ABC and DEF, userid USER76 is also mentioned by userid in the access list, with READ access:

```
Userid/Group      Access
ABC               UPDATE
DEF               ALTER
USER76            READ
```

what access does userid USER76 now get?

Question 3

Regarding Question 2, what access would be granted to USER76 if, in addition to the above, the user also has the OPERATIONS attribute? (Remember, the OPERATIONS attribute by default allows the user ALTER access to all datasets.)

Question 4

True or false?

If a user is not in the access list of a profile, is not connected to any groups in the access list, the UACC for the profile is NONE, and the user does not have the OPERATIONS attribute, then the user does not get any access to the profile (user's access is NONE).

Question 5

If you use the RACF PERMIT command like this:

```
PERMIT profile.name, USER(XYZ)
```

and forget to specify the ACCESS(…) parameter, the command does not fail. Neither does it prompt you to specify the ACCESS(…) parameter. What is the default access in a PERMIT command if you forget to specify it?

Question 6

If you want to grant, say, READ access to all users in a profile, you can do it two ways – you can specify Universal Access of READ (UACC(READ) in the profile), or you can specify the userid '*' access(READ) in the access list of the profile.

What is the subtle difference in the access being granted by the two methods? Which method is preferable?

Question 7

If Global Access Checking grants ALTER access to a user for a profile, but in the access list of the profile itself the user has only READ access, which one prevails?

Question 8

In the recent past, RACF introduced a new user attribute called RESTRICTED. If a userid has the RESTRICTED attribute, what does it mean?

Answer 1

The effective access for userid USER76 is ALTER. RACF resolves this by granting the highest level of access the user is permitted by their various group connections; in this case, ALTER.

Answer 2

USER76 now gets READ access. If individual userid access is specified in an access list, then RACF will override the access it determines by checking various groups to which the user may belong.

RACF will always grant the access specified at the userid level, if one is specified. RACF does this to allow reduction (or increase) in access for a user, outside of the group-level access if the user belongs to a group in the access list. In the example, if the group ABC has 100 users connected to it, an override such as this may be necessary for one or two individuals in the group, and this allows that to happen.

Answer 3

USER76 will in this case get only READ access to the profile.

Even though USER76 has the OPERATIONS attribute, the explicit READ specified in this profile will override the ALTER access normally allowed and reduce the user's access to READ. This is RACF's way of removing some of the powers of users with the OPERATIONS attribute for important or classified datasets.

Answer 4

False. The profile can be in the Global Access Checking (GAC) Table, in which case the user will get whatever access

is specified there. If the Global Access Checking Table grants some access to a user, it overrides any restrictions that may be in place in the actual profile.

Answer 5

The default is READ access. This is one of the quirks in RACF. It would be nice if the access level was prompted, or if the command failed, forcing you to specify the access level in the PERMIT command.

Answer 6

The method of specifying UACC(READ) will grant READ access to *all* users, whether they are defined to RACF or not.

The method using the generic entry '*' with READ access in the access list will grant READ access to all users *defined* to RACF. It will not grant any access to undefined users.

For this reason, the second method is preferred. The UACC of most of your profiles should be NONE.

Answer 7

The user gets ALTER access. With Global Access Checking (GAC) active, the profile is not checked if the GAC entry allows access. The profile is, however, checked if GAC does *not* allow the required access.

Having entries in the GAC Table is not foolproof, should you, for instance, overlook this table. For this reason you should have a 'mirror' profile defined for each of your entries in the GAC Table that reflects the specifications in the GAC entry, to make you aware at a glance of what the profile allows.

Answer 8

If a userid has the RESTRICTED attribute, the userid will get only the access specifically granted to the userid via profiles' access lists.

The userid does not get general-purpose accesses specified in the Global Access Checking Table, or in UACCs, or via ID(*)

entries in access lists. These are bypassed for a RESTRICTED userid.

In a sense, this is the converse of the OPERATIONS attribute, where the userid has full (ALTER) access to all datasets unless specifically prohibited (or access reduced) in an access list.

Dinesh Dattani is an Independent Consultant specializing in mainframe security. He welcomes comments and feedback on this column. He can be contacted at dinesh123@rogers.com.

Dinesh Dattani
Independent Consultant
Toronto (Canada)

Querying and reporting the RACF database

The RACF database unload utility, IRRDBU00, enables users to create a sequential file from a RACF database. The sequential file can be used in several ways – viewed directly, used as input for installation-written programs, and manipulated with sort/merge utilities. It can also be uploaded to a relational database, such as DB2, to process complex inquiries and create user-defined reports.

The IRRDBU00 utility processes either a copy of the RACF database, a back-up RACF database, or the active RACF database. You must have UPDATE authority on the database.

The following JCL copies the data from a back-up RACF file (SYS1.RACF.BACK.MB) to the output sequential file (SYSADM.RACFDB.FLATFILE):

```
//SYSADMX JOB (1200-1205-00),CLASS=A,
//            MSGCLASS=X,NOTIFY=SYSADM,
//            MSGLEVEL=(1,1),USER=,REGION=4M
```

```
//* RACF DATABASE UNLOAD UTILITY
//UNLOAD EXEC PGM=IRRDBU00,PARM=NOLOCKINPUT
//* RACF INPUT DATASET
//INDD1 DD DISP=SHR,DSN=SYS1.RACF.BACK.MB
//* SEQUENTIAL OUTPUT DATA SET
//OUTDD DD DISP=SHR,DSN=SYSADM.RACFDB.FLATFILE
//SYSPRINT DD SYSOUT=*
/*
```

The PARM field on the EXEC statement must be specified. The parameters NOLOCKINPUT, LOCKINPUT, and UNLOCKINPUT are allowed. If you are running the unload utility IRRDBU00 against an active RACF database, LOCKINPUT is recommended. If you use NOLOCKINPUT on the active database, the data might contain inconsistencies.

The output (OUTDD) sequential file of IRRDBU00 is a dataset of variable (RECFM=VB) length records. The recommended record length (LRECL) is 4096 and block size is 27998.

The output file from the IRRDBU00 unload utility can be loaded into a relational database management system (DBMS) such as DB2, but first you must create one or more DB2 databases, one or more DB2 table spaces, DB2 tables, and the DB2 indexes, then load data into the tables, and, finally, reorganize and runstat the data in the tables (optional).

The Data Definition Language (DDL – CREATE DB2 objects) statements to define the relational presentation of the RACF database are in member SYS1.SAMPLIB(RACDBUTB).

The sample control statements for the DB2 load utility that map the output from the database unload utility (IRRDBU00) are in member SYS1.SAMPLIB(RACDBULD).

The following JCL loads the data from the output sequential file (SYSADM.RACF.FLATFILE) into DB2 tables:

```
//SYSADMX JOB (1200-1205-00),'',
//             NOTIFY=SYSADM,REGION=4M,
//             CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//LOAD     EXEC DSNUPROC,SYSTEM=DSNN,
//         UID='SYSADM.LOAD1',UTPROC='',COND=(4,LT)
//SYSREC   DD DSN=SYSADM.RACFDB.FLATFILE,DISP=SHR
//SYSUT1   DD DSN=DB2V7.SYSUT1.TABLE.TB1,
```

```
//          DISP=(MOD,DELETE,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),RLSE)
//SYSERR    DD DSN=DB2V7.SYSERR.TABLE.TB1,
//          DISP=(MOD,DELETE,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),RLSE)
//SYSMAP    DD DSN=DB2V7.SYSMAP.TABLE.TB1,
//          DISP=(MOD,DELETE,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),RLSE)
//SORTWKØ1  DD UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),,,ROUND)
//SORTWKØ2  DD UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),,,ROUND)
//SORTWKØ3  DD UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),,,ROUND)
//SORTWKØ4  DD UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),,,ROUND)
//SORTOUT   DD UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø),,,ROUND)
//SYSPRINT DD SYSOUT=*
//SYSIN     DD DSN=SYSADM.DB2.CNTL(RACDBULD),DISP=SHR
/*
```

The control statements in SYSIN dataset SYSADM.DB2.CNTL(RACDBULD) show only one DB2 table (USER_TSO_DATA):

```
INTO TABLE RACFID.USER_TSO_DATA
WHEN(1:4)='Ø22Ø'    (
   USTSO_NAME           POSITION(ØØ6:Ø13)   CHAR(8),
   USTSO_ACCOUNT        POSITION(Ø15:Ø54)   CHAR(4Ø),
   USTSO_COMMAND        POSITION(Ø56:135)   CHAR(8Ø),
   USTSO_DEST           POSITION(137:144)   CHAR(8),
   USTSO_HOLD_CLASS     POSITION(146:146)   CHAR(1),
   USTSO_JOB_CLASS      POSITION(148:148)   CHAR(1),
   USTSO_LOGON_PROC     POSITION(15Ø:157)   CHAR(8),
   USTSO_LOGON_SIZE     POSITION(159:168)   INTEGER EXTERNAL(1Ø),
   USTSO_MSG_CLASS      POSITION(17Ø:17Ø)   CHAR(1),
   USTSO_LOGON_MAX      POSITION(172:181)   INTEGER EXTERNAL(1Ø),
   USTSO_PERF_GROUP     POSITION(183:192)   INTEGER EXTERNAL(1Ø),
   USTSO_SYSOUT_CLASS   POSITION(194:194)   CHAR(1),
   USTSO_USER_DATA      POSITION(196:2Ø3)   CHAR(8),
   USTSO_UNIT_NAME      POSITION(2Ø5:212)   CHAR(8),
   USTSO_SECLABEL       POSITION(214:221)   CHAR(8)
   )
```

When you have RACF data in a DB2 environment, you can generate complex queries and user-defined reports. The REXX procedure (RDB0) generates some ISPF reports as follows:

- Find all users with the GLOBAL – SPECIAL authority.
- Check each of the dataset access list entries and verify that each user ID is a valid user or group ID.

- Find all the users connected to a particular group ID (USCON_GRP_ID) and list their names, user IDs, and authority within the group.
- Find all of the group connections that a particular user ID (USCON_NAME) has, and retrieve the authority within each group.
- Find all the RACF users defined with valid OMVS UIDs and valid OMVS GIDs. List the OMVS program name and home path associated with the users.
- Find all of the RACF users defined with valid OMVS UIDs.
- Find all of the RACF groups defined with valid OMVS GIDs.
- Find all of the RACF users having RRSF associations defined with a user ID on another system (USRSF_TARG_NODE). For these users, list their user ID, name, and user ID with which the association exists.
- Find all of the RACF users being managed by another user. For these users, list their user ID, name, the user ID managing them and the node of the managing user ID.
- Find all DCE users and display their RACF user IDs, DCE UUIDs, DCE home cells, DCE home cell UUIDs, OpenEdition UIDs, and their names.
- Find all the DCE users without a corresponding profile in the DCEUUIDS class where the RACF user ID is in the APPLDATA field.
- Find all the DCE users without a corresponding OMVS segment.

RDB0 – REXX DRIVER PROCEDURE

```
/* REXX */
/* trace r */
 zpfctl = 'OFF'
 /* The subsystem name                                  */
 db2='DSNN'
```

```
address ispexec 'vput (zpfctl) profile'
CUR='F1'
address ispexec "display panel(rdbpØ) cursor("CUR")"
do while rc=Ø
   if kurs='F1' then do
        Call rdb1 db2 'F1'
        CUR='F1'
   end
   if kurs='F2' then do
        Call rdb1 db2 'F2'
        CUR='F2'
   end
   if kurs='F3' then do
        Call rdb1 db2 'F3'
        CUR='F3'
   end
   if kurs='F4' then do
        Call rdb1 db2 'F4'
        CUR='F4'
   end
   if kurs='F5' then do
        Call rdb1 db2 'F5'
        CUR='F5'
   end
   if kurs='F6' then do
        Call rdb1 db2 'F6'
        CUR='F6'
   end
   if kurs='F7' then do
        Call rdb1 db2 'F7'
        CUR='F7'
   end
   if kurs='F8' then do
        Call rdb1 db2 'F8'
        CUR='F8'
   end
   if kurs='F9' then do
        Call rdb1 db2 'F9'
        CUR='F9'
   end
   if kurs='F1Ø' then do
        Call rdb1 db2 'F1Ø'
        CUR='F1Ø'
   end
   if kurs='F11' then do
        Call rdb1 db2 'F11'
        CUR='F11'
   end
   if kurs='F12' then do
        Call rdb1 db2 'F12'
```

```
         CUR='F12'
      end
      address ispexec "display panel(rdbpØ) cursor("CUR")"
 end
 exit
```

RDB1 – REXX DETAIL REPORT PROCEDURE

```
/* REXX */
/* List all RACF DB2 tables                                  */
/* trace r */
 arg db2 fs
 if db2 = ' ' then db2 ='DSNN'
 zpfctl = 'OFF'
 Y=MSG("OFF")
 address ispexec 'vput (zpfctl) profile'
 hitem=''
 /* DSNREXX Language Support                                  */
 Address TSO "SUBCOM DSNREXX"
 IF RC THEN
 S_RC = RXSUBCOM(ADD,DSNREXX,DSNREXX)
 SSID = db2
 ADDRESS DSNREXX "CONNECT" SSID
 /* User with the GLOBAL-SPECIAL authority                    */
 if fs='F1' then do
    SQLSTMT= "SELECT USBD_NAME                                      ",
    "          FROM RACFID.USER_BD                                  ",
    "          WHERE USBD_SPECIAL='Y'                               ",
    "          ORDER BY USBD_NAME                                   ",
    "          WITH UR                                              "
    Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
    Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
    Address DSNREXX "EXECSQL OPEN C1"
    Address DSNREXX "EXECSQL FETCH C1 INTO :hitem"
    address ispexec 'tbcreate "flist" names(hitem)'
    do while(sqlcode=Ø)
       address ispexec 'tbadd "flist"'
       Address DSNREXX "EXECSQL FETCH C1 INTO :hitem"
    end
    Address DSNREXX "EXECSQL CLOSE C1"
    address ispexec 'tbtop "flist"'
    title='GLOBAL-SPECIAL authority'
    address ispexec 'addpop row(1) column(2Ø)'
    address ispexec 'tbdispl "flist" panel(RDBP1)'
    address ispexec rempop all
    address ispexec 'tbend "flist"'
 end
 /* User with the GLOBAL-SPECIAL authority                    */
 if fs='F2' then do
```

```
   SQLSTMT= "SELECT DSACC_NAME                                          ",
   "               ,DSACC_AUTH_ID                                       ",
   "               ,DSACC_ACCESS                                        ",
   "               ,DSACC_ACCESS_CNT                                    ",
   "         FROM RACFID.DS_ACCESS X                                    ",
   "         WHERE NOT EXISTS                                           ",
   "               ( SELECT *                                           ",
   "                 FROM RACFID.AUTH_IDS                               ",
   "                 WHERE X.DSACC_AUTH_ID=AUTHID_NAME                  ",
   "                )                                                   ",
   "           AND X.DSACC_AUTH_ID^='*'                                 ",
   "         ORDER BY 1                                                 ",
   "         WITH UR                                                    "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hitem1, :hitem2, :hitem3, :hitem4"
   address ispexec 'tbcreate "flist",
           names(hitem1 hitem2 hitem3 hitem4)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hitem1, :hitem2, :hitem3, :hitem4"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   title='Check each of the dataset authorities'
   address ispexec 'tbdispl "flist" panel(RDBP2)'
   address ispexec 'tbend "flist"'
end
/* Information about user group connections              */
if fs='F3' then do
   title='Information about user group connections'
   address ispexec "display panel(rdbp3) cursor("grpid")"
   TOPF3:
   SQLSTMT= "SELECT T2.USCON_NAME                                       ",
   "               ,T1.USBD_NAME                                        ",
   "               ,T1.USBD_PROGRAMMER                                  ",
   "               ,T2.USCON_GRP_SPECIAL                                ",
   "               ,T2.USCON_GRP_OPER                                   ",
   "               ,T2.USCON_GRP_AUDIT                                  ",
   "               ,T2.USCON_REVOKE                                     ",
   "               ,VALUE(CHAR(T2.USCON_REVOKE_DATE),' ')               ",
   "         FROM  RACFID.USER_BD T1,                                   ",
   "               RACFID.USER_CONNECT_DATA T2                          ",
   "         WHERE T2.USCON_GRP_ID ='"grpid"'                           ",
   "           AND T2.USCON_NAME = T1.USBD_NAME                         ",
   "         ORDER BY T2.USCON_NAME                                     ",
   "              ,T1.USBD_NAME                                         ",
```

```
   "         WITH UR                                                "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6,:hv7,:hv8"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4 hv5 hv6 hv7 hv8)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6,:hv7,:hv8"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP3)'
   rrc=rc
   address ispexec 'tbend "flist"'
   if rrc=8
   then Exit
   else Signal TOPF3
end
/* Information about a user connection to a group        */
if fs='F4' then do
   title='Information about a user connection to a group'
   address ispexec "display panel(rdbp4) cursor("uscname")"
   TOPF4:
   SQLSTMT= "SELECT USCON_NAME                                      ",
   "               ,USCON_GRP_SPECIAL                               ",
   "               ,USCON_GRP_ID                                    ",
   "               ,USCON_GRP_OPER                                  ",
   "               ,USCON_GRP_AUDIT                                 ",
   "               ,USCON_REVOKE                                    ",
   "               ,VALUE(CHAR(USCON_REVOKE_DATE),' ')              ",
   "               ,VALUE(CHAR(USCON_RESUME_DATE),' ') ",
   "         FROM  RACFID.USER_CONNECT_DATA                         ",
   "         WHERE USCON_NAME='"uscname"'                           ",
   "         ORDER BY USCON_GRP_ID                                  ",
   "         WITH UR                                                "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6,:hv7,:hv8"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4 hv5 hv6 hv7 hv8)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6,:hv7,:hv8"
```

```
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP4)'
   rrc=rc
   address ispexec 'tbend "flist"'
   if rrc=8
   then Exit
   else Signal TOPF4
end
/* Find the users that are defined with valid OMVS UID and OMVS GID */
if fs='F5' then do
   title='Users with valid OMVS UID and OMVS GID'
   address ispexec "display panel(rdbp5) cursor("uscname")"
   TOPF5:
   SQLSTMT= "SELECT USBD_NAME                                          ",
   "               ,GPOMVS_GID                                         ",
   "               ,GPOMVS_NAME                                        ",
   "               ,USBD_PROGRAMMER                                    ",
   "               ,USCON_NAME                                         ",
   "               ,USCON_GRP_ID                                       ",
   "         FROM  RACFID.USER_BD,                                     ",
   "               RACFID.USER_CONNECT_DATA,                           ",
   "               RACFID.GROUP_OMVS_DATA                              ",
   "         WHERE GPOMVS_GID IS NOT NULL                              ",
   "           AND USBD_NAME ='"uscname"'                              ",
   "           AND USCON_NAME = USBD_NAME                              ",
   "           AND USCON_GRP_ID = GPOMVS_NAME                          ",
   "         ORDER BY USBD_NAME                                        ",
   "         WITH UR                                                   "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4 hv5 hv6)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5,:hv6"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP5)'
   rrc=rc
   address ispexec 'tbend "flist"'
   if rrc=8
   then Exit
   else Signal TOPF5
```

```
end
/* Find the users that are defined with valid OMVS UID */
if fs='F6' then do
   title='Find the users that are defined with valid OMVS UID'
   SQLSTMT= "SELECT USOMVS_UID                                          ",
   "               ,USBD_NAME                                           ",
   "               ,USBD_PROGRAMMER                                     ",
   "         FROM  RACFID.USER_BD,                                      ",
   "               RACFID.USER_OMVS_DATA                                ",
   "         WHERE USOMVS_NAME = USBD_NAME                              ",
   "           AND USOMVS_UID IS NOT NULL                               ",
   "         ORDER BY USOMVS_UID                                        ",
   "         WITH UR                                                    "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP6)'
   address ispexec 'tbend "flist"'
end
/* Find the users that are defined with valid OMVS GID */
if fs='F7' then do
   title='Find the users that are defined with valid OMVS GID'
   SQLSTMT= "SELECT GPOMVS_GID                                          ",
   "               ,GPOMVS_NAME                                         ",
   "          FROM RACFID.GROUP_OMVS_DATA                               ",
   "         WHERE GPOMVS_GID IS NOT NULL                               ",
   "         ORDER BY GPOMVS_GID                                        ",
   "         WITH UR                                                    "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2"
```

```
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP7)'
   address ispexec 'tbend "flist"'
end
/* Find the users that have RRSF defined on another system */
if fs='F8' then do
   title='Find the users that have RRSF defined on another system'
   address ispexec "display panel(rdbp8) cursor("usrnode")"
   TOPF8:
   SQLSTMT= " SELECT USRSF_NAME                                    ",
   "                ,USBD_PROGRAMMER                               ",
   "                ,USRSF_TARG_USER_ID                            ",
   "                ,USRSF_TARG_NODE                               ",
   "          FROM  RACFID.USER_RRSF_DATA,                         ",
   "                RACFID.USER_BD                                 ",
   "          WHERE USRSF_NAME = USBD_NAME                         ",
   "            AND USRSF_NAME IS NOT NULL                         ",
   "            AND USRSF_TARG_NODE = 'EMDSYS  '                   ",
   "          ORDER BY USRSF_NAME                                  ",
   "         WITH UR                                               "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP8)'
   rrc=rc
   address ispexec 'tbend "flist"'
   if rrc=8
   then Exit
   else Signal TOPF8
end
/* Find the users that are being managed by another user */
if fs='F9' then do
   title='Find the users that are being managed by another user'
   SQLSTMT= "SELECT USRSF_NAME                                     ",
   "               ,USBD_PROGRAMMER                                ",
   "               ,USRSF_TARG_USER_ID                             ",
   "               ,USRSF_TARG_NODE                                ",
```

```
   "               ,USRSF_MANAGED                                  ",
   "         FROM  RACFID.USER_RRSF_DATA,                          ",
   "               RACFID.USER_BD                                  ",
   "         WHERE USRSF_NAME = USBD_NAME                          ",
   "           AND USRSF_NAME IS NOT NULL                          ",
   "           AND USRSF_MANAGED = 'Y'                             ",
   "         ORDER BY USRSF_NAME                                   ",
   "         WITH UR                                               "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4 hv5)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4,:hv5"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP9)'
   address ispexec 'tbend "flist"'
end
/* DCE information about users */
if fs='F1Ø' then do
   title='DCE information about users'
   SQLSTMT= " SELECT USDCE_NAME,                                   ",
   "             USBD_PROGRAMMER,                                  ",
   "             USDCE_UUID,                                       ",
   "             USOMVS_UID                                        ",
   "          FROM                                                 ",
   "             RACFID.USER_DCE_DATA,                             ",
   "             RACFID.USER_OMVS_DATA,                            ",
   "             RACFID.USER_BD                                    ",
   "          WHERE USDCE_NAME = USOMVS_NAME                       ",
   "            AND USBD_NAME = USOMVS_NAME                        ",
   "          ORDER BY USDCE_NAME                                  ",
   "         WITH UR                                               "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3 hv4)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
```

```
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3,:hv4"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP1Ø)'
   address ispexec 'tbend "flist"'
end
/* DCE information about users without profile in the DCEUUIDS */
if fs='F11' then do
   title='Users without profile in the DCEUUIDS'
   SQLSTMT= "SELECT USDCE_NAME,                                    ",
   "            USDCE_NAME,                                        ",
   "            USDCE_UUID,                                        ",
   "            USDCE_HOMECELL                                     ",
   "         FROM RACFID.USER_DCE_DATA                             ",
   "         WHERE                                                 ",
   "         USDCE_NAME NOT IN                                     ",
   "                ( SELECT GRBD_APPL_DATA                        ",
   "                   FROM RACFID.GENR_BD                         ",
   "                 )                                             ",
   "         ORDER BY  USDCE_NAME                                  ",
   "         WITH UR                                               "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX,
   "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
   address ispexec 'tbcreate "flist",
           names(hv1 hv2 hv3)'
   do while(sqlcode=Ø)
      address ispexec 'tbadd "flist"'
      Address DSNREXX,
      "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbtop "flist"'
   address ispexec 'tbdispl "flist" panel(RDBP11)'
   address ispexec 'tbend "flist"'
end
/* DCE information about users without corresponding OMVS segment */
if fs='F12' then do
   title='Users without corresponding OMVS segment'
   SQLSTMT= "SELECT USDCE_NAME,                                    ",
   "            USDCE_NAME,                                        ",
   "            USDCE_UUID,                                        ",
   "            USDCE_HOMECELL                                     ",
   "         FROM RACFID.USER_DCE_DATA                             ",
   "         WHERE                                                 ",
   "            USDCE_NAME NOT IN                                  ",
   "                ( SELECT USOMVS_NAME                           ",
```

```
    "                   FROM RACFID.USER_OMVS_DATA                  ",
    "                 )                                             ",
    "          ORDER BY  USDCE_NAME                                 ",
    "          WITH UR                                              "
    Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
    Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
    Address DSNREXX "EXECSQL OPEN C1"
    Address DSNREXX,
    "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
    address ispexec 'tbcreate "flist",
            names(hv1 hv2 hv3)'
    do while(sqlcode=Ø)
       address ispexec 'tbadd "flist"'
       Address DSNREXX,
       "EXECSQL FETCH C1 INTO :hv1,:hv2,:hv3"
    end
    Address DSNREXX "EXECSQL CLOSE C1"
    address ispexec 'tbtop "flist"'
    address ispexec 'tbdispl "flist" panel(RDBP11)'
    address ispexec 'tbend "flist"'
 end
 ADDRESS DSNREXX "DISCONNECT"
 Exit
```

RDBP0 – MAIN MENU

```
)attr default(%+_)
   [ type (output) intens(low)  color(green) caps(off)
   # type (output) intens(low)  color(white) caps(off)
   _ type (input)  intens(low)  color(yellow) caps(off) pad('_')
   + type (text)   intens(low)  color(green)
   / type (text)   intens(low)  color(yellow)
   ~ type (text)   intens(high) color(turquoise)
   @ type (text)   intens(high) color(red)   caps(off) hilite(reverse)
)body window(8Ø,24) expand ($$)
/.....................................................................
                          + @ The RACF Database +
%Command ===>_zcmd                                                   +
/.....................................................................
+
_z[rfld1                                                             +
_z[rfld2                                                             +
_z[rfld3                                                             +
_z[rfld4                                                             +
_z[rfld5                                                             +
_z[rfld6                                                             +
_z[rfld7                                                             +
_z[rfld8                                                             +
_z[rfld9                                                             +
```

```
_z[rfld1Ø                                                              +
_z[rfld11                                                              +
_z[rfld12                                                              +
+
/.....................................................................
+
/PF1 - Help+     #msg                                                  +
/PF3 - End +                                          ~Jun 2Ø05,"ZB"
)init
  .ZVARS = '(f1 f2 f3 f4 f5 f6 f7 f8 f9 f1Ø f11 f12)'
  &rfld1 = 'User with the GLOBAL-SPECIAL authority'
  &rfld2 = 'Check each of the dataset authorities'
  &rfld3 = 'Information about user group connections'
  &rfld4 = 'Information about a user connection to a group'
  &rfld5 = 'Find the users that are defined with valid OMVS UID and OMVS
GID'
  &rfld6 = 'Find the users that are defined with valid OMVS UID'
  &rfld7 = 'Find the users that are defined with valid OMVS GID'
  &rfld8 = 'Find the users that have RRSF defined on another system'
  &rfld9 = 'Find the users that are being managed by another user'
  &rfld1Ø= 'DCE information about users'
  &rfld11= 'DCE information about users without profile in the DCEUUIDS'
  &rfld12= 'DCE information about users without corresponding OMVS
segment'
  &msg = 'Place cursor on choice and press <Enter>'
  IF (&kurs = F1,rfld1)
      .attr (rfld1) = 'color (yellow) caps(on)'
  IF (&kurs = F2,rfld2)
      .attr (rfld2) = 'color (yellow) caps(on)'
  IF (&kurs = F3,rfld3)
      .attr (rfld3) = 'color (yellow) caps(on)'
  IF (&kurs = F4,rfld4)
      .attr (rfld4) = 'color (yellow) caps(on)'
  IF (&kurs = F5,rfld5)
      .attr (rfld5) = 'color (yellow) caps(on)'
  IF (&kurs = F6,rfld6)
      .attr (rfld6) = 'color (yellow) caps(on)'
  IF (&kurs = F7,rfld7)
      .attr (rfld7) = 'color (yellow) caps(on)'
  IF (&kurs = F8,rfld8)
      .attr (rfld8) = 'color (yellow) caps(on)'
  IF (&kurs = F9,rfld9)
      .attr (rfld9) = 'color (yellow) caps(on)'
  IF (&kurs = F1Ø,rfld1Ø)
      .attr (rfld1Ø) = 'color (yellow) caps(on)'
  IF (&kurs = F11,rfld11)
      .attr (rfld11) = 'color (yellow) caps(on)'
  IF (&kurs = F12,rfld12)
      .attr (rfld12) = 'color (yellow) caps(on)'
  .HELP = rdbhØ
```

```
)proc
  &kurs = .CURSOR
  if (.pfkey = pfØ3) &pf3 = exit
)end
```

RDBP1 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(low ) caps(off) just(asis ) color(turquoise)
)Body WINDOW(5Ø,2Ø)
$title                                          +
%Command =>_zcmd                                +
%Scroll ===>_amt +
+-----------------------------------------------+
#User    +
)Model
]z       +
)Init
  .ZVARS = '(hitem)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP2 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                +
%Command =>_zcmd                                        %Scroll =>_amt +
+---------------------------------------------------------------------+
                                     # DSACC  #DSACC   #  DSACC   +
#DSACC NAME                          #AUTH ID #ACCESS  #ACCESS CNT+
)Model
]z                                   ]z       ]z       ]z         +
)Init
  .ZVARS = '(hitem1 hitem2 hitem3 hitem4)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP3 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   { type(text)   intens(high) caps(off) color(yellow)
   [ type(input)  intens(high) caps(on ) color(red) just(left )
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                 +
%Command =>_zcmd                                          %Scroll =>_amt +
{USCON GRP ID[grpid   +
+-------------------------------------------------------------------------+
                                    # USCON #USCON#USCON#USCON # USCON   +
#USCON NAME#USBD NAME#USBD PROGRAMMER    #SPECIAL#OPER #AUDIT#REVOKE#
DATE    +
)Model
]z         ]z        ]z              ]z      ]z    ]z    ]z     ]z     +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4 hv5 hv6 hv7 hv8)'
  &amt = PAGE
)Reinit
)Proc
  VPUT (grpid) PROFILE
)End
```

RDBP4 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   { type(text)   intens(high) caps(off) color(yellow)
   [ type(input)  intens(high) caps(on ) color(red) just(left )
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title
%Command =>_zcmd                                          %Scroll =>_amt
{USCON NAME[uscname +
+-------------------------------------------------------------------------
    #USCON GRP#USCON   # USCON  #USCON GRP#USCON #   USCON   #   USCON
#USCON NAME# SPECIAL #GRP ID  #GRP OPER#  AUDIT  #REVOKE#REVOKE
DATE#RESUME DATE
)Model
]z        ]z       ]z      ]z      ]z       ]z    ]z         ]z        +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4 hv5 hv6 hv7 hv8)'
  &amt = PAGE
)Reinit
)Proc
```

```
  VPUT (uscname) PROFILE
)End
```

RDBP5 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   { type(text)   intens(high) caps(off) color(yellow)
   [ type(input)  intens(high) caps(on ) color(red) just(left )
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title
%Command =>_zcmd                                              %Scroll =>_amt
{USBD NAME[uscname +
+-------------------------------------------------------------------------
#USBD NAME#GPOMVS GID#GOMVS NAME#USBD PROGRAMMER     #USCON NAME#USCON
GRP ID
)Model
]z       ]z        ]z        ]z                 ]z         ]z         +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4 hv5 hv6)'
  &amt = PAGE
)Reinit
)Proc
  VPUT (uscname) PROFILE
)End
```

RDBP6 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title
+
%Command =>_zcmd                                          %Scroll =>_amt +
+-------------------------------------------------------------------------+
#USOMVS UID#USBD NAME#USBD PROGRAMMER     +
)Model
]z         ]z        ]z                  +
)Init
  .ZVARS = '(hv1 hv2 hv3)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP7 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                  +
%Command =>_zcmd                                          %Scroll =>_amt +
+-----------------------------------------------------------------------+
#GPOMVS GID#GPOMVS NAME+
)Model
]z         ]z          +
)Init
  .ZVARS = '(hv1 hv2)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP8 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   { type(text)   intens(high) caps(off) color(yellow)
   [ type(input)  intens(high) caps(on ) color(red) just(left )
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title
%Command =>_zcmd                                          %Scroll =>_amt
{USRSF TARG NODE[usrnode +
+-----------------------------------------------------------------------
#USRSF NAME#USBD PROGRAMMER     #USRSF USER ID#USRSF TARG NODE+
)Model
]z         ]z                   ]z          ]z                +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4)'
  &amt = PAGE
)Reinit
)Proc
  VPUT (usrnode) PROFILE
)End
```

RDBP9 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
```

```
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                    +
%Command =>_zcmd                                          %Scroll =>_amt +
+-------------------------------------------------------------------------+
                                                        #USRSF TARG+
#USRSF NAME#USBD PROGRAMMER     #USRSF TARG USER ID#   NODE   #USRSF
MANAGED+
)Model
]z        ]z                  ]Z                ]Z        ]Z           +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4 hv5)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP10 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                    +
%Command =>_zcmd                                          %Scroll =>_amt +
+-------------------------------------------------------------------------+
#USDCE NAME#USBD PROGRAMMER     #USDCE UUID                 #USOMVS UID+
)Model
]z         ]z                   ]Z                          ]Z         +
)Init
  .ZVARS = '(hv1 hv2 hv3 hv4)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBP11 – PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(output) intens(high) caps(on ) just(left )
)Body
$title                                                                    +
%Command =>_zcmd                                          %Scroll =>_amt +
+-------------------------------------------------------------------------+
#USDCE NAME#USDCE UUID                       #USDCE HOMECELL            +
)Model
```

```
]z         ]Z                                  ]Z                          +
)Init
  .ZVARS = '(hv1 hv2 hv3)'
  &amt = PAGE
)Reinit
)Proc
)End
```

RDBH0 – HELP PANEL0

```
)attr default(/+")
  @ type(text) intens(high) color(red) caps(off) hilite(reverse)
  ~ type(text) intens(high) color(red) caps(off)
  } type(text) intens(high) color(white) hilite(reverse)
  { type(text) intens(high) color(white)
  [ type(text) intens(high) color(white) hilite(uscore)
  ( type(text) intens(high) color(green)
  ) type(text) intens(high) color(pink)
  \ type(text) intens(high) color(blue)
  ] type(text) intens(high) color(yellow)
)body expand ($$)
+  $_$@ H e l p +$_$
+
+[Panel Explanation:+                                              1 of 2
+
+(The~RDBØ(procedure shows some{RACF Information(from RACF database.   +
+(Sample queries uses output data from the Unload Utility{IRRDBUØØ.(   +
+(The REXX precedure generates the following SQL reports.              +
+                                                                      +
+]Find all of the user with the GLOBAL-SPECIAL authority.              +
+                                                                      +
+(Check each of the dataset access list entries and verify that each   +
+(user ID is a valid user or group ID.                                 +
+
+]Find all of the users connected to a particular department, and      +
+]list their names, user IDs, and authority within the group.         +
+                                                                      +
+(Find all of the group connections that a particular user ID has,     +
+(and retrieve the authority within each group.                        +
+
+]Find all of the RACF users that are defined with valid OMVS UIDs     +
+]and valid OMVS GIDs.                                                 +
+                                                                      +
}Enter: Next Panel+                                          }F3: Return
)init
  .HELP = rdbh1
)proc
  .HELP = rdbh1
  &zcont = rdbh1
)end
```

RDBH1 – HELP PANEL1

```
)attr default(/+")
  @ type(text) intens(high) color(red) caps(off) hilite(reverse)
  ~ type(text) intens(high) color(red) caps(off)
  } type(text) intens(high) color(white) hilite(reverse)
  { type(text) intens(high) color(white)
  [ type(text) intens(high) color(white) hilite(uscore)
  ( type(text) intens(high) color(green)
  ) type(text) intens(high) color(pink)
  \ type(text) intens(high) color(blue)
  ] type(text) intens(high) color(yellow)
)body expand ($$)
+  $_$@ H e l p +$_$
+
+[Panel Explanation:+                                              2 of 2
+
+]Find all of the RACF users that are defined with valid OMVS UIDs.    +
+                                                                      +
+(Find all of the RACF groups that are defined with valid OMVS GIDs.   +
+
+]Find all of the RACF users that have RRSF associations defined with  +
+]a user ID on another system.                                         +
+                                                                      +
+(Find all of the RACF users that are being managed by another user.   +
+
+]Find all of DCE users and display their RACF user IDs, DCE UUIDs,
+
+]DCE home cells, DCE home cell UUIDs, OpenEdition UIDs, and their
names.     +
+                                                                     +
+(Find all of the DCE users who do not have a corresponding profile in +
+(the DCEUUIDS class where the RACF user ID is in the APPLDATA field.  +
+
+]Find all of the DCE users who do not have a corresponding OMVS
segment.     +
+
+
}Enter: Previous Panel+                                        }F3: Return
)init
  .HELP = rdbh1
)proc
  .HELP = rdbhØ
  &zcont = rdbhØ
)end
```

Bernard Zver (bernard.zver@informatika.si)
DBA
Informatica (Slovenia)

RACF and encryption

One way to ensure that mainframe data is secure is through the use of cryptography. The data is encrypted so that it cannot be decrypted without access to a key that specifies how it has been encrypted. If RACF security is penetrated, the hacker will be unable to understand the information they have accessed.

Cryptography provides three important security functions – confidentiality, integrity, and non-repudiation. What this means is that the data can't be read or altered except by authorized users. Also, the identity of the sender of the data is known.

Cryptography comes in two forms – hardware and software. IBM's hardware solution is its Cryptographic Coprocessor, which is coupled with a z/OS Integrated Cryptographic Service Facility (ICSF) component.

RACF uses the OCEP (Open Cryptographic Enhanced Plug-ins, which are part of the OS/390 Security Server) to act as a limited certificate authority. OCEP provides an application interface for managing server certificates and helping to protect server private keys. Applications must comply with CDSA (Common Data Security Architecture) standard interfaces in order to use OCEP.

The ideal software solution would comply with RFC2440 (OpenPGP) standards to ensure that it interoperates with encryption products on other platforms. The product should also support well-known encryption algorithms (such as AES, DES, Triple-DES, etc). It should allow the use of private and public keys. And it should ensure data integrity by offering cyclic redundancy checking, message digest algorithms, and message authentication codes. A digital signature can be used to ensure that the sender of the data can be positively identified.

Basically, RACF is no longer enough in terms of mainframe

security. Additional protection is required and cryptography is a useful methodology.

Stephen Hare
Security Consultant (UK)

If you have an article, or an idea for an article, that you think would be suitable for inclusion in *RACF Update*, please send it to the editor, Trevor Eddolls, at TrevorE@xephon.com.

Beta Systems Software AG has added two modules to its SAM Jupiter IdM suite. SAM Virtual Directory and SAM Data Synchronization Engine provide a way to create an IdM architecture based on SAM Jupiter. The components make use of technology from Maxware AS.

SAM Jupiter automates the life-cycle management of IT users, passwords, access rights, and security settings. The product offers standard connectors for the management of systems such as RACF, ACF2, TopSecret, Windows, SAP, or Unix.

With the SAM Data Synchronization Engine, policies can be applied uniformly in order to synchronize data between primary sources, for example between HR systems, the SAM provisioning solution and any existing directories in which user information is held. In addition the SAM Virtual Directory provides middleware guaranteeing simple access to elements of user information from different databases and directories.

For further information contact:
URL: www.betasystems.com/e_beta.nsf/(Docs)/0300000205.

* * *

IBM has announced Tivoli Security Administrator for RACF R1, which helps to improve RACF administrator productivity with new tools and views, reduces training costs with a Java-based GUI or new ISPF panels, improves audit readiness and troubleshooting capabilities with flexible queries against the RACF database, minimizes repetitive tasks by providing one interface to multiple RACF databases, increases flexibility by enabling delegated security administration, and improves application access to RACF data.

For further information contact:
URL: www.ibm.com/software/tivoli/products/security-admin-racf.

* * *

Vanguard Integrity Professionals has announced an agreement with IBM in which IBM will resell Vanguard's security software products for the eServer zSeries. Vanguard's offerings will add zSeries support to IBM's Tivoli security software portfolio. The agreement includes Vanguard Administrator, Advisor, Analyzer, Enforcer, and SecurityCenter. These provide a toolset for administration, reporting, auditing, and intrusion detection for RACF.

Vanguard has also announced Verion 6.1 of Vanguard Security Solutions, with enhancements for Vanguard Administrator, Advisor, Analyzer, Enforcer, and INCompliance. Included in the release is ez/AccessControl for Windows, a single point of control for routing all access requests to a drive, file, or directory on a Microsoft Windows machine through RACF. This allows companies to extend the security authorization and auditing capabilities of the mainframe to their Windows 2000 and 2003 systems.

For further information contact:
URL: www.go2vanguard.com/docs/marketing/press_releases/PR_2005_6.1_Release.pdf.

* * *