36

RACF

May 2004

In this issue

update

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

Nicole Thomas
E-mail: nicole@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.

Extending RACF security through a simple LDAP server

INTRODUCTION

In issue 35 (February 2004) of *RACF Update* (*Easy LDAP program to authenticate RACF userids*), authentication through an IBM LDAP server was discussed. This kind of implementation requires a configuration based on only a RACF database and lets users and groups reference all the information relating to users, groups, and connections.

The next step in our exploration of the features of an LDAP server is the configuration based on both the RACF database and DB2.

This implementation lets you project a typical LDAP tree where the access list of each node may include RACF-defined users and group.

The reason why this implementation is interesting is that it lets you extend and enhance RACF's capabilities to build user profiles made up of attributes and assign them directly to groups.

What follows is the method we used to project the LDAP tree. It's a working example, it may easily be extended, and it requires little intervention by the security administrator when new users need to be enabled to Web applications.

OUR IMPLEMENTATION

The tree has two main sub-trees: the first one is the pure RACF implementation, the second one is the company Web applications structure and is implemented in DB2. Let's describe the latter sub-tree whose root we named *o=COMPANY_NAME*.

Access to the sub-tree is generically granted only to those users who previously successfully authenticated through RACF.

This characteristic is inherited by every node of this sub-tree if not explicitly overridden at the node level by access lists and new propagation attributes.

Let's see the sub-trees we generated: the first one differs from all the others.

The first sub-tree

We decided to implement a sub-tree dedicated to information common to all applications, and for this reason visible to all users (previously authenticated). In our company, for example, there is a strict correspondence between the first two characters of a userid and a previously-decided numerical code. This is typically information that is not suitable for implementation in RACF, but is used in every application or database. In the sub-tree *appl=CODES,o=COMPANY_NAME*, we made a leaf for every numerical code used, and for each one we defined an access list with only the RACF group of corresponding users.

All the other sub-trees

Every Web application has a dedicated sub-tree, whose root name is something like *appl=WEBAPPLx,o=COMPANY_NAME*.

Here is a schematic representation of our LDAP tree.

```
- root
- sysplex=SYSPLEX_NAME                          RACF subtree
+ profileType=USERS
+ profileType=GROUPS
+ profileType=CONNECTIONS
- o=COMPANY_NAME                                Webapplications sub-tree
            - appl=CODES                          numerical codes sub-tree
                  + profv = 21                              code = 21
                  + profv = 23                              code = 23
                  ……….
            + appl=WEBAPP1                        web application number 1
            + appl=WEBAPP2                        web application number 2
            ……….
```

Normally Web applications need to know which profile a user

has after log-on so that only proper masks may be shown to him. Profiles typically are coded in some kind of control files on the server on which authentication is required.

We tried to think of profiles as being composed of attributes whose values may vary in a range of possible predefined choices. This way, in a Web application sub-tree, we coded as many sub-trees as the number of attributes that make a profile. Every attribute is a profile-name and every possible value it may assume is a leaf of type *profilevalue* and whose name is the real value.

This is a schematic representation of a Web application sub-tree:

```
-      appl=WEBAPPn
- profn=URLADDRESS
+ profv=www.blabla1…..
+ profv=www.blabla2…..
……..
- profn=TYPEOFUSER
+ profv=master
+ profv=normal
…….
- profn=TYPEOFMENU
+ profv=complete
+ profv=minimal
                  + profv=administrator
                  ……
- profn=MULTIPLE_NUMERICAL_CODES
+ profv=45
+ profv=54
+ profv=99
            -…..and so on as you like
```

Every node or leaf has two main LDAP attributes. We use:

- Access list – containing a list of RACF users or groups that may search and read the node.
- aclPropagate – an LDAP attribute of the node (whose possible values are *True* or *False*) saying whether from this node downward the access list (explicitly defined or inherited from upwards) is propagated.

And now, let's have a look at LDAP code

First of all a little notation: the name of every node is built appending the name of the nodes from the current up to the root. So for example the name of the last leaf in the above LDAP tree is:

```
profv=99,profn=MULTIPLE_NUMERICAL_CODES,appl=WEBAPPn,o=COMPANY_NAME.
```

For this reason, parsing a node name lets you rebuild the information pertaining to the profile of an application.

Well, the trick is simple:

```
ldapbind –D "racfid=USERID,profileType=USER,sysplex=SYSPLEX_NAME"  -w
racf_password


ldapsearch –D "racfid=USERID,profileType=USER,sysplex=SYSPLEX_NAME" –w
racf_password  -b "o=COMPANY_NAME" (objectclass=pval)
```

The result depends on the visibility the user has of the tree: every node that has or inherits an access list in which the same user, or a group to which he belongs, is present, is returned by the **ldapsearch** call just with the entire node name.

Also when a user can search a node, he can read its attributes, which may be, for example, a URL, a mail address, or so on.

Now a few lines of code in whatever language you like are necessary for parsing and building the user profile. You may decide to give some values special meaning, such as *all_values*, and to use whatever method you like to decompose the user profiles.

TECHNICAL NOTES AND SAMPLES

And now a few notes about the implementation of this solution.

First the configuration of this LDAP tree is typically located in */etc/ldap/slapd.conf* and looks like this:

```
include      /etc/ldap/schema.system.at
include      /etc/ldap/schema.IBM.at
include      /etc/ldap/schema.user.at
```

```
include      /etc/ldap/schema.system.oc
include      /etc/ldap/schema.IBM.oc
include      /etc/ldap/schema.user.oc
port 390
maxconnections 500
timelimit 3600
sizelimit 500
adminDN "racfid=USERADM,profileType=USER,sysplex=SYSPLEX_NAME"

database sdbm GLDBSDBM
suffix  "sysplex=SYSPLEX_NAME"

database tdbm GLDBTDBM
dsnaoini /etc/ldap/dsnaoini.ini
servername YOURDB2LOCATION
databasename YOURDB2DB
dbuserid YOURDB2USERID
attroverflowsize 255
extendedgroupsearching on
suffix "o=YOUR_COMPANY"
and dsnaoini.ini is like

[COMMON]
MVSDEFAULTSSID=DBxx

[DBxx]
MVSATTACHTYPE=CAF
PLANNAME=DSNACLI

[YOURDB2LOCATION]
AUTOCOMMIT=0
CONNECTTYPE=1
```

Then, the necessary DB2 definitions to implement TDBM configuration are included in GLD.SGLDSAMP(TDBMDB) and TDBMINDX. They must be properly customized and then executed. That's all.

Then you need to update two files, schema.user.ldif and schema.IBM.ldif, after copying them from */usr/lpp/ldap/etc* to a temporary directory. Change *dn: cn=schema, <suffix>* to *dn: cn=schema,o=COMPANY_NAME*.

After starting the LDAP server, load your modified LDIF as follows:

```
Ldapmodify –v –h ip_address_ldap_server –p port –D
```

```
"racfid=LDAP_ADM,profileType=USER,sysplex=SYSPLEX_NAME" –w
ldap_adm_password –f <schema.user.ldif>

Ldapmodify –v –h ip_address_ldap_server –p port –D
"racfid=LDAP_ADM,profileType=USER,sysplex=SYSPLEX_NAME" –w
ldap_adm_password –f <schema.IBM.ldif>
```

Then you need to build and load up your LDAP tree structure.

Below is an example of the LDIF statements we used:

```
dn: o=COMPANY_NAME
objectclass: top
objectclass: organization
o: COMPANY_NAME
aclentry: group:cn=Authenticated:normal:rs

dn: appl=CODES, o=COMPANY_NAME
appl: CODES
description: numerical codes
objectclass: top
objectclass: cedacriApplid
entryowner:racfid=LDAPADM,profileType=USER,sysplex=COPLEX
ownerPropagate:TRUE

dn: pval=Ø3,appl=CODES, o=COMPANY_NAME
pval: Ø3
description: users of society Ø3
objectclass: top
objectclass: profileValue
aclEntry:group:racfid=GROUPØ3,profileType=GROUP,sysplex=COPLEX:normal:rs
aclsource:pval=Ø3,appl=CODES,o=COMPANY_NAME
aclPropagate=FALSE

dn: appl=WEBAPPL1, o=COMPANY_NAME
appl: WEBAPPL1
description: web application 1
objectclass: top
objectclass: cedacriApplid

dn: pval=cgadm,appl=WEBAPPL1, o=COMPANY_NAME
url:www.xxx.yyy/zzz
pval: cgadm
description: access for the adm users
objectclass: top
objectclass: profileValue
aclEntry:group:racfid=WAP1ADM,profileType=GROUP,sysplex=COPLEX:normal:rs
aclsource:pval=cgadm,appl=WEBAPPL1,o=COMPANY_NAME
aclPropagate=FALSE
```

How to add a new permission

If you build your RACF groups accordingly, enabling a user to use a Web application and giving him the right profile may be managed by simply connecting him to a RACF group you inserted in the access list of the leaf corresponding to the desired profile. So the RACF administrator doesn't even need to know any details about the LDAP structure. Once the LDAP administrator has built the tree using RACF groups in the access lists, he needs to act only when a new RACF group has to be explicitly inserted in an access list or when a Web application has changed its user profile structure.

Below is an example of an LDIF command to add a new RACF group to an access list (like a permit):

```
dn: pval=cgadm,appl= WEBAPPL1,o=COMPANY_NAME
changetype:modify
add:aclentry
aclEntry:group:racfid=RACFGRP1,profileType=GROUP,sysplex=SYSPLEX_NAME:normal:rs
aclEntry:group:racfid=RACFGRP2,profileType=GROUP,sysplex=SYSPLEX_NAME:normal:rs
aclEntry:group:racfid=RACFGRP3,profileType=GROUP,sysplex=SYSPLEX_NAME:normal:rs
```

and then the corresponding **LDAPmodify** to load the update:

```
ldapmodify -V 3   -h ip_address_ldap -p port ;
  -D "racfid=LDAP_ADM,profileType=USER,sysplex=SYSPLEX_NAME" -w
pass_ldap_adm ;
  -v -f /tmp/permit.ldif
```

How to write the Web application call to the LDAP server

Last but not least, there follows an example of an **ldapsearch** that the Web application needs to make of the LDAP server to obtain the leaves the user is enabled to read, so that it can build the user profile.

```
ldapsearch -D "racfid=USER_RACF,profileType=USER,sysplex=SYSPLEX_NAME" ;
-w passw   -b "pval=cgadm,appl=WEBAPPL1,o=COMPANY_NAME" ;
-h ip_address_ldap -p port "(objectclass=*)" ;
description
```

CONCLUSIONS

Our experience shows that, by just using the standard security

server components, it's possible to enhance the management of mainframe security – such as assigning to users extra profiles or groups of information normally not coded in a RACF database but necessary for Web applications, so that, after authenticating users on the mainframe, we are now able, with just a couple of calls to the LDAP server active on mainframe, to build a profile (even a very specialized one) with which to control user activity in the application. The impact of new activities on the RACF administrator is really minimal and the LDAP administrator too is less busy if he/she writes the LDAP tree efficiently.

We can confirm that it works. Try it and see!

M Elena Campidoglio
System Programmer
Cedacri Ovest SpA (Italy)

Drop-in RACF security, for your in-house utilities

You know the problem – if you have code in your shop, particularly sensitive code, you need to have some way to protect it from unauthorized use. Some pieces of code are simply too dangerous for anyone other than a select number of users to have access to. Examples of these types of program are modules that zap VTOCs directly, manipulate system control blocks, or simply examine and modify storage. Then there are other pieces of code that you support, where you may need to provide access for some individuals or groups for legitimate reasons, but which are potentially too destructive to allow open and free access for everyone. Finally you probably have some code that you just want to keep within your area because, well, maybe you just don't want to formally support it for everyone. An example of this code would be internal programs that are helpful, but if they break, or if someone finds a bug, or tries to use it for something it wasn't intended for, you will find yourself

being woken in the middle of the night to start working on a fix. Another more common reason to protect these pieces of code is a mandate from the auditors, if you let them find out that the code exists in the first place.

Even a junior systems programmer knows how to solve this basic requirement; you locate the ACEE, get the user's userid, and check it against a table that you maintain inside the code itself. In a perfect world that would be the end of it – but we don't live in a perfect world. Userids and people come and go, you will need to add userids to the table and remove old ones from it. Even if you have the forethought to mask your ID checks, as most people who get tired of maintaining these tables do, then you find that users move around, and they refuse to give up their old privileged IDs. Or worse yet, the userid standard changes entirely! This happens, not only for your code, but the code you have inherited, and the code from other potentially less experienced systems programmers in your group. So you end up doing table maintenance, and potentially a lot of it. It is simple maintenance to be sure, but the bottom line is that you have more important things to do than maintain a table of userids, a job that a first-year applications programmer could handle without any problems at all.

The formally endorsed way to handle access authorization is of course RACF. RACF calls, unfortunately, have a reputation for being difficult and cumbersome, something most systems programmers avoid whenever they can. So to solve the problems mentioned above I developed a single simple module. Just call it, and it will use the program name you are calling it from and concatenate that behind a literal, and use the resultant value to make a unique RACF call for any piece of code it is called from. You just have to check the return code, and you're done. It really becomes nothing more than three to four lines of code to permanently protect each new program, or old program that we pull an outdated userid table search out of.

Now we have no reason to keep maintaining userid lists, we let the RACF administrators handle that instead of us. Furthermore,

any of your systems programmers can handle the simple call, so no-one has an excuse for not using this security method. It's much easier to use this new module than the older way of locating the ACEE and checking it against a table. I'm sure it will become your standard for handling access to sensitive programs in your shop if you try it on one or two projects. The security module can be linked into your project, or dynamically loaded at execution time. It works the same for batch jobs as for TSO or ISPF modules.

The logic of the module is actually fairly simple. It finds the name of the module it is being called on behalf of, and appends that to a literal to develop a unique resource name. The developed resource name is then used as the object of a call to RACF to check each user's authorization. If the authorization to the generalized resource name is allowed, a return code of zero is set; and if the authorization is not allowed, a return code of 4 is set. If for some reason, eg if RACF is not active, or some other environmental error, we are not able to determine the access, a return code of 16 is set. In no case is the program cancelled, or ended, the return code is just set and the final determination for what is to be done is made by the calling program, which in our shop is usually just to end with an error message. Your code could just as easily run in a restricted mode, like viewing storage only, but not performing any updates that would have otherwise been allowed if the RACF access had been granted.

We use the literal of *TECHPGM* and append to it the name of the module being checked, to form a resource name to pass to RACF. As an example, if I were developing a new program named PROG1, the generalized resource name that the called security module would construct to be checked would be TECHPGMPROG1. We check for READ authorization, but the access type is really arbitrary and could have just as easily been update or control; we are just looking for a simple yes or no answer to the question, 'Is this user authorized to use this program?'. Since you know the name of the program you have, all you need to do is tell the security administrator to provide READ access to generalized resource name TECHPGMPROG1

or TECHPGMUTL1, or whatever name would be checked, based on your new program name, or the name of the next program you pull an old userid table out of. Now the responsibility for changing the access for your program is shifted to the security administrator, and the auditors will love you.

Below is the source for the called module, the one that actually handles the RACF call for us. It simply checks the CDE for the current module name. If the RACHEK module name is found, indicating that it was loaded, it backs up on the PRB chain of one program, and uses the program name it finds there to format the RACF call.

```
         PUNCH ' ENTRY RACHEK '
         PUNCH ' SETOPT PARM(AMODE=31,RMODE=ANY) '
* *%PDSDOC ØØ DO RACROUTE TO CHECK ACCESS TO RESTRICED ACCESS PGMS
* *-*-*----------------------------------------------------------*-*-*
* *- PROGRAM NAME - RACHEK                                        -*
* *-     FUNCTION - RACF CHECK FOR ACCESS TO RESTRICED PROGRAMS    -*
* *-     PASSED PARM = STD FORMAT PARMS - POINTER TO PGM NAME TO CHK -*
* *-     RC = Ø =  ACCESS PERMITTED                               -*
* *-     RC = 4 =  ENTITY UNKNOWN - CREATE FAC. CLASS STPGMMODNAME  -*
* *-     RC = 8  = ACCESS DENIED                                  -*
* *-                                                              -*
* *-  ** NOTE ***                                                 -*
* *-  THIS MODULE DOES NOT ALLOW, OR DENY ACCESS TO ANYTHING, IT   -*
* *-  ONLY SETS A RETURN CODE AND POSSIBLY ISSUES A MESSAGE BASED  -*
* *-  ON THE USERS "READ" ACCESS TO A FACILITY CLASS OF:           -*
* *-     T E C H P G M  +  THE MODULE NAME                        -*
* *-  IE IF THE MODULE NAME IS CONSOLE THEN THE FACILITY CLASS     -*
* *-  THAT IS CHECKED IS TECHPGMCONSOLE                           -*
* *-                                                              -*
* *-*-*----------------------------------------------------------*-*-*
RACHEK   CSECT
RACHEK   AMODE  ANY
RACHEK   RMODE  31
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
```

```
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         USING RACHEK,R15              TEMP ADDRESSABILITY
         B     STARTIT                 AROUND EYECATCHER
         DC    C'RACHEK  - TECH SERVICES AUTH. CHECK &SYSDATE'
         DS    ØH                      SET PROPER ALIGNMENT
STARTIT  BAKR  R14,RØ                  SAVE REGISTER/PSW STATUS
         LR    R12,R15
         DROP  15
         USING RACHEK,R12              R12 IS NOW BASE
         L     R2,DYNSIZE              LENGTH TO GET
         STORAGE OBTAIN,ADDR=(1),LENGTH=(2),LOC=24
         LR    R13,R1                  GET ADDRESS OF AREA
         USING DYNAREA,13              USING FOR THE DYNAMIC AREA
         MVC   4(4,R13),=C'F1SA'       FORMAT 1 SAVE AREA USED
* *-*-*-------------------------------------------------------*-*-*
* *-  PRE-FORMAT DYNAMIC AREA BASED ON STATIC MODELS            -*
* *-*-*-------------------------------------------------------*-*-*
         LA    R2,DYNMODEL             ADDRESS OF DYNAMIC AREA MODEL
         L     R3,MDLSIZE              LENGTH OF DYNAMIC AREA MODEL
         LA    R4,DYNAREA1             ADDRESS OF DYNAMIC AREA
         LR    R5,R3                   LENGTH OF DYNAMIC MODEL AREA
         ICM   R5,B'1ØØØ',X'4Ø'        SET A PAD CHARACTER OF BLANK
         MVCL  R4,R2                   COPY MODEL TO DYNAMIC AREA
* *-
         BAL   R11,GETPROG             GO GET THE CALLING PGM NAME
         BAL   R11,FORMENT             GO FORM THE ENTITY NAMES
         LA    R6,ENTITY               SET UP PTR FOR ENTITY TO CHK
         LA    R4,RACFWORK             SET UP PTR TO RACF WORK AREA
** * ------ MAKE SURE THIS GUY IS "ONE OF THE GANG" (RACHECK)   -- * **
         RACROUTE REQUEST=AUTH,WORKA=(R4),                            X
               RELEASE=2.4,                                           X
               ENTITYX=((R6)),         POINT TO OUR DEVELOPED NAME    X
               MF=(E,MRFX),                                           X
               ATTR=READ
         LR    R9,R15                  SAVE THE RETURN CODE A BIT
         LTR   R9,R9                   *IS IT OK TO ACCESS THE RESOURCE
         BZ     ACCEPT                  *- YES, ACCEPT USAGE
         B      END1                   - AND EVENTUALLY END.
ACCEPT   DS     ØH
** * ********************************************************* * **
** * **   ANY  SPECIAL PROCESSING FOR ALLOWED ACCESSES HERE   ** * **
** * **   - for example you may want to write an SMF record   ** * **
** * ********************************************************* * **
         B     RETURN
END1     EQU   *
** * ********************************************************* * **
```

```
** * **  ANY SPECIAL PROCESSING FOR REJECTED ACCESSES HERE    ** * **
** * ********************************************************** * **
** * **     IF AUTH. IS NOT OBTAINED ISSUE MESSAGE - BUT THE   - * **
** * **          CALLING PGM IS RESPONSIBLE FOR DENYING ACCESS - * **
** * ************************************************************ **
         LA    R8,ERRMSG2
         SR    RØ,RØ                     ALWAYS CLEAR RØ BEFORE A WTO
         WTO   TEXT=(R8),MF=(E,WTOLIST)  ISSUE AN ERROR MESSAGE
RETURN   L     R2,DYNSIZE                LENGTH TO RELEASE
         LR    R8,R13                    ADDR OF STORAGE TO RELEASE
         STORAGE RELEASE,ADDR=(8),                                    X
               LENGTH=(2)                RELEASE AQUIRED STORAGE
         LR    R15,R9                    RESET RETURN CODE IN R15
         PR                              RESTORE STATUS & RETURN
** * ********************************************************** * **
** * **  GET THE CALLING PROGRAM NAME  -                       ** * **
** * **                                                        ** * **
** * **    FIRST GET THE CURRENT RB'S CDE                      ** * **
** * **                                                        ** * **
** * **       - IF IT IS RACHEK ( THEN WE WERE LOADED )        ** * **
** * **           WALK BACK THE RB CHAIN LOOKING FOR A PRIOR   ** * **
** * **           PRB AND THEN USE THE CDE FOUND THERE TO      ** * **
** * **           RESOLVE THE PGM NAME.                        ** * **
** * **                                                        ** * **
** * **       - IF NOT RACHEK THEN WE WERE STATICALLY LINKED   ** * **
** * **           AND WE SHOULD JUST USE THAT NAME             ** * **
** * **                                                        ** * **
** * ********************************************************** * **
GETPROG  EQU   *                       GET THE PROGRAM NAME TO CHECK
         USING PSA,RØ                  MAP THE PSA
         L     R6,PSATOLD              R6 HAS THE TCB ADDRESS
         USING TCB,R6                      MAP IT
         L     R8,TCBRBP               R8 HAS THE RB POINTER
         USING RBSECT,R8                   MAP IT
CHKMOD   ICM   R9,B'Ø111',RBCDE1       GET CDE ADDRESS FOR THIS RB
         USING CDENTRY,R9
* - FIRST IS THIS THE RACHEK MODULE    (WE MAY HAVE BEEN LOADED)
         CLC   CDNAME(7),=C'RACHEK '   GET A COPY OF THE MODNAME
         BE    BACKUP1                 IF IT IS US, THEN BACKUP SOME
         MVC   MODNAME,CDNAME          KEEP THE PGM NAME TO CHECK
         B     Ø(R11)                  RETURN - WE ARE DONE HERE
BACKUP1  EQU   *                       BACK UP TO PREVIOUS PRB
* FIRST CHECK TO SEE IF WE CAN BACK UP - OR - IF AT TOP ALREADY
         TM    RBSTAB2,RBTCBNXT        IS THIS THE TOP RB?
         BO    MAJERR1                 IF SO THEN ERROR HAS OCCURED
         ICM   R8,B'Ø111',RBLINKB      GET PREVIOUS RB IN CHAIN
         TM    RBSTAB1,RBFTP           IS THIS ONE A PRB?
         BE    CHKMOD                    IF SO THEN CHECK CDE
         B     BACKUP1                   IF NOT LOOK BACK FURTHER
MAJERR1  EQU   *                       - NO NON RACHEK PRB IS FOUND
```

```
         LA    R9,ERRMSG1
         SR    RØ,RØ                    ALWAYS CLEAR RØ BEFORE A WTO
         WTO   TEXT=(R9),MF=(E,WTOLIST)
         L     R9,=F'16'                 SET RETURN CODE IN R9
         B     RETURN                     RELEASE STORAGE AND GO HOME
** * ********************************************************* * **
** * ** THIS ROUTINE WILL FORMAT ALL REQUIRED AREAS FOR THE   ** * **
** * ** RACROUTE CALL WE ARE ABOUT TO MAKE                    ** * **
** * ********************************************************* * **
FORMENT  EQU   *
         MVI   ENTITY,X'4Ø'           ENSURE WE START WITH ALL BLANKS
         MVC   ENTITY+1(L'ENTITY-1),ENTITY
         MVC   ENTITY(4),=X'ØØØØØØØØ' ZERO THE PREFIX AREA FOR ENTITY
         MVC   ENTITY+4(7),=C'TECHPGM'  GENERAL GROUPING
         MVC   ENTITY+11(8),MODNAME   DETAIL RESOURCE LEVEL
         LA    R9,9(Ø,RØ)             MAX LENGTH TO LOOK 4 BLANK
         LA    R8,ENTITY+11            WHERE TO START LOOKING
LOOKMORE CLI   Ø(R8),X'4Ø'            LOOK FOR DELIMITING BLANK
         BE    FNDEND                 BRANCH IF FOUND
         LA    R8,1(R8)                - ELSE BUMP OUR POINTER
         BCT   R9,LOOKMORE            CHECK LIMIT AND LOOP
FNDEND   EQU   *
         LA    R6,ENTITY+4
         SR    R8,R6                  LENGTH OF NAME IS NOW IN R8
         STH   R8,ENTITY+2            SAVE LENGTH IN PREFIX
         STH   R8,ENTITY                DO IT TWICE
         B     Ø(R11)                 RETURN FROM SUB ROUTINE
* *-*-*----------------------------------------------------------*-*-*
* *- STATIC STORAGE AREA HERE - LTORG - MODELS ETC.                -*
* *-*-*----------------------------------------------------------*-*-*
DYNMODEL DS    ØF
MWTOPL   WTO   TEXT=,            WTO PARAMETER LIST                  X
               MF=L
MRFX     RACROUTE REQUEST=AUTH,             Authorization check      X
               RELEASE=2.4,                                         X
               LOG=ASIS,                                            X
               CLASS='FACILITY',            Facility class to check X
               ATTR=READ,                    Check for read access  X
               MF=L
         DS    ØF
@MDLSIZE EQU   *-DYNMODEL             LENGTH OF THE COPIED AREA
DYNSIZE  DC    AL4(@DYNSIZE)          DYNAM AREA SIZE
MDLSIZE  DC    AL4(@MDLSIZE)          MODELED AREA SIZE
ERRMSG1  DC    AL2(L'ERRØ1)           LENGTH OF WTO MESSAGE
ERRØ1    DC    C'RACHEKØØ IS NOT INDEPENDENTLY EXECUTABLE - PROGRAM ENDX
               ING '
ERRMSG2  DC    AL2(L'ERRØ2)           LENGTH OF WTO MESSAGE
ERRØ2    DC    C'RACHEKØ1 - FACILITY CLASS CALL HAS BEEN DENIED - GET AX
               PPROPRIATE RACF AUTH. TO USE THIS PGM'
         LTORG
```

```
*
DYNAREA  DSECT
SAVEAREA DS    18F
         DS    ØF                ALIGN FOR PERFORMANCE AND RACF USAGE
ENTITY   DS    CL2Ø              ENTITY NAME BUFFER
USERID   DS    CL8
MODNAME  DS    CL8
DYNAREA1 DS    ØF
WTOLIST  WTO   TEXT=,            WTO PARAMETER LIST                  X
               MF=L
RFX      RACROUTE REQUEST=AUTH,       Authorization check           X
               RELEASE=2.4,                                         X
               LOG=ASIS,                                            X
               CLASS='FACILITY',     Check for facility class       X
               ATTR=READ,              check read authorization     X
               MF=L
RACFWORK DS    CL512
@ENDDYN  DS    ØX                    USED TO CALC DYNAM AREA SIZE
@DYNSIZE EQU   ((@ENDDYN-DYNAREA+7)/8)*8  DYNAM AREA ROUNDED TO DBLWD
         IKJTCB
         IHARB
         IHACDE
         IHAPSA
         END   RACHEK
```

Here is an example of calling the module from within a batch program, when the module is statically linked with the calling module:

```
         PUNCH ' INCLUDE MYLOAD(RACHEK) '
         PUNCH ' ENTRY CONSOLE '
         PUNCH ' SETOPT PARM(AMODE=24,RMODE=24) '
* *%PDSDOC ØØ TEST PROGRAM TO USE WITH RACHEK
* *-*-*-------------------------------------------------------*-*-*
* *- PROGRAM NAME - CONSOLE                                     -*
* *-     FUNCTION - CHECK THE USE OF THE STRACHEK PROGRAM       -*
* *-*-*-------------------------------------------------------*-*-*
CONSOLE CSECT
CONSOLE AMODE   ANY
CONSOLE RMODE   24
RØ       EQU    Ø
R1       EQU    1
R2       EQU    2
R3       EQU    3
R4       EQU    4
R5       EQU    5
R6       EQU    6
R7       EQU    7
R8       EQU    8
R9       EQU    9
```

```
R1Ø      EQU    1Ø
R11      EQU    11
R12      EQU    12
R13      EQU    13
R14      EQU    14
R15      EQU    15
         USING CONSOLE,15              TEMP ADDRESSABILITY
         B     @PROLOG                 AROUND EYECATCHER
         DC    C'CONSOLE  - TECH SERVICES AUTH. CHECK &SYSDATE'
         DS    ØH                      PRESERVE ALIGNMENT
@PROLOG  BAKR  R14,RØ                  SAVE REGISTER/PSW STATUS
         LR    R5,R1                   SAVE PARM POINTER
         LR    12,15
         DROP  15
         USING CONSOLE,R12             R12 IS NOW BASE
         L     R2,DYNSIZE              * LENGTH TO GET
         STORAGE OBTAIN,ADDR=(1),SP=Ø,LENGTH=(2)
         LR    R13,R1                  GET ADDRESS OF AREA
         USING DYNAREA,R13             USING FOR THE DYNAMIC AREA
         MVC   4(4,R13),=C'F1SA'       FORMAT 1 SAVE AREA USED
*****************************************************************
** HERE IS THE CALL TO RACHECK FOR AUTHORIZATION CHECK         **
** JUST CALL THE MODULE AND CHECK THE RETURN CODE              **
** R15=Ø = AUTHORIZATION IS OK ** R15>Ø = DON'T ALLOW ACCESS   **
*****************************************************************
         SR    R9,R9                   PRE-CLEAR RETURN CODE
         L     R15,=V(RACHEK)
         BALR  14,15
         LTR   15,15
         BZ    DOFUNCTN
         SR    RØ,RØ                   ALWAYS CLEAR RØ BEOFRE WTO
         WTO   'CONSOLEØ1 -ACCESS NOT PERMITTED- FUNCTION ENDING'
         LA    R9,16(Ø,RØ)             SET RETURN CODE OF 16
ENDIT    EQU   *
RETURN   L     R2,DYNSIZE              LENGTH TO GET
         LR    R8,R13
         STORAGE RELEASE,ADDR=(8),LENGTH=(2)
         LR    R15,R9                  RESTORE RETURN CODE IN R15
         PR                            RESTORE STATUS & RETURN
DOFUNCTN EQU   *
*  PERFORM THE SENSITIVE WORK KNOWING IT HAS BEEN AUTHORIZED
         XR    R9,R9                   CLEAR THE RETURN CODE
*        . . .
*        . . .
*        . . .
         B     ENDIT
DYNSIZE  DC    AL4(@DYNSIZE)
         LTORG
*
DYNAREA  DSECT
```

```
SAVEAREA DS    18F
         DS    ØF
*  OTHER VARIABLES GO IN HERE
@ENDDYN  DS    ØX                      USED TO CALC DYNAM AREA SIZE
@DYNSIZE EQU   ((@ENDDYN-DYNAREA+7)/8)*8  DYNAM AREA SIZE ON DBLWD BDY
         END   CONSOLE
```

Alternatively, the module can be dynamically loaded at execution time rather than being statically linked. This obviously has the advantage of allowing maintenance to be performed on the module, if it is ever needed, without having to relink every module that calls it. By loading the module at execution time, you also remove the literals used to form the resource name – which some may think has certain security advantages. Of course, when you load the module the calling sequence should be a bit different to account for potentially different addressing modes.

An example of the load and call would be:

```
SR    15,15                   Pre-clear the return code
LOAD  EP=RACHEK               LOAD OR GET ENTRY ADDRESS
LR    15,Ø                    ENTRY POINT ADDRESS IN R15
BASSM 14,15                   MAKE THE CALL TO RACHEK
LTR   15,15                   FINALLY CHECK THE RETURN CODE
```

Now you have everything you need to easily handle legitimate RACF controlled program access for all of your in-house code. Anyone in your group that does any coding at all should be able to handle it, the auditors should be quite satisfied, and best of all you make the security administrators handle the administration of your userid tables.

I hope this saves you some time in the future, while you satisfy the auditor's requirements, and generally provide better products to your shop.

Stephen McColley
Senior Systems Programmer
SunTrust Bank (USA)

Obtaining RACF information the easy way

A few years ago, about ten I guess, when I was beginning to understand a little bit about the way RACF worked, I started feeling the need to obtain information in a way that the available RACF commands were not able to give me.

At that time, if memory serves me right, I needed to check the access list of several profiles in the FACILITY class. I could, and did, obtain the relevant profiles, by means of a **SEARCH CLASS (FACILITY) MASK (something)** or **FILTER (something)**, and then I would proceed with the relevant **RL FACILITY (something) AUTH**, and check the access lists that RACF provided me with.

That proved to be a bit cumbersome and time consuming. And then, the access list would be shown in the order that it had been established, not in an orderly way. When what I wanted was to compare access lists, this would mean added work and complication.

In order to overcome this hurdle, I wrote the first version of the RL@CLAS command, which I have been improving since then. The first version gave me the basic information I was looking for at the time: profile name, universal access, and access list. Over time, I made a few changes, and included, in the output list, the owner, creation date, and installation data.

I sorted the access list, by access (ALTER down to NONE), and alphabetically, for each access type. And I displayed the entries in the access list in an evenly-spaced way, which makes it easier when you are comparing access lists of different profiles. It makes for easier reading.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
YYYY-MM-DD ==> Profile-name
               --> Entry #one
               --> Entry #two
               --> . . .
               --> Entry #nnn
     OWNER ==> Group-Owner
```

```
       (N) ==> Whatever Installation Data it may have
   ALTER   ==> OtherGrp ThisGrp
   UPDATE  ==> AntGroup
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Then, when the STARTED class made its appearance, I included the STDATA:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
YYY-MM-DD ==> STCNAME.*
     OWNER ==> GroupName
       (N) ==> This STC Installation
           ==> No Users
    STDATA ==> USR(USERID)    GRP(GROUP)    TRUST(NO)  PRIV(NO)
TRACE(NO)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

At some point in time, I included some code so that, when dealing with the PROGRAM class, I could validate whether the datasets were on the volumes referred by the profile (for this one, it is better if you do not have the TSO MOUNT authorization; you might get stuck with a MOUNT pending for some forgotten volume definition). And you must be aware that even if the volumes do not exist in the system where you are working, they can exist in another system that shares the RACF you are using:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
YYYY-MM-DD ==> *
      --> SYS1.C.SCLBDLL............................/SYSRS2/NO
     ---> SYS1.C.SCLBDLL............................/SYSRS1/NO
      --> SYS1.C.SCLBDLL............................/SYSRS3/NO
      --> SYS1.C.SCLBDLL............................/SYSRS2/NO
      --> SYS1.LE.SCEERUN.........................../SYSRS2/NO
      --> SYS1.LE.SCEERUN.........................../SYSRS1/NO
      --> SYS1.LE.SCEERUN.........................../ NONE /NO
      --> SYS1.LE.SCEERUN.........................../SYSRS2/NO
      --> SYS1.LE.SCEERUN.........................../SYSRS4/NO /NOT OK
      --> SYS1.LE.SCEERUN.........................../ NONE /NO /NOT OK
      --> SYS1.LINKLIB............................../******/YES
     OWNER ==> Owner-Group
       (R) ==> No Installation Data
   NONE    ==> GrPØ1   Grp1     ZGrp2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The YES or NO has to do with the PADCHK option. The NOT OK means that the dataset was not found on that volume, or that the

volume itself was not found. You will have to work out which one. If there is no volume associated with the library definition, then NONE will be displayed. I am using the TSO ALLOCATE command to check for library existence. It is a bit heavy on resource consumption, and it delays the overall execution, but it is not too much of a problem if you send it through batch, or if you are not checking too many profiles.

There are two small caveats: you do not get to see your access level, and it does not handle conditional access list. I never needed that, so I never wrote the code for it. And I wanted condensed information! If I wanted everything, I would use (and I use it) the basic RL command. However, I have been using it a lot, on a day-to-day basis, ever since I wrote it. It has proved itself quite valuable.

This program can generate from a few to a lot of lines. The way it works is, it will check if it is running in batch or in ISPF foreground. If the former, it will display its results by means of the REXX **say** command. If the latter, it will create a temporary dataset, then it will write the results into it, and it will invoke the ISPF **BROWSE** command to display it.

This file will be deleted on exit. Usually, if I need the output for something more than a quick look, I will send it through batch execution and use SDSF to save it in a file.

The way to use it is simple. You can invoke it with just one parameter – that being the class that you want to check. If you do not want to check all the profiles in the class, you must enter a filter. This can be done in one of two ways. RL@CLAS uses the RACF **SEARCH** command, and it will use the second parameter to determine the way it will be invoked. If you do not pass it anything, it will do a **SEARCH NOMASK**. If you pass it the first positions of the profiles that you want to check, it will use a **SEARCH MASK(second parameter)**. If the common part of the profile names is not in the first positions, you can use, as the second parameter, **FILTER(whatever)**. This will generate a **SEARCH FILTER(whatever)** command, which means that all you have to do is to specify a valid RACF

SEARCH mask. The parsing is done by masking, so it is imperative that, if using the FILTER keyword, you do not leave any space between 'FILTER' and the '('. Alternatively, you could change the code to alter this. So, you can invoke it thus:

```
RL@CLAS FACILITY $DASDI.P
```

This will list all the profiles in the FACILITY class, beginning with $DASDI.P.

If you invoke it like this:

```
RL@CLAS FACILITY FILTER(**.*DDR*.**)
```

it will list all the profiles in the FACILITY class that have the string DDR in any place in their name.

More recently, I included the ability to exclude some profiles from the SEARCH result, prior to doing the actual RLISTs. If you want to use this feature, you will have to specify **EXCLUDE(exclude_mask)** as the third (or second) parameter. This exclude mask is not a RACF SEARCH mask. RL@CLAS will check this mask against all the profiles returned by the RACF SEARCH, and it will exclude the matches. At the present time I am using a WORDPOS to do the match, but you can use a simple POS instead. As in the use of the FILTER keyword, the use of EXCLUDE must be without spaces between 'EXCLUDE' and '('. This is a very recent addition, so it may need improvement.

If you invoke it this way:

```
RL@CLAS FACILITY FILTER(**.*DDR*.**) EXCLUDE(SYSV.)
```

you will get the same list as in the previous example, minus the profiles beginning with SYSV.

CODE

```
/* rexx
                        *************************
                        *                       *
                        *  Joao Bentes de Jesus  *
                        *     RL@CLASS 2.3.Ø     *
```

```
                        *                          *
                        ***************************
-                                                                        */
arg class_name mask excl
if  class_name="" then
    do
        say"You did not specify the class to list"
    end
else
    do
        call get_info
    end
return
/* - - - - - - - - - - - - - */
get_info:
ok=1
if  mask="" then
    do
        sr_out.Ø=1
        sr_out.1="*"
    end
else
    do
        if  pos("FILTER",mask)=1 then
            nop
        else
            do
                mask="mask ("mask")"
            end
        x=outtrap(sr_out.,,"noconcat")
        "sr "mask" class ("class_name")"
        x=outtrap("OFF")
        if  rc=Ø then
            do
                if  excl¬="" then
                    do
                        call clean_sr_out
                    end
            end
        else
            do
                ok=Ø
                say sr_out.1
            end
    end
if  ok then
    do
        call parse_results
    end
```

```
return
/* - - - - - - - - - - - - - - */
parse_results:
z=Ø
sep=copies("- ",38)"-"
opts="HIST AUTH"
if  abbrev("STARTED",class_name) then
    do
        opts=opts" STDATA"
    end
else
    do
        if  abbrev("PROGRAM",class_name) then
            do
                cvt_start = c2x(storage(1Ø,4))
                ucb_addr  = c2x(substr(storage(cvt_start,52),49,4))
                sysres    = substr(storage(ucb_addr,5Ø),29,6)
            end
    end
do  a=1 to sr_out.Ø
    x=outtrap(i_line.,,"noconcat")
    "rl "class_name" ("word(sr_out.a,1)")" opts
    x=outtrap("OFF")
    do  l=1 to i_line.Ø
        select
            when space(i_line.l)="CLASS NAME" then
                do
                    call make_sep
                    l=l+2
                    z=z+1
                    o_line.z=word(i_line.l,2)
                    base=z
                end
            when subword(space(i_line.l),3,2)="UNIVERSAL ACCESS" then
                do
                    l=l+2
                    z=z+1
                    o_line.z=right("OWNER",1Ø)" ==> "word(i_line.l,2)
                    warning=word(i_line.l,words(i_line.l))
                    if  warning="YES" then
                        do
                            o_line.z=overlay(o_line.z,,
                                right("***** WARNING MODE *****",79))
                        end
                    z=z+1
                    o_line.z="       ("left(word(i_line.l,3),1)") ==>"
                end
            when subword(space(i_line.l),1,2)="CREATION DATE" then
                do
```

```
            l=l+3
            parse value i_line.l with dias ano .
            cr_date=date("S",right(ano,2,Ø),
                ||right(dias,3,Ø),"J")
            cr_date=insert("-",insert("-",cr_date,6,1),4,1)
            o_line.base=cr_date" ==> "o_line.base
        end
    when subword(space(i_line.l),1,2)="STDATA INFORMATION" then
        do
            z=z+1
            o_line.z="    STDATA ==>"
            do  l=l+2 to l+6
                parse value i_line.l with um"= "dois .
                select
                    when um="USER" then
                        opt="USR"left("("dois")",1Ø)
                    when um="GROUP" then
                        opt="GRP"left("("dois")",1Ø)
                    when um="TRUSTED" then
                        opt="TRUST"left("("dois")",5)
                    when um="PRIVILEGED" then
                        opt="PRIV"left("("dois")",5)
                    otherwise
                        opt=um||"("dois")"
                end
                o_line.z=o_line.z opt
            end
        end
    when space(subword(i_line.l,1,3))="DATA SET NAME" &,
            abbrev("PROGRAM",class_name) then
        do  l=l+2
            if  i_line.l="" then
                do
                    leave l
                end
            z=z+1
            parse value i_line.l with dsn vol pads .
            o_line.z=copies(" ",1Ø)"-->",
                left(dsn,44,".")
            if  pads="" then
                do
                    pads=vol
                    vol=" NONE "
                end
            o_line.z=o_line.z,
                ||"/"left(vol,6),
                ||"/"left(pads,3)
            call check_pds
        end
```

```
            when space(i_line.l)="INSTALLATION DATA" then
                do
                    l=l+2
                    lx=length(o_line.z)
                    if  i_line.l="NONE" then
                        do
                            o_line.z=o_line.z "No Installation Data"
                        end
                    else
                        do
                            data=i_line.l
                            do  l=l+1
                                if  i_line.l="" then
                                    do
                                        leave l
                                    end
                                else
                                    do
                                        data=data||i_line.l
                                    end
                            end
                            data=space(data)
                            call format_data data
                        end
                end
            when subword(space(i_line.l),1,2)="USER ACCESS" then
                do
                    call get_acc_level
                end
            when space(i_line.l)="RESOURCES IN GROUP" then
                do  l=l+2
                    if  i_line.l="" | i_line.l="NONE" then
                        do
                            leave l
                        end
                    z=z+1
                    o_line.z=copies(" ",15)"--> "strip(i_line.l)
                end
            otherwise
                nop
        end
    end
end
call make_sep
if  z>0 then
    do
        if  sysvar("SYSENV")="FORE" & sysvar("SYSISPF")="ACTIVE" then
            do
                call browse_output
```

```
            end
        else
            do  a=1 to z
                say o_line.a
            end
    end
return
/* - - - - - - - - - - - - */
clean_sr_out:
parse value excl with excl"("excl_mask")"
if  abbrev("EXCLUDE",excl) then
    do
        x=outtrap(ex_out.,,"NOCONCAT")
        "SR MASK("excl_mask") CLASS("class_name")"
        x=outtrap("OFF")
        if  rc=Ø then
            do
                tab_out=""
                do  a=1 to sr_out.Ø
                    tab_out=tab_out word(sr_out.a,1)
                end
                do  a=1 to ex_out.Ø
                    px=wordpos(word(ex_out.a,1),tab_out)
                    if  px>Ø then
                        do
                            tab_out=delword(tab_out,px,1)
                        end
                end
                drop sr_out.
                sr_out.Ø=words(tab_out)
                do  a=1 to sr_out.Ø
                    sr_out.a=word(tab_out,a)
                end
                drop tab_out
            end
    end
return
/* - - - - - - - - - - - - */
check_pds:
dd="I"time("S")
x=outtrap("ON")
if  vol=" NONE " then
    do
        "alloc f("dd") shr da('"strip(dsn)"')"
    end
else
    do
        if  vol="******" then
            do
```

```
                chk_vol=sysres
            end
        else
            do
                chk_vol=vol
            end
        "alloc f("dd") shr da('"strip(dsn)"') VOL("chk_vol")"
    end
if  rc=Ø then
    do
        o_line.z=o_line.z
        "free f("dd")"
    end
else
    do
        o_line.z=o_line.z"/NOT OK"
    end
x=outtrap("OFF")
return
/* - - - - - - - - - - - - */
get_acc_level:
acc_lvls="ALTER CONTROL UPDATE READ EXECUTE NONE"
do  acl=1 to words(acc_lvls)
    ac_lvl=word(acc_lvls,acl)
    interpret "ac_"ac_lvl"='"
end
do  l=l+2
    if  i_line.l="" | l =i_line.Ø then
        do
            users=words(acc_lvls)
            do  acl=1 to words(acc_lvls)
                ac_lvl=word(acc_lvls,acl)
                grupos=value("ac_"ac_lvl)
                if  grupos¬="" then
                    do
                        grupos=sort_tab(grupos)
                        z=z+1
                        o_line.z="   "left(ac_lvl,8)"==>",
                            put_8(subword(grupos,1,7))
                        do  ws=8 by 7 to words(grupos)
                            z=z+1
                            o_line.z=left("",14),
                                put_8(subword(grupos,ws,7))
                        end
                    end
                else
                    do
                        users=users-1
                    end
```

```
            end
            if  users=Ø then
                do
                    z=z+1
                    o_line.z=right("==> No Users",23)
                end
            leave l
        end
    parse value i_line.l with ac_group ac_lvl .
    interpret "ac_"ac_lvl"=ac_"ac_lvl" ac_group"
end
return
/* - - - - - - - - - - - - */
put_8:
parse arg in_data
out_data=""
do  p8=1 to words(in_data)
    out_data=out_data left(word(in_data,p8),8)
end
return strip(out_data)
/* - - - - - - - - - - - - */
sort_tab:
arg tab_input
qx=words(tab_input)
do  wk=1 to qx
    var.wk=word(tab_input,wk)
end
do  w1=1 to qx-1
    w2=w1+1
    if  var.w1>var.w2 then
        do
            var_xx=var.w1
            var.w1=var.w2
            var.w2=var_xx
            do  w4=w1 by -1 to 2
                w3=w4-1
                if  var.w3>var.w4 then
                    do
                        var_xx=var.w3
                        var.w3=var.w4
                        var.w4=var_xx
                    end
                else
                    do
                        leave w4
                    end
            end
        end
end
```

```
tab_output=""
do  wk=1 to qx
    tab_output=tab_output var.wk
end
return tab_output
/* - - - - - - - - - - - - */
format_data:
arg i_line
zx=length(i_line)
if  zx>64 then
    do
        px=lastpos(" ",left(i_line,64))
        if  px=Ø then
            do
                px=65
            end
        o_line.z=o_line.z left(i_line,px-1)
        z=z+1
        o_line.z=left("",lx)
        data=substr(i_line,px+1)
        call format_data data
    end
else
    do
        o_line.z=o_line.z i_line
    end
return
/* - - - - - - - - - - - - */
make_sep:
if  o_line.z¬=sep then
    do
        z=z+1
        o_line.z=sep
    end
return
/* - - - - - - - - - - - - */
browse_output:
dsn ="'"userid(),
   ||"."mvsvar("SYSNAME"),
   ||".D"date("J"),
   ||".T"space(translate(time(),,":"),Ø),
   ||".TEMP'"
dd="O"time("S")
"alloc f("dd") da("dsn") recfm(v b) lrecl(84) space(1Ø 5)",
    "tracks new"
if  rc=Ø then
    do
        "execio "z" diskw "dd" (finis stem o_line.)"
        address "ISPEXEC" "control errors return"
        address "ISPEXEC" "browse dataset("dsn")"
```

```
        "free f("dd") delete"
    end
else
    do
        say"Error ("rc") on "dsn" ALLOC"
    end
return
/* - - - - - - - - - - - - */
```

Joao Bentes de Jesus
Systems Programmer (Portugal)

RACF 101 – RACF commands

This is a new column that will address basic topics in RACF. It is meant to help newbies to the RACF world better understand the intricacies of RACF. This time, we will look at RACF command structure.

New RACF administrators often have to decide whether or not to learn RACF command syntax. Quite often, installations have RACF products from vendors that simplify the administration work, and often reduce the need to look up command syntax, if not eliminate the need entirely.

Even if you have vendor products (or their home-grown equivalents) to simplify RACF administration, in the long run nothing beats knowing the basics of RACF commands and how they work. If you can find some structure and logic in how RACF commands are organized, so much the better – it will make it easier to remember them.

In this discussion we will deal with only the most common RACF commands.

To understand the basic RACF commands, think of the entire RACF database consisting of nothing but profiles. This is indeed the case, with very few exceptions. And then, think of RACF commands doing nothing more than manipulating these profiles.

What could you possibly want to do with profiles? Well, only four things – list, add, delete, or alter them! Now, there are only four types of RACF profiles – dataset, general resource, user, and group.

So we need to figure out how to perform our four basic operations on these four types of profiles. Perhaps we will even see some rhyme and reason here!

LISTING RACF PROFILES

The commands for listing RACF profiles are:

- LISTDSD (dataset profiles)
- RLIST (resource profiles)
- LISTUSER (user profiles)
- LISTGRP (group profiles).

You can see that the commands look similar, except maybe RLIST.

ADDING RACF PROFILES

The commands for adding RACF profiles are:

- ADDSD (dataset profiles)
- RDEFINE (resource profiles)
- ADDUSER (user profiles)
- ADDGROUP (group profiles).

Again, you see that they have similar forms except maybe RDEFDINE.

ALTERING (CHANGING) RACF PROFILES

The commands for altering (changing) RACF profiles are:

- ALTDSD (dataset profiles)

- RALTER (resource profiles)
- ALTUSER (user profiles)
- ALTGROUP (group profiles).

RALTER looks a little different from the rest.

DELETING RACF PROFILES

The commands for deleting RACF profiles are:

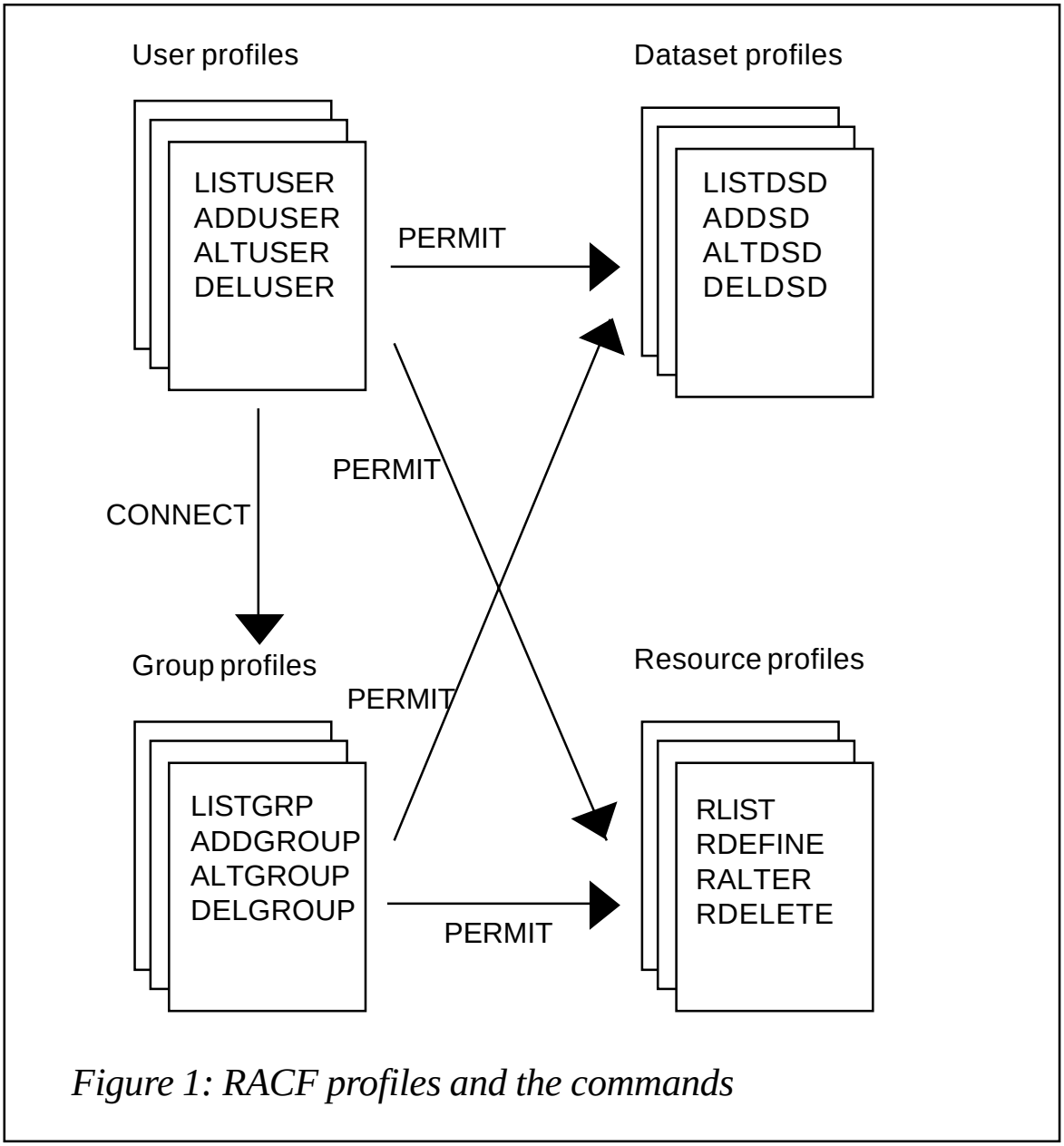

Figure 1: RACF profiles and the commands

- DELDSD (dataset profiles)
- RDELETE (resource profiles)
- DELUSER (user profiles)
- DELGROUP (group profiles).

The command RDELETE is a little different.

So, we see that the commands look similar, except when it comes to commands for resource profiles. They are a bit of an oddity, but easy to remember after a while.

Figure 1 shows the RACF profiles and the commands to manipulate them. It also introduces two other basic RACF commands – CONNECT and PERMIT.

The CONNECT command is used to connect users to groups. In RACF, access can be granted to a user or to a group. Granting access at the group level simplifies RACF administration, but first we would need to use the CONNECT command to put a user in a group.

Lastly, we need to permit users and groups to profiles. For that, there is the PERMIT command.

SUMMARY

We have looked at only the basic RACF commands, and even then, only the format of the commands. The idea was to see how they are structured and named. If you view the various types of profiles as 'decks of cards', as in Figure 1, the RACF structure makes more sense.

Dinesh Dattani (dinesh123@rogers.com)
Independent Consultant
Toronto (Canada)

Coding RACF exits using IBM C/C++

Using documented exit points in MVS and its related subsystems has been a popular practice for many years. The exit points are provided to allow different users of the operating system or subsystem software to effect alternative results based on their own specific needs rather than using the default system behaviour. In almost every case, the interface to these exit points is documented in its Assembler terms and the corresponding input parameter data mappings are frequently provided in Assembler DSECT format.

This holds true for RACF as well. Documentation for the RACF exits discusses register contents on entry and register return value settings and their corresponding effects on the exit primarily as they relate to use in an Assembler program.

No-one can argue the operational efficiency of an Assembler program when compared with a high-level language program that performs the same function, but occasionally it is more interesting, more fun, or maybe even more practical to use a high-level language in a situation more normally associated with using an Assembler program.

This article discusses an Assembler stub program that acts as the framework for setting up the environment to call a C coded subroutine that supplies exit logic. The stub program can be used (with one very minor modification) for each different C 'exit' code program you would like to create. To demonstrate this, the Assembler stub program and two sample C code RACF exit subroutines – an IRREVX01 exit and an ICHPWX01 exit – have been included.

THE STUB PROGRAM – XITFRMWK

The stub program, XITFRMWK, provides the set up call to the C subroutine and the return logic. It passes the value of R1, the most common exit parameter list pointer, through to the C

subroutine unaltered, allowing the C subroutine to use the parameter list pointer as the input parameter for addressing the parameter list as a C structure. The return code from the C subroutine is passed back as the return value in R15 from the XITFRMWK stub program. The stub program can be used to drive any C subroutine by changing the CALL statement to reference the specific program to be invoked.

THE IRREVXC1 SUBROUTINE

The IRREVXC1 subroutine provided in this article is for the IRREVX01 RACF exit point. This particular exit examines the specific RACF command information to determine whether the command in question is a RACF LISTUSER command. If the command is a LISTUSER command, this subroutine will force on the display of the TSO and OMVS segment information for the userid regardless of what has been specified in the original command. The EVXPL structure definition has been created from the IRREVXP DSECT mapping macro by using the CBC3DSCT program to convert the Assembler DSECT into a C structure definition.

The C subroutine also sets **#pragma runopts(TRAP(OFF))**. This allows the C code to run in supervisor state.

The **ENTREGS()** function referenced in the IRREVXC1 subroutine is provided in the XITFRMWK stub program. The purpose of the **ENTREGS()** function is to return the values of R0 – R15 as they existed on entry to the XITFRMWK stub program. A 16-entry register value vector is used for this purpose. Check the program for information on how to activate the register display console messages.

The IBM C/C++ supplied **__console()** function is used to display exit messages to the operator console. This function provides functions similar to the WTO macro for Assembler programs.

To create a usable IRREVX01 exit, perform the following:

- Assemble the XITFRMWK stub program. Be sure to specify IRREVXC1 on the call macro as follows:

```
CALL IRREVXC1
```

- Compile the IRREVXC1 subroutine. Before running the compile, be sure to convert all occurrences of '[' to X'AD' and ']' to X'BD'. Be sure to run the C compile with the RENT option.
- To accommodate subroutine names greater than eight characters and the RENT compile option, the compile step object code (created in the previous step) must be run through the pre-linker. If the object module created in the previous step is called IRREVXCC, use the following sample JCL to create the IRREVXC1 object module:

```
//PLKED1  EXEC PGM=EDCPRLK,PARM='UPCASE',
//    REGION=2Ø48K
//SYSMSGS  DD  DSNAME=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  DSN=TCPIP.SEZARNT1,DISP=SHR
//SYSOBJ   DD  DSN=object.code.pds,DISP=SHR
//SYSMOD   DD  DSN=object.code.pds(IRREVXC1),DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  INCLUDE SYSOBJ(IRREVXCC)
```

- Create the IRREVX01 load module using the following linkedit JCL:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT   DD    DSN=object.code.pds,DISP=SHR
//SYSLMOD  DD    DSN=load.library,DISP=SHR
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN   DD    *
   INCLUDE OBJECT(XITFRMWK)
   INCLUDE OBJECT(IRREVXC1)
   ENTRY   XITFRMWK
   SETCODE AC(1)
   NAME    IRREVXØ1(R)
```

The exit can be activated dynamically using the following OS/390 operator command:

```
SETPROG EXIT,ADD,EXITNAME=IRREVXØ1,MODNAME=IRREVXØ1,DSNAME=load.library
```

THE ICHPWXC1 SUBROUTINE

The ICHPWXC1 subroutine provided in this article is for the ICHPWX01 RACF exit point. This particular exit processes the PWXPL parameter list and pulls out the userid and password value that is about to be set for this userid. It then displays these values through a console message. This is not something you would normally do, but the purpose of this exit subroutine is simply to demonstrate potential capability. The PWXPL structure definition has been created from the ICHPWXP DSECT mapping macro by using the CBC3DSCT program to convert the Assembler DSECT into a C structure definition.

To create a usable ICHPWX01 exit, perform the following:

- Assemble the XITFRMWK stub program. Be sure to specify ICHPWXC1 on the call macro as follows:

```
CALL ICHPWXC1
```

- Compile the ICHPWXC1 subroutine. Before running the compile, be sure to convert all occurrences of '[' to X'AD' and ']' to X'BD'. Be sure to run the C compile with the RENT option.
- To use the RENT compile option, the compile step object code (created in the previous step) must be run through the pre-linker. If the object module created in the previous step is called ICHPWXCC, use the following sample JCL to create the ICHPWXC1 object module:

```
//PLKED1  EXEC PGM=EDCPRLK,PARM='UPCASE',
//    REGION=2Ø48K
//SYSMSGS  DD  DSNAME=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  DSN=TCPIP.SEZARNT1,DISP=SHR
//SYSOBJ   DD  DSN=object.code.pds,DISP=SHR
//SYSMOD   DD  DSN=object.code.pds(ICHPWXC1),DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  INCLUDE SYSOBJ(ICHPWXCC)
```

- Create the ICHPWX01 load module using the following linkedit JCL:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT   DD    DSN=object.code.pds,DISP=SHR
//SYSLMOD  DD    DSN=a.lpa.library,DISP=SHR
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLIN   DD    *
   INCLUDE OBJECT(XITFRMWK)
   INCLUDE OBJECT(ICHPWXC1)
   ENTRY   XITFRMWK
   SETCODE AC(1)
   NAME    ICHPWXØ1(R)
```

Unless you have a way of dynamically activating your LPA resident RACF exits, you will need an IPL with a CLPA option to activate the ICHPWX01 exit for test purposes.

SUMMARY

Traditionally, subsystem exits have been coded in Assembler. As OS/390 Assembler coding skills erode, using a high-level language such as IBM's C/C++ may become a viable option for those site-specific customization requirements. Try out the provided exits and maybe you will see ways of working this exit coding technique into your environment.

XITFRMWK.ASM

```
*-------------------------------------------------------------------*
*                                                                   *
*   The XITFRMWK program is a representative model of an Assembler  *
*   program that can be used as an exit point entry program stub    *
*   that calls a C program subroutine for primary exit code logic.  *
*                                                                   *
*   The CBC3DSCT utility program can be used to create the C        *
*   structure definition for any exit control blocks necessary.     *
*                                                                   *
*   The following macros are required to set up and take down the   *
*   environment that makes this operation possible:                 *
*      CEEENTRY                                                     *
*      CEETERM                                                      *
*      CEEPPA                                                       *
*      CEEDSA                                                       *
*      CEECAA                                                       *
```

```
*                                                                     *
*   The following restrictions are in effect for the CEEENTRY macro:  *
*      With MAIN=YES:                                                 *
*         BASE= can be any register R3-R11 (R11 is the default)       *
*         R12 is to the address of the CEECAA                         *
*         R13 is to the address of the CEEDSA                         *
*         PARMREG= the parameter list address (R1 is the default)     *
*                                                                     *
*   This exit driver can be used for any exit.  Simply change the     *
*   CALL macro instruction to reflect the appropriate module for      *
*   your requirements.                                                *
*                                                                     *
*---------------------------------------------------------------------*
XITFRMWK CEEENTRY PPA=MAINPPA,     ** Label of CEEPPA mapping macro  **X
               AUTO=WORKSIZE,      ** Size of DSA & local work area  **X
               MAIN=YES,           ** This rtn is main rtn in enclave**X
               EXECOPS=NO,         ** No runtime options in parms    **X
               PARMREG=R1,         ** R1 is the default parm reg     **X
               BASE=R11,           ** R11 is the default base reg    **X
               PLIST=HOST          ** Standard JCL PARM= parm list   **
*---------------------------------------------------------------------*
         USING WORKAREA,R13        Set addressability to temp storage
*---------------------------------------------------------------------*
*   Uncomment whichever of the following CALL macro invocations you   *
*   desire prior to assembling this code.                             *
*---------------------------------------------------------------------*
**       CALL  IRREVXC1            ** IRREVXC1 is the C subroutine   **
**       CALL  ICHPWXC1            ** ICHPWXC1 is the C subroutine   **
         LR    R5,R15              Copy the return code
*---------------------------------------------------------------------*
RETURN   DS    ØH
         CEETERM  RC=(R5),MF=(E,CEETERMW)
*---------------------------------------------------------------------*
         LTORG ,
*---------------------------------------------------------------------*
MAINPPA  CEEPPA                   Constants describing the code block
*---------------------------------------------------------------------*
*        The Workarea and DSA                                         *
*---------------------------------------------------------------------*
WORKAREA DSECT
         ORG    *+CEEDSASZ         Leave space for the DSA fixed part
CALLLST  CALL  ,(Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø),VL,MF=L
CEETERMW CEETERM MF=L
WORKSIZE EQU    *-WORKAREA
*---------------------------------------------------------------------*
         CEEDSA ,                  Mapping of the Dynamic Save Area
         CEECAA ,                  Mapping of the Common Anchor Area
*---------------------------------------------------------------------*
RØ       EQU   Ø
R1       EQU   1
```

```
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*-------------------------------------------------------------------*
ENTREGS  CSECT
ENTREGS  AMODE 31
ENTREGS  RMODE ANY
         ENTRY ENTREGS
         GBLB  &CEESTART
&CEESTART SETB 1
         EDCPRLG BASEREG=R12,DSALEN=WORKLEN
         USING REGSWORK,R13
         L     R2,Ø(,R1)           Get register vector address
         L     R3,4(,R13)          Get savearea -1 address
         L     R4,4(,R3)           Get savearea -2 address
         L     R5,4(,R4)           Get savearea -3 address
         MVC   Ø(52,R2),2Ø(R5)     Copy RØ-R12 values
         ST    R5,52(,R2)          Save R13 value
         MVC   56(8,R2),12(R5)     Copy R14-R15 values
         LA    R15,Ø               Set return code
         EDCEPIL
         LTORG ,
*-------------------------------------------------------------------*
REGSWORK EDCDSAD
WORKLEN  EQU   *-REGSWORK
         END
```

IRREVXC1.C

```
/*
 *  This is a C program subroutine which is invoked from a calling
 *  Assembler program that has established the main() enclave.  The
 *  IRREVXFC subroutine will be invoked via a standard Assembler
 *  CALL macro from the calling program.
 *
 *  This subroutine expects one incoming parameter.
 *     PARM1:  is the address of the EVXPL.
 *             The structure definition has been created using the
```

```
 *             CBC3DSCT program which is used to create C structure
 *             definitions from Assembler DSECTs.
 *
 *  For demonstration purposes, this exit examines the RACF command
 *  buffer looking for LISTUSER commands.  If a LISTUSER command is
 *  detected, this exit will force on the displaying of TSO and OMVS
 *  segment information regardless of what was requested in the
 *  original command.  Although this may have practical use in your
 *  environment, this code and the accompanying Assembler driver
 *  are primarily provided to demonstrate a technique that could be
 *  employed to allow for the coding of RACF system exits using
 *  IBM C/C++.
 *
 *  This subroutine is used in conjuction with the XITFRMWK Assembler
 *  program to create the IRREVXØ1 RACF exit.  This exit point is
 *  entered in supervisor state, key Ø, so be careful.  The IRREVXØ1
 *  exit is eligible to be managed by the OS/39Ø dynamic exit manager.
 *  It can be enabled and disabled with the following operator
 *  commands:
 *
 *  SETPROG EXIT,ADD,EXITNAME=IRREVXØ1,MODNAME=exitname,
 *          DSNAME=catalogd.dsn
 *
 *  SETPROG EXIT,DELETE,EXITNAME=IRREVXØ1,MODNAME=exitname
 *
 *  where 'exitname' is the member name of the IRREVXØ1 exit that
 *  resides in the cataloged dataset specified in 'catalogd.dsn'.
 *
 *  Before compiling this program, remember to convert all occurrences
 *  of '[' to x'AD' and ']' to x'BD'
 *
 */

#pragma runopts(TRAP(OFF))      // Allows code to run supervisor state
#pragma linkage(ENTREGS, OS)    // Define the ENTREGS external routine

#include <string.h>
#include <stdlib.h>
#include <__messag.h>
#include <fcntl.h>

#pragma pack(packed)

struct EVXPL {
  void          *EVXLEN;   /* Length address:                */
  void          *EVXCALLR; /* Caller address:                */
  void          *EVXFLAGS; /* Flag byte address:             */
  void          *EVXCMBUF; /* Command buffer address:        */
  void          *EVXACEE;  /* ACEE address:                  */
  void          *EVXWORK;  /* Communication word address:    */
```

```
  void          *EVXCMDRC; /* Command return code address:    */
  void          *EVXABCD;  /* Abend code address:              */
  void          *EVXSRCND; /* Command source node address:    */
  void          *EVXSRCUS; /* Command source user ID address: */
  void          *EVXMSSG;  /* Message text address:            */
  };

/* Values for field "EVXCALLR" */
#define EVXADDGR Øx01 /* ADDGROUP                              */
#define EVXADDSD Øx02 /* ADDSD                                 */
#define EVXADDUS Øx03 /* ADDUSER                               */
#define EVXALTDS Øx04 /* ALTDSD                                */
#define EVXALTGR Øx05 /* ALTGROUP                              */
#define EVXALTUS Øx06 /* ALTUSER                               */
#define EVXCONNE Øx07 /* CONNECT                               */
#define EVXDELDS Øx08 /* DELDSD                                */
#define EVXDELGR Øx09 /* DELGROUP                              */
#define EVXDELUS Øx0A /* DELUSER                               */
#define EVXLISTD Øx0B /* LISTDSD                               */
#define EVXLISTG Øx0C /* LISTGRP                               */
#define EVXLISTU Øx0D /* LISTUSER                              */
#define EVXPASSW Øx0E /* PASSWORD                              */
#define EVXPERMI Øx0F /* PERMIT                                */
#define EVXRALTE Øx1Ø /* RALTER                                */
#define EVXRDEFI Øx11 /* RDEFINE                               */
#define EVXRDELE Øx12 /* RDELETE                               */
#define EVXREMOV Øx13 /* REMOVE                                */
#define EVXRLIST Øx14 /* RLIST                                 */
#define EVXSEARC Øx15 /* SEARCH                                */
#define EVXSETRO Øx16 /* SETROPTS                              */

/* Values for field "EVXFLAGS" */
#define EVXPRE   Øx8Ø /* Pre-processing call                   */
#define EVXPOST  Øx4Ø /* Post-processing call                  */
#define EVXOPER  Øx2Ø /* Command issued as operator command    */
#define EVXPARM  Øx1Ø /* Command issued from RACF parmlib      */
#define EVXAT    Øx08 /* Command directed with AT or ONLYAT    */
#define EVXACD   Øx04 /* Command directed with automatic       */
#define EVXRASP  Øx02 /* Command execution in RACF subsystem   */
#define EVXABND  Øx01 /* Command abended during execution      */

struct CMDBUF {
  short int      CMDBUFL; /* length of command buffer         */
  short int      CMDBUFO; /* offset in command buffer to the  */
  };

#pragma pack(reset)

#define REG_DISPLAY Ø          // Set != Ø to display register content
```

```
void Console_Msg(char *msg_buf)

/*
 * The Console_Msg() function is used to issue a message to the
 * operator console similar to the function of a WTO macro in
 * an Assembler program.  To issue a multi-line message, include
 * newline characters ("\n") within the data wherever you want
 * your message to start on a new display line at the console.
 */

{
  int  msg_i;
  int  msg_j;
  char con_buf[256];
  struct __cons_msg *msg;

  msg = (struct __cons_msg *)con_buf;

/*
 * Copy the message buffer and message length into the message
 * structure and use the __console() function to issue the message
 * to the console.
 */
  msg->__format.__f1.__msg = (char *)msg_buf;
  msg->__format.__f1.__msg_length = strlen(msg_buf);
  msg_i = __console(msg,NULL,&msg_j);

  return;
}

int IRREVXC1(struct EVXPL EVXPL)

{
int* i;
int* k;
int* l;
int  j;
int  m;
int  p;
int  q;
int  r;
int  term_char;
int  omvs_flag;
int  tso_flag;
char* mm;
char* ii;
char* jj;
char flags[3];
char flags_save[3];
char caller[2];
```

```
char caller_save[2];
char source_userid[9];
char msgarea[256];
unsigned int reg_vector[16];

/*
 * Capture register values as they existed on entry to the exit
 * point.  They may be of use at a later point.
 */

j = ENTREGS(&reg_vector);

/*
 * If you want to display the contents of the registers as they
 * existed on entry to the called exit, set the value of REG_DISPLAY
 * in the prior #define statement to a value other than Ø.
 */
if (REG_DISPLAY != Ø)
{
  for (j=Ø; j<16; j++)
  {
    sprintf(msgarea,"R%d on entry: %#Ø1Øx\Ø",j,reg_vector[j]);
    Console_Msg(msgarea);
  }
}

/*
 *  Extract and save the EVXFLAGS information for future reference.
 */

  for (m=Ø; m<3; m++)
  {
    flags_save[m] = Ø;
  }
  memcpy(flags_save+Ø,(char *)(EVXPL.EVXFLAGS)+Ø,2);

/*
 *  Extract and save the EVXCALLR information for future reference.
 */

  for (m=Ø; m<2; m++)
  {
    caller_save[m] = Ø;
  }
  memcpy(caller_save+Ø,(char *)(EVXPL.EVXCALLR)+Ø,1);

/*
 *  Extract the EVXLEN value which indicates the number of
 *  fullwords contained in the EVXPL.  Indicate this value
 *  in a message issued to the console using the __console()
```

```
 *  function call.
 */

if (flags_save[0] == 0x80)
{
  j = (int)(*(int *)(EVXPL.EVXLEN));
  i = &(*(int *)(EVXPL.EVXLEN));

  sprintf(msgarea,"EVXLEN   addr: %#010x   EVXLEN is %d fullwords.\0",
          i,j);
  Console_Msg(msgarea);
}

/*
 *  Extract the EVXCALLR value which indicates the command that
 *  triggered the exit call.  Indicate this value
 *  in a message issued to the console using the __console()
 *  function call.
 */

if (flags_save[0] == 0x80)
{
  for (m=0; m<2; m++)
  {
    caller[m] = 0;
  }
  memcpy(caller+0,(char *)(EVXPL.EVXCALLR)+0,1);
  mm = &((*((char *)(EVXPL.EVXCALLR))));

  sprintf(msgarea,"EVXCALLR addr: %#010x   EVXCALLR is %#04x\0",
          mm,caller[0]);
  Console_Msg(msgarea);
}

/*
 *  Extract the EVXFLAGS value which indicates the flag values
 *  in effect for this exit call.  Indicate this value
 *  in a message issued to the console using the __console()
 *  function call.
 */

  for (m=0; m<3; m++)
  {
    flags[m] = 0;
  }
  memcpy(flags+0,(char *)(EVXPL.EVXFLAGS)+0,2);
  mm = &((*((char *)(EVXPL.EVXFLAGS))));

  sprintf(msgarea,"EVXFLAGS addr: %#010x   EVXFLAGS are %#04x%02x\0",
          mm,flags[0],flags[1]);
```

```
  Console_Msg(msgarea);

  m = Ø;
  mm = (char *)&m;
  memcpy(mm+2,flags+Ø,2);

/*
 *  Extract the EVXSRCUS value which indicates the command
 *  source userid for the issued command.  Indicate this value
 *  in a message issued to the console using the __console()
 *  function call.
 */

if (flags_save[Ø] == Øx8Ø)
{
  mm = &((*((char *)(EVXPL.EVXSRCUS))));
  strncpy(source_userid,mm,8);
  source_userid[8] = Ø;

  if (strncmp(source_userid,"        ",8) != Ø)
  {
    sprintf(msgarea,"EVXSRCUS addr: %#Ø1Øx   EVXSRCUS is %s\Ø",
            mm,source_userid);
    Console_Msg(msgarea);
  }
}

/*
 *  Extract the EVXACEE address which indicates the ACEE in
 *  effect for the current task.  Indicate this value
 *  in a message issued to the console using the __console()
 *  function call.
 */

if (flags_save[Ø] == Øx8Ø)
{
  mm = &((*((char *)(EVXPL.EVXACEE))));
  strncpy(source_userid,mm+21,8);
  source_userid[8] = Ø;

  sprintf(msgarea,"EVXACEE  addr: %#Ø1Øx   Task userid is %s\Ø",
          mm,source_userid);
  Console_Msg(msgarea);
}

/*
 *  Extract the EVXCMBUF address which contains the address of
 *  the command buffer for the issued command.  Indicate this value
 *  in a message issued to the console using the __console()
 *  function call.
```

```
 */

if (flags_save[0] == 0x80)
{
  mm = &((*((char *)(EVXPL.EVXCMBUF))));
  ii = (char *)(&j);
  j = 0;
  memcpy(ii+2,(char *)(EVXPL.EVXCMBUF)+0,2);

  sprintf(msgarea,"EVXCMBUF addr: %#010x   Buffer length is %d\0",
          mm,j);
  Console_Msg(msgarea);

  q = j;                //  Save command buffer length

  if (j > 70)
  {
    j = 70;
  }

  strcpy(msgarea,"Command buffer content <= 70 bytes:  \0");
  strncpy(msgarea+strlen(msgarea),mm+4,j);

  for (m=strlen(msgarea)-1; m>=0; m--)
  {
    if (msgarea[m] == 64)
    {
      msgarea[m] = 0;
    }
    else
    {
      break;
    }
  }

  Console_Msg(msgarea);

/*
 *  The command buffer has been echoed to the console.  Check to see
 *  if this is a LISTUSER (LU) command and, if it is, force the
 *  command to include TSO and OMVS segment information displayed.
 */

  if (caller_save[0] == EVXLISTU)
  {
//  Get offset of first byte past the LISTUSER command.

    jj = (char *)(&r);
    r = 0;
    memcpy(jj+2,(char *)(EVXPL.EVXCMBUF)+2,2);
```

```
/*
 *  At this point, q contains the command buffer length and r
 *  contains the offset of the first byte past the LISTUSER command.
 */

//  Skip to the userid list.

    for (p = r; p<q; p++)
    {
      if (mm[4+p] != 64)
      {
        break;
      }
    }
    if (p >= q-1)
    {
      goto BADCMD;
    }

//  Skip past the userid list.

    term_char = 64;            //  Set default termination character.
    if (mm[4+p] == 77)         //  Does userid list start with '('?
    {
      term_char = 93;          //  Set termination character to ')'.
    }

    for (r=p; r<q; r++)
    {
      if (mm[4+r] == term_char)
      {
        break;
      }
    }
    if (r >= q-1)
    {
      goto BADCMD;
    }
/*
 *  At this point, r contains the offset into the command buffer at
 *  the point where the userid list ends.
 */
    omvs_flag = Ø;
    for (p=r; p<q-6; p++)
    {
      if (strncmp(mm+4+p,"NOOMVS",6) == Ø)
      {
        mm[4+p] = 64;
        mm[4+p+1] = 64;
```

```
    omvs_flag = 1;
  }
}
tso_flag = Ø;
for (p=r; p<q-5; p++)
{
  if (strncmp(mm+4+p,"NOTSO",5) == Ø)
  {
    mm[4+p] = 64;
    mm[4+p+1] = 64;
    tso_flag = 1;
  }
}
if (omvs_flag == Ø)
{
  for (p=r; p<q-4; p++)
  {
    if (strncmp(mm+4+p,"OMVS",4) == Ø)
    {
      omvs_flag = 1;
      break;
    }
  }
}
if (tso_flag == Ø)
{
  for (p=r; p<q-3; p++)
  {
    if (strncmp(mm+4+p,"TSO",3) == Ø)
    {
      tso_flag = 1;
      break;
    }
  }
}
if (omvs_flag == Ø)
{
  for (p=r; p<q-6; p++)
  {
    if (strncmp(mm+4+p,"      ",6) == Ø)
    {
      strncpy(mm+4+p+1,"OMVS",4);
      mm[4+p+5] = 64;
      break;
    }
  }
}
if (tso_flag == Ø)
{
  for (p=r; p<q-5; p++)
```

```
      {
        if (strncmp(mm+4+p,"     ",5) == Ø)
        {
          strncpy(mm+4+p+1,"TSO",3);
          mm[4+p+4] = 64;
          break;
        }
      }
    }

  }

}
BADCMD:
  return(Ø);
}
```

ICHPWXC1.C

```
/*
 *  This is a C program subroutine which is invoked from a calling
 *  Assembler program that has established the main() enclave.  The
 *  ICHPWXØC subroutine will be invoked via a standard Assembler
 *  CALL macro from the calling program.
 *
 *  This subroutine expects one incoming parameter.
 *     PARM1:  is the address of the PWXPL.
 *             The structure definition has been created using the
 *             CBC3DSCT program which is used to create C structure
 *             definitions from Assembler DSECTs.
 *
 *  This subroutine is used in conjuction with the XITFRMWK Assembler
 *  program to create the ICHPWXØ1 RACF exit.  This exit point is
 *  entered in supervisor state, key Ø, so be careful.
 *
 *  Before compiling this program, remember to convert all occurrences
 *  of '[' to x'AD' and ']' to x'BD'
 *
 */
#pragma runopts(TRAP(OFF))
#include <string.h>
#include <stdlib.h>
#include <__messag.h>
#include <fcntl.h>
#pragma pack(packed)
struct PWXPL {
  void          *PWXLEN;   /* Length address:                    */
  void          *PWXCALLR; /* Caller address:                    */
  void          *PWXCPPL;  /* CPPL address:                      */
```

```
  void          *PWXNEWPW; /* NEWPASS address:                    */
  void          *PWXINTVL; /* INTERVAL address:                   */
  void          *PWXUSRID; /* Userid address:                     */
  void          *PWXWA;    /* Exit work area address:             */
  void          *PWXCURPW; /* Current password address points     */
  void          *PWXPLCDA; /* Password Last Change Date           */
  void          *PWXACEE;  /* ACEE address:                       */
  void          *PWXGROUP; /* Group name address:                 */
  void          *PWXINSTL; /* Installation data address:          */
  void          *PWXPWHST; /* Password history address:           */
  void          *PWXFLAG;  /* Flag byte address:                  */
  void          *PWXPLCD4; /* Password Last Change Date      @Ø1A */
  };
/* Values for field "PWXCALLR" */
#define PWXRINIT Øx Ø1 /* RACINIT                                */
#define PWXPWORD Øx Ø2 /* PASSWORD Command                       */
#define PWXALTUS Øx Ø3 /* ALTUSER Command                        */
/* Values for field "PWXFLAG" */
#define PWXCTEXT Øx ØØ /* Clear text form                        */
#define PWXETEXT Øx Ø1 /* Encrypted form (If                     */
#define PWXPTKT  Øx Ø2 /* Passticket is passed in the old @P1A */
#pragma pack(reset)
/*  Indicate to the compiler that standard OS linkage will be used.  */
#ifdef __cplusplus
 extern "OS" int ICHPWXC1(struct PWXPL*);
#else
// #pragma linkage (ICHPWXC1,OS)
#endif
int ICHPWXC1(struct PWXPL PWXPL)
{
int* i;
int  j;
int  msg_i;
int  msg_j;
unsigned int cvtloc;
unsigned int cvt;
unsigned int rcvt;
char* mm;
char userid[9];
char password[9];
char msgarea[256];
char con_buf[256];
struct __cons_msg *msg;
msg = (struct __cons_msg *)con_buf;
i = &(*(int *)ICHPWXC1);
sprintf(msgarea,"Module entry addr: %#Ø1Øx\Ø",i);
msg->__format.__f1.__msg = (char *)&msgarea;
msg->__format.__f1.__msg_length = strlen(msgarea);
msg_i = __console(msg,NULL,&msg_j);
cvtloc = 16;
```

```
cvt = *(unsigned int *)cvtloc;
rcvt = *(unsigned int *)(cvt + 0x000003e0);
if (strncmp((char *)(rcvt + 0),"RCVT",4) == 0)
{
  sprintf(msgarea,"RCVT located at addr: %#010x\0",rcvt);
}
else
{
  sprintf(msgarea,"RCVT not located at addr: %#010x\0",rcvt);
}
msg->__format.__f1.__msg = (char *)&msgarea;
msg->__format.__f1.__msg_length = strlen(msgarea);
msg_i = __console(msg,NULL,&msg_j);
mm = &((*((char *)(PWXPL.PWXUSRID))));
strncpy(userid,mm+1,8);
userid[8] = 0;
for (j=0; j<8; j++)
{
  if (userid[j] == 64)
  {
    userid[j] = 0;
    break;
  }
}

mm = &((*((char *)(PWXPL.PWXNEWPW))));

strncpy(password,mm+1,8);
password[8] = 0;
for (j=0; j<8; j++)
{
  if (password[j] == 64)
  {
    password[j] = 0;
    break;
  }
}

sprintf(msgarea,"ICHPWX01 userid: %s  password: %s.\0",
        userid,password);
msg->__format.__f1.__msg = (char *)&msgarea;
msg->__format.__f1.__msg_length = strlen(msgarea);
msg_i = __console(msg,NULL,&msg_j);
return(0);
}
```

Rudy Douglas
System Programmer (Canada)

RACF in focus – implementing audit features

This is a regular column focusing on specific aspects of RACF. In this issue, we will discuss various auditing and logging options available in RACF, and how best to implement them.

BACKGROUND

First, when we talk of auditing in RACF, we are talking about the logging of events and activities that occur to the System Management Facility (SMF) datasets of the operating system. In this discussion, auditing and logging are used interchangeably and mean the same thing.

SMF data contains not only RACF logging, but various other types of system activity logging as well. RACF logging in SMF is separated from these other activities by the type of SMF records that are generated.

Typically, SMF data (and therefore RACF logging) at an installation is retained for many years, so you can always go back and trace some event that occurred in the past.

Installations have considerable freedom in choosing what RACF activity and events they want audited and which ones to ignore. It is important to choose your auditing options carefully – too much auditing often means that you do not have the time to review the audit reports in a meaningful way, and important events may go unnoticed. Too little auditing, of course, means that you are not capturing the important security-related events that are occurring in your system.

Some of the RACF auditing activities, especially those related to violations and warnings, are also seen in the system log and the operator console in real time, that is, as they occur.

We shall now discuss the various auditing options available in RACF.

AUDITING USER ACTIVITY

If you want to audit and monitor all activity of a user, you can use the UAUDIT operand of the alter user command:

```
ALTUSER USER123 UAUDIT
```

This will log all datasets and other resources the user references, and all the RACF commands the user enters. The logging will be in effect until you remove the UAUDIT attribute from the user profile:

```
ALTUSER USER123 NOUAUDIT
```

UAUDIT will log not only the violations, but successful accesses as well. This will generate lots of logging and, for this reason, should be used appropriately. It is meant to be used when you suspect a user of malicious intentions. Sometimes it is also used for debugging purposes, for example when you want to find out what RACF profiles are accessed by a user in certain cases.

UAUDIT should not be used to log all activities of users with special powers, such as operations or special. Some RACF administrators do this, under the mistaken assumption that all activities of users with special powers should be monitored. This will needlessly create a lot of logging activity. There are other ways to monitor the activities of users with special powers – see the section *AUDITING USERS WITH SPECIAL ATTRIBUTES* below.

AUDITING RESOURCES AT THE PROFILE LEVEL

The AUDIT specification of the RACF commands ADDSD, ALTDSD, RALTER, and RDEFINE relate to auditing resources covered by dataset and resource profiles.

You can indicate, for each profile, whether you want to log successful accesses, or failures, or both. If you want to see what kind of logging is in effect for a profile, you can list the profile.

In general, you want to log only successful accesses for those profiles that cover highly sensitive data. Over-use of successful auditing will generate a lot of logging, and needlessly clutter your RACF reports. (The DATASET profiles are used in these examples, but it applies to other classes, although the command will be RALTER.) The command:

```
ALTDSD 'ds_profile_name' AUDIT(SUCCESS(READ))
```

will log all successful accesses, but not failures (violations).

For most profiles, you would want to see only the failures or violations. This is highly recommended. The command to do this is:

```
ALTDSD 'ds_profile_name' AUDIT(FAILURES(READ))
```

If you want to see both successes and failures, the command to use is:

```
ALTDSD 'ds_profile_name' AUDIT(ALL(READ))
```

USING THE GLOBALAUDIT OPERAND

Like the AUDIT specification, GLOBALAUDIT also applies to individual profiles in the dataset and general resource classes. The command syntax for GLOBALAUDIT is also very similar to that of AUDIT. The difference is that GLOBALAUDIT is meant for auditors (who have the AUDITOR attribute or the GROUP-AUDITOR attribute).

An auditor can, using GLOBALAUDIT, increase the amount of logging for dataset and general resource profiles, as specified in the AUDIT operand. They cannot decrease the loggings specified in the AUDIT specification.

For example, if a dataset profile has the specification:

```
AUDIT(FAILURES(UPDATE))
```

it indicates logging of all violations for UPDATE, CONTROL, and ALTER (but not READ).

An auditor may also wish to log violations at the READ level, so

they can specify:

```
GLOBALAUDIT(FAILURES(READ))
```

When you list a DATASET profile or a general resource profile, you see both specifications, AUDIT and GLOBALAUDIT, for that profile.

AUDITING RESOURCES AT THE CLASS LEVEL

In addition to auditing resources at the profile level, RACF also provides for auditing specifications at the class-level. This can be helpful because you may have many profiles in a class (for example the DATASET class), and you may not have consistent AUDIT specifications for all of them.

If you want consistency in auditing at the class level, you can use the LOGOPTIONS specification of the **SETROPTS** command. We will use the DATASET class as an example, but this applies to all classes at your installation.

All auditing for the DATASET class is suppressed by:

```
SETROPTS LOGOPTIONS(NEVER(DATASET))
```

This overrides any auditing requirements you may have specified in individual profiles in the DATASET class.

Conversely, the command:

```
SETROPTS LOGOPTIONS(ALWAYS(DATASET))
```

will log all activity to resources in the DATASET class, overriding any specification at the profile level.

If you want to see all successes for all profiles in the DATASET class, you can use the following command. Successes will be logged in addition to what is already specified in the individual profiles:

```
SETROPTS LOGOPTIONS(SUCCESSES(DATASET))
```

If you want to see all failures for all profiles in the DATASET class, you can use the following command. Failures will be

logged in addition to what is already specified in the individual profiles. This is highly recommended, in case the logging of failures is missing in some of the profiles:

```
SETROPTS LOGOPTIONS(FAILURES(DATASET))
```

If you want to respect the audit requirements as specified in individual profiles, you can do so by specifying the following command. It is also the default specification if you do not specify one of the above LOGOPTIONS keywords:

```
SETROPTS LOGOPTIONS(DEFAULT(DATASET))
```

AUDITING USERS WITH SPECIAL ATTRIBUTES

Activities of users having either the SPECIAL attribute or OPERATIONS attribute should be logged. But you should only log – and monitor – those activities that pertain to the use of their special powers.

This is done by the **SETROPTS** commands (you need to be authorized to use them).

To log users with SPECIAL or group-SPECIAL attribute use the command:

```
SETROPTS SAUDIT
```

To log users with OPERATIONS or group-OPERATIONS authority use the command:

```
SETROPTS OPERAUDIT
```

The defaults for both these specifications are NOSAUDIT and NOOPERAUDIT respectively, so you will have to explicitly set the SAUDIT and OPERAUDIT specifications if you want to benefit from the use of these options. You may even want to check what is specified at your installation by using the LIST operand of the **SETROPTS** command.

If you have SAUDIT and OPERAUDIT in effect, it indicates that you want to log all accesses granted to these users because of their special powers. Other superfluous logging for special

userids will not occur – for example where profiles already allow access to the special users, the activity to those profiles will not be logged because of these **SETROPTS** specifications.

AUDITING CHANGES MADE TO PROFILES

You may wish to log changes to all profiles made by RACF commands, such as ADDSD, ALTDSD, etc. This is not to be confused with auditing resource accesses protected by the profiles, which was discussed under the section *AUDITING RESOURCES AT THE PROFILE LEVEL* above.

If you want to log changes made to profiles by RACF commands, you can do so on a class by class basis. The command is:

```
SETROPTS AUDIT(DATASET)
```

This will record changes made to profiles in the DATASET class. Other valid class names are any of the general resource classes at your installation, or USER, or GROUP.

If you specify:

```
SETROPTS AUDIT(*)
```

you will see changes made to all profiles in all classes at your installation. This is recommended, and has the added benefit that if you were to activate a new class, it would automatically be covered.

Note that the default for this option is NOAUDIT(*), so no logging occurs for any of the classes unless you take specific action.

AUDITING FAILURES TO RACF COMMANDS

Just as you want to see violations for accesses to datasets and other resources, you may want to see all failures of RACF commands themselves. You can audit all violations detected by RACF commands by specifying the following command (you need to be authorized to use the command):

```
SETROPTS CMDVIOL
```

This command will ignore violations caused to the list commands, such as LISTUSER, but report on all other violations to RACF commands. For example, it will report on violations that occur because a user is not authorized to modify a particular profile.

SOME AUTOMATIC LOGGING

Certain events automatically get logged. For example, if PROTECTALL(FAILURES) is in effect at your installation, and a user with the SPECIAL attribute requests access to an unprotected dataset, RACF will audit this event and issue a warning message to the operator console.

In the above case, if it were a 'trusted' started task requesting access to an unprotected dataset, the auditing would still occur, but no warning message would be generated.

RACF also logs all activity to a profile in WARN mode if access that would otherwise have been denied is being granted because the profile is in WARN mode.

CONCLUSION

In this article we discussed various logging options. But logging is meaningless if you do not follow through with reports from these logs. These reports do not come out automatically. You need to generate them using SAS or some other vendor's product that will extract useful data from these logs.

And you need to have procedures, policies, and practices in place that will ensure the reports are reviewed by appropriate people, on a regular basis, and action is taken whenever necessary.

In general, the options you specify at the global level, using the **SETROPTS** command, are more important than the specifications in the profiles themselves. You can control logging at the CLASS level, which ensures that you do not miss any logging because of inappropriate specifications at the profile level.

And finally, a suggestion – even though SMF records (and therefore, RACF logging activities) are retained for several years at most installations, it would be a good idea to separate the RACF portion of the log on a daily, weekly, monthly, and even yearly basis. That way, if you ever need to go back and trace some RACF activity you will not have to rely on the general SMF records. Your log will be independent of the rest of the SMF records.

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

Dinesh Dattani
Security Consultant
Toronto (Canada)

Critical Path and Calendra have announced a worldwide agreement for Critical Path to integrate and offer Calendra's technology as part of Critical Path's packaged identity provisioning solutions.

The new offerings enable organizations to provision users in accordance with business approval processes and migrate large numbers of accounts during application rollout in heterogeneous environments. In addition, the new solutions help customers to automate change management and implement user self-service in order to simplify and reduce the cost of Help Desk operations.

Calendra has directory management technology, Critical Path has an identity management platform. Critical Path's Meta-Directory includes built-in connectors for RACF mainframe security and other operating system platforms, as well as databases from Oracle, Sybase, and IBM; enterprise applications such as SAP HR, Lotus Notes, and Microsoft Exchange.

For further information contact:
Critical Path, 350 The Embarcadero, San Francisco, CA 94105-1204, USA.
Tel: (415) 541 2500.
URL: http://www.criticalpath.net.
Calendra, 13800 Coppermine Road, Herndon, VA 20171, USA.
Tel: (609) 273 1438.
URL: http://www.calendra.com.

* * *

OpenNetwork Technologies has announced Version 5.1 of Universal Identity Platform (Universal IdP). The new version provides expanded capabilities with 13 new connectors that automate the provisioning and de-provisioning of user accounts and simplify ongoing password management. Additionally, Universal IdP 5.1 has expanded its Web Single Sign-On (SSO) capabilities with added support for Apache, SAP Portal, and the latest releases of PeopleSoft and BEA WebLogic.

Universal IdP 5.1 leverages the data synchronization capabilities of Microsoft Identity Integration Server (MIIS) to enable role-based provisioning across a broad range of enterprise resources, including 13 new systems not currently supported by MIIS. It integrates MIIS with RACF, as well as AS/400, SAP R/3, PeopleSoft, Solaris, AIX, and others. Additionally, Universal IdP 5.1 includes a Web Services framework that simplifies the development of additional connectors to custom applications.

For further information contact:
OpenNetwork Technologies, 13577 Feather Sound Drive, Clearwater, FL 33762, USA.
Tel: (877) 561 9500.
URL: http://www.calendra.com.

* * *