“RACF commands” to administer UNIX security

Author: Bruce R. Wells

z/OS Security Server RACF

brwells@us.ibm.com

Last updated: 01/24/2020

Change Date	Change Description
01/24/2020	Introduction of this download

I gratefully acknowledge Bill Schoen of IBM z/OS development for his advice, patience with my questions and user errors, and even a few code snippets along the way!

This download is dedicated to my long-time colleague, mentor, and friend, John Dayka. Read more here.

Table of Contents

Background

The security data associated with UNIX files and directories is managed by a set of shell commands. Many clients have expressed a wish to protect these objects by (path)name as RACF profiles. However, such an approach is in direct conflict with the architecture of a UNIX file system, in which path names are convenient ways to reference an object but aren't necessarily permanent or unique.

Security administrators generally prefer to use RACF commands. The UNIX shell, and its suite of commands, are unfamiliar to the RACF administrator and can be daunting to learn. (Did you know that there are seven different shell commands to update security attributes? Or that to display these attributes requires two different commands, and the use of several different options on the 'ls' command?) The lack of understanding can lead to mistakes, which can open vulnerabilities in the protection of UNIX-based data.

This download provides a set of REXX execs that accept a command syntax like that of RACF TSO commands and use UNIX syscall support in REXX to display or update the appropriate security attributes for a file or directory.

- **ORLIST**: Displays UNIX security attributes in a format like the RLIST command. To provide a deeper understanding of how the object is actually protected, it also displays mount attributes of the file system data set in which the object resides, and optionally, the name of RACF profiles covering that file system where applicable.
- **ORALTER**: Allows for UNIX security management using a syntax like the RALTER command.
- **OPERMIT**: Manages UNIX access control lists (ACLs) using a syntax like the PERMIT command.

The idea is to *pretend* that path names are protected by RACF profiles, and to use 'RLIST', 'RALTER', and 'PERMIT' commands to view and manage them. Note that there isn't a 'ORDEFINE' or 'ORDELETE" command as part of this download. This is because, with UNIX files and directories, security information is part of the object itself. It would make no sense to create and delete the actual files and their contents as a security administrator.

Notes:

- SMF 80 records are only created for UNIX security information changes if SETROPTS LOGOPTIONS is in effect for the FSSEC class. I recommend `SETROPTS LOGOPTIONS(ALWAYS(FSSEC))` so that both successful and failed attempts to update UNIX security information are logged. That is, if you are logging all updates to RACF profiles, you also want to log all UNIX security changes.
- Further, RACF will not issue an ICH408I message for an authorization error unless an SMF record is created. There can be more than one authorization mechanism controlling a given change, and ICH408I is essential in determining the nature of the authorization failure for such cases.

The syntax for each command is described below, in the same format that RACF commands are documented in z/OS Security Server RACF Command Language Reference.

Invoking the execs

The standard way to invoke a REXX exec on the TSO command line is to specify the TSO 'execute' command, passing it the data set/member name in one quoted string, followed by the parameters in another quoted string. For example:

```
EX 'HLQ.UNIX.EXECS(ORALTER)' 'FSSEC /u/bruce/myFile OWNER(JDAYKA) GROUP(SECADMIN)'
```

But that is awkward, and burdensome on the user.

These execs will most closely resemble RACF commands by placing them into a PDS that is part of your SYSEXEC concatenation, so that they can be invoked the same way you would invoke a TSO command. For example:

```
ORALTER FSSEC /u/bruce/myFile OWNER(JOHND) GROUP(SECADMIN)
```

You can, of course, name these execs whatever you want to as you download into a PDS. One option is to name them the same as the model RACF command (RLIST, RALTER, and PERMIT). However, TSO will give preference to a TSO command with the same name as a REXX exec. The RACF TSO command will then fail with a syntax error, due to the syntax differences. However, you can prefix the exec with the percent sign (“%”) so that TSO understands that you want to run the exec instead of the TSO command. For example:

```
%RALTER FSSEC /u/bruce/myFile OWNER(JOHND) GROUP(SECADMIN)
```

To allow the ‘commands’ to look as much like the RACF commands as possible, we use the FSSEC (**F**ile **S**ystem **SEC**urity) class in positions where the RACF commands would require a class name. However, the class name is completely optional in all the commands, so feel free to save keystrokes by omitting it.

Using the execs – an overview

There are many ways in which you might find value in using this download. I suggest a possible progression:

1. Read this document. If you never even use the execs, the documentation can serve as a one-stop-shop to learn what the various UNIX file security attributes are, and what authority is required to view and modify them. Have you ever wondered what the sticky bit does? What authority is required to turn on the APF bit? Read the ‘Authorization required’ sections and the individual keyword descriptions to find out.
2. Try the execs. By default, they won’t update anything. Invoke the execs with no keywords to see the syntax and some examples. When you specify keywords, the execs will display the shell commands that would update security information based on the keywords you specify. They will also create a REXX shell script in your home directory containing these commands. The prolog contains the date and time of creation, the name of the exec, and the keywords you specified. The body contains the shell command(s) that would perform the requested change(s).
3. Use the execs for real. There are two ways to do this:
   a. Edit/Run the shell script that was generated. You don’t even have to leave the TSO command line! (Read on for details).
   b. Once you’ve gained some confidence, edit the exec to take it out of

‘noRun mode’, and the update(s) will be made immediately.

4. Teach others. You’re an expert now! Spread the word.

Here are more details, before we dive into individual command syntax:

As stated above, ORALTER and OPERMIT create a REXX shell script in your home directory named *exec-name*.script, where *exec-name* is ORALTER, OPERMIT, of whatever name you’ve chosen to save them as. You can change the name and location of the script that gets created by changing the `execName` configuration variable within the exec.

The file is created such that you, the owner, are the only person with permission to read, update, or execute it (i.e. the permission bits are rwx --- ---, or 700 in octal notation). If the script file already exists, it is replaced with each command issued.

Note that you need not even enter the UNIX shell to edit and execute your shell script. The z/OS OEDIT and OSHELL commands can be invoked directly from the TSO command line to edit and run your script.

For example, say that you enter the following command on the TSO command line:

```
oralter /u/jdayka/testFile owner(tsousr6) group(sys1) apf noprogram
nosticky setuid nosetgid audit(failures(rw)) perms(rwx------)
```

You can then view/edit the contents of the generated script by entering the following:

```
OEDIT ORALTER.script
```

This puts you into an ISPF edit session. When you are finished viewing/editing the script, you can leave your edit session and run the script by issuing:

```
OSHELL ORALTER.script
```

Note that OSHELL displays an exit status that is not 0, which might appear as if the script failed. This is not necessarily the case. For example, the following execution is successful:

```
OSHELL RC =  65280
OSHELL Exit Status =  255
```

If you enter the ORALTER command shown, you also see the following on the display as part of the command output:

```
This command would result in the following UNIX shell command(s):
chown TSOUSR6 /u/jdayka/testFile
chgrp SYS1 /u/jdayka/testFile
chmod 700 /u/jdayka/testFile
chmod u+s /u/jdayka/testFile
chmod g-s /u/jdayka/testFile
chmod -t /u/jdayka/testFile
extattr +a /u/jdayka/testFile
extattr -p /u/jdayka/testFile
chaudit rw+f /u/jdayka/testFile
```

If you want ORALTER and OPERMIT to update security information directly, edit the execs, search for the `noRun` configuration variable, and change its value to 0. The script is always generated for your reference.

IMPORTANT:

- When ORALTER directly updates UNIX security information, it uses UNIX REXX syscall command support (except for SECLABEL changes, which use the chlabel shell command because no syscall API exists), as opposed to executing the generated shell commands. Thus, if you truly want 'what you see is what you get', then leave `noRun=1` and execute the ORALTER.script file. Be aware, however, that when ORALTER uses the syscall support, it issues some reassuring confirmation messages when in verbose mode that you won't see when executing shell commands.
- In contrast, OPERMIT executes the generated script. I chose this approach for OPERMIT because I feel that it leaves a better audit trail than does the aclset() syscall command.

The shell commands generated were the choice of the author (me!). With shell commands, there is generally more than one way to skin a cat, and so the command syntax is not necessarily the only way the command could have been generated. For example, the permission bits used in the chmod command (in response to use of the PERMS keyword on ORALTER) are expressed in octal notation but could have been expressed in symbolic notation.   I also made no attempt to use a "-r" style recursive option in response to the RECURSIVE keyword. In fact, some of the commands lack such an option. Rather, individual shell commands are generated for each object affected. For

OPERMIT, I chose to generate individual setfacl commands for each ID value specified, and for each acl type specified, rather than attempting to combine them on a single command.

Do not abbreviate keywords, *unless otherwise noted* (for example, for the RECURSIVE keyword). A keyword that allows abbreviation is shown in the syntax diagram with the minimum allowable abbreviation in uppercase, and the remainder in lowercase. For example, the recursive keyword is shown as "RECursive", meaning "REC", "RECU", "RECUR", "RECURS", "RECURSI", "RECURSIV", and "RECURSIVE" are all valid ways to specify the keyword.

General fun facts:

- These work from the UNIX shell also! Note that the generated script will be *unix-file-name*.script where *unix-file-name* is whatever you've named it, with case preserved.
- ORALTER and OPERMIT have a RECURSIVE keyword! Be careful using it. When in 'run mode', it can generate a large number of commands, and a large number of ICH408I violations if you are not authorized for a given file or keyword. If you specify RECURSIVE with the DEBUG keyword, a list of the objects affected will be displayed. You can specify RECURSIVE and DEBUG in the absence of any other keywords in order to see the set of objects that would be affected.
- By default, ORALTER and OPERMIT run in 'verbose mode'. In addition to displaying the generated shell command(s), verbose mode will issue additional informational messages where appropriate for the path name you specify. If you crank the `verboseVal` configuration variable up to 11, it applies to all objects when RECURSIVE is specified. Beware, this can result in a lot of messages! If verbose mode gets annoying, edit the execs, search for the `verboseVal` configuration variable, and change its value to 0. You can optionally specify the VERBOSE keyword for individual commands, at your discretion. This will act as if `verboseVal=1`. There is no keyword equivalent of `verboseVal=11`.
- What if you are not authorized to make a given update? When specifying RECURSIVE, there is always the possibility that you are authorized to update some files but not others. I have chosen to implement OPERMIT and ORALTER such that they continue even when an authorization error is encountered. With ORALTER, even when not using RECURSIVE, specifying multiple keywords can update multiple security attributes, each of which have different authorization requirements. Again, I have chosen to continue attempting to process the keywords when an authorization error has occurred. Consider limiting the keywords specified on each

command instance until you are confident you have all the necessary authority.

- One of the frustrating 'features' of UNIX is that you can have authority to the target object, but lack authority to *reach* it due to lack of search (execute) permission on an intermediate directory higher up in the specified path name. If this is the case, the execs will tell you the first such directory to which you lack search permission.
- ORLIST takes a holistic approach when displaying security information. That is, regardless of what an object's security attributes are, they may not accurately reflect the effective access authority. For example, the permission bits may indicate write access, but the containing file system is mounted read-only. Or, the permission bits may indicate execute access, but the file system is covered by an FSACCESS class profile (for a directory), or an FSEXEC class profile (for a file) which restrict that access. ORLIST will display attributes of the containing file system. Optionally, ORLIST will use IRRXUTIL (which calls the R_admin callable service) to look for a covering FSACCESS or FSEXEC profile name, when applicable. R_admin requires READ access to IRR.RADMIN.RLIST in the FACILITY class, and the authority you would need to RLIST the profile. As such, IRRXUTIL is not called by default, so as not to result in possible FACILITY class violation messages. If you want this extra information, change the value of the `noIrrxutil` configuration variable to 0.
- If you want to run reports on *many* UNIX file system objects, consider using the IRRHFSU (Hierarchical File System Unload) download, which 'unloads' file system security data much as SMF Unload (IRRADU00) and Database Unload (IRRDBU00) do for RACF SMF records and the RACF database, respectively.

And now, on with the show.

Configuration variable reference

These variables can be found towards the top of each script. Search for the variable name itself, or for “Start of Configuration Variables”.

Variable name	**Description**	**Applies to**
noRun	1 = (default) Don’t make changes 0 = Make changes immediately	ORALTER OPERMIT
verboseVal	0 = Don’t display extra informational messages 1 = (default) Display extra messages for specified path 11 = Display extra messages for all objects when RECURSIVE is specified	ORALTER OPERMIT
scriptName	Name of script file to create. This can be a simple file name, in which case is created in your home directory. Or it can be any relative or absolute path name. By default, the name is *exec-name*.script.	ORALTER OPERMIT
noIrrxutil	1 = (default) Don’t invoke IRRXUTIL to determine if file system is covered by an FSACCESS or FSEXEC profile 0 = Perform extra checking	ORLIST

ORLIST

Purpose

Use the ORLIST command to display security information of a UNIX file or directory, including information about the file system data set in which it resides, and applicable RACF profiles that may cover the data set.

Details listed

- The containing data set's
  - name
  - file system type
  - mount point directory
  - mount mode (read/only, read/write, NOSECURITY, NOSETUID, etc)
  - the covering FSACCESS class profile name, if one exists. This is only displayed for non-root zFS file systems, if the `noIrrxutil` configuration variable is set to 0.
  - the covering FSEXEC profile name, if one exists. This is only displayed for executable files (one with any user, group, or other execute bit set ON), if the `noIrrxutil` configuration variable is set to 0.
- the file type (directory or regular file)
- owner (the UID is mapped to a user ID where possible)
- group owner (the GID is mapped to a group name where possible)
- the 'other' permission bits are displayed as the 'UNIVERSAL ACCESS'
- your access to the file or directory
- security label
- owner audit options
- auditor audit options
- creation date, last reference date, and last status change date
- For a regular file, the extended attributes, if present: APF, program-controlled, SHAREAS (_BPX_SHAREAS=YES and _BPX_SHAREAS=REUSE environment variable settings are ignored when the file is spawned), and the 'loaded from shared library'
- the value of the sticky, set-user-ID, and set-group-ID bits
- the file permission bits (owner, group, and other)
- the access list, if one exists
- for a directory:
  - the file model acl, if one exists
  - the directory model acl, if one exists

Authorization required

To list a file or directory, search access to each directory component up to and including the parent directory is required. This authority can be granted with either:

- execute (search) access to the directory via permission bits or an acl entry
- READ access to SUPERUSER.FILESYS.DIRSRCH or SUPERUSER.FILESYS in the UNIXPRIV class
- The RACF AUDITOR or ROAUDIT attribute

If search access is lacking, ORLIST will indicate the first directory component to which the issuer lacks access.

To display the covering FSACCESS and FSEXEC class profiles, the following authority is required:

- READ access to IRR.RADMIN.RLIST in the FACILITY class
- RACF command authority to list profiles in these classes (See the RLIST command in z/OS Security Server RACF Command Language Reference).

Syntax

Note: Simply invoking the exec with no parameters results in the command syntax and examples being displayed.

ORLIST (or whatever name you have chosen for it)
[FSSEC]
[absolute-path-name]
[AUTH]

FSSEC

Specifies the FSSEC class. This keyword is optional, and is supported so that ORLIST syntax can match the RACF RLIST command syntax as closely as possible.

absolute-path-name

Specifies the UNIX absolute path name of the file or directory you want to list. If the path is a symbolic link, the link is followed, and the target object is listed.

AUTH

Limits the output to only those fields required to determine POSIX access authority to the file or directory.

- Owner (user and group), universal access, and your access
- Permission bits
- Access ACL

Examples

To display security information about the file at /etc/inetd.conf:

```
ORLIST /etc/inetd.conf


FILE SYSTEM CONTAINER ATTRIBUTES
--------------------------------
NAME = ZOS24.ETC.ZFS                         TYPE = ZFS
MOUNT POINT = /SYSTEM/etc
Mount mode = READ/WRITE
Covered in FSACCESS class by ZOS24.ETC.*

FILE TYPE
----------------------
Regular file

OWNER        GROUP OWNER  UNIVERSAL ACCESS  YOUR ACCESS
----------   -----------  ----------------  -----------
IBMUSER      SYS1         r--               rw-

SECLABEL
--------
SYSMULTI

AUDITING
--------
FAILURES(READ),FAILURES(UPDATE),FAILURES(EXECUTE)

GLOBALAUDIT
-----------
NONE(READ),NONE(UPDATE),NONE(EXECUTE)

CREATION DATE  LAST REFERENCE DATE  LAST STATUS CHANGE DATE
-------------  -------------------  -----------------------
2019-09-20     2019-10-02           2019-09-20
```

```
EXTENDED ATTRIBUTES
-------------------
SHAREAS

FILE MODE BITS
--------------
Sticky  bit is: 0
Set-uid bit is: 0
Set-gid bit is: 0

FILE PERMISSIONS
----------------

  OWNER GROUP OTHER
  ----- ----- -----
  rw-   r--   r--           (644 in octal notation)

ID          TYPE   ACCESS
--          ----   ------
TSOUSR4     USER   R-X
SYS1        GROUP  R-X
```

To limit the above display to show only the access-related fields, use the AUTH keyword:

```
ORLIST /etc/inetd.conf AUTH

OWNER        GROUP OWNER  UNIVERSAL ACCESS  YOUR ACCESS
----------   -----------  ----------------  -----------
IBMUSER      SYS1         r--               rw-

FILE PERMISSIONS
----------------

  OWNER GROUP OTHER
  ----- ----- -----
  rw-   r--   r--           (644 in octal notation)

ID          TYPE   ACCESS
--          ----   ------
TSOUSR4     USER   R-X
SYS1        GROUP  R-X
```

ORALTER

Purpose

Use the ORALTER command to modify security information of a UNIX file or directory.

Authorization required

To modify an object's attributes, you must first be able to reach it. This requires search access to each directory component up to and including the parent directory. This authority can be granted with either:

- execute (search) access to the directory via permission bits or an acl entry
- READ access to SUPERUSER.FILESYS.DIRSRCH or SUPERUSER.FILESYS in the UNIXPRIV class
- The RACF AUDITOR or ROAUDIT attribute

If search access is lacking, ORALTER will indicate the first directory component to which the issuer lacks access.

Upon reaching the object, there may be additional authorization requirements depending on the attribute(s) being changed.

If you are the owner of the object, you can change the object's permission bits, access control list, and audit bits. You can also change the group owner to another group to which you are connected.

If you have a UID value of 0 or READ access to the appropriate UNIXPRIV profile (where relevant), you are said to have 'superuser privilege', and you can change many of the security attributes for any file or directory. However, there are exceptions.

The individual keyword descriptions document the authority required to change the attribute when ownership or UID(0) are insufficient. If a specific UNIXPRIV resource grants authority, it is shown.

Syntax

Note: Simply invoking the exec with no parameters results in the command syntax and examples being displayed.

ORALTER (or whatever name you have chosen for it)
[FSSEC]
[*absolute-path-name*]
[APF | NOAPF]
[AUDIT(access-attempt [(audit-access-level)])]
[DEBUG]
[GLOBALaudit(access-attempt[(audit-access-level)])]
[GROUP(*group-name*)]
[OWNER(*userId*)]
PERMS(*uuugggooo*)
[PROGRAM | NOPROGRAM]
[RECursive[(<u>CURRENT</u>|FILESYS|ALL)]]
[SECLABEL(*seclabel-name*)]
[SETGID | NOSETGID]
[SETUID | NOSETUID]
[STICKY | NOSTICKY]
[VERBOSE]

FSSEC

Specifies the FSSEC class. This keyword is optional, and is supported so that ORALTER syntax can match the RACF RALTER command syntax as closely as possible.

absolute-path-name

Specifies the UNIX absolute path name of the file or directory you want to change. If the path is a symbolic link, the link is followed, and the target object is updated.

APF | NOAPF

Either turn on or off the APF extended attribute, which indicates that the program is APF authorized or not.

These keywords require write access to the file being modified (via permission bit or acl entry – SUPERUSER.FILESYS authority is not sufficient) ***and*** READ access to

BPX.FILEATTR.APF in the FACILITY class. If you receive an authorization error, look on the security console for an ICH408I message that identifies the nature of the authorization failure.

APF

Turns on the APF extended attribute. APF is not allowed with RECURSIVE.

NOAPF

Turns off the APF extended attribute.

AUDIT(*access-attempt*[(*audit-access-level*)])

Specifies which access attempts and access levels you want logged to the SMF data set. This keyword sets the owner (or "user") audit bits for the file.

Unlike the RACF RALTER command, you can specify only one value for *access-attempt*(*audit-access-level)* per ORALTER command. For example, RALTER allows "`AUDIT(SUCCESS(READ) FAILURES(UPDATE))`". However, two ORALTER commands are necessary in this case: one specifying "`AUDIT(SUCCESS(R))`" and another specifying "`AUDIT(FAILURES(W))`"

You must have a UID value of 0 or be the file owner in order to change the owner audit bits.

access-attempt

Specifies which access attempts you want logged to the SMF data set. The following options are available:

ALL

Specifies that you want to log both authorized accesses and unauthorized attempts to access the resource.

FAILURES

Specifies that you want to log unauthorized attempts to access the resource.

NONE

Specifies that you do not want any logging to be done for accesses to the resource.

SUCCESS

Specifies that you want to log authorized accesses to the resource.

audit-access-level

Specifies which access levels you want logged to the SMF data set. The levels you can specify are:

R
Logs read access-level attempts.

W
Logs write access attempts.

X
Logs execute access attempts.

These values can be specified in any combination and in any order (E.G. “R”, “RW”, “X”, and “XRW” are all valid.

DEBUG

Specifies that various internal information is displayed to help understand and debug exec processing. This can be particularly useful if you want to see what objects are affected when the RECURSIVE option is used. In fact, you can specify RECURSIVE and DEBUG without specifying other keywords so that you can see the set of objects without making any changes to them.

GLOBALaudit(*access-attempt*[(*audit-access-level*)])

Specifies which access attempts and access levels you want logged to the SMF data set. “GLOBAL” is the shortest allowable abbreviation for this keyword. This keyword sets the AUDITOR audit bits for the file. The syntax is exactly same as for the AUDIT keyword, documented above.

You must have the RACF AUDITOR attribute to use this keyword.

GROUP(*group-name*)

Specifies a RACF-defined group to be assigned as the new group owner of the resource you are changing. The group must have an OMVS segment with a GID assigned. ORALTER maps the group name to a GID and stores that value as the file’s new group owner. In verbose mode (and not noRun mode), ORALTER issues a message notifying you of the mapped-to GID value.

Changing the group owner (even to its current value) also turns off the set-user-ID bit and set-group-ID bit of the named file or directory. If SETUID or SETGID is specified on the same command, the bit(s) are turned back on.

In the absence of superuser privilege, the file owner can change the group owner to any group to which the user is connected. Additionally, the file owner can change the group owner to *any* value if the user has READ access to CHOWN.UNRESTRICTED in the UNIXPRIV class.

The UNIXPRIV resource that allows any group change to a file for which you are not the owner is SUPERUSER.FILESYS.CHOWN.

OWNER(*userId*)

Specifies a RACF-defined user to be assigned as the new owner of the resource you are changing. *UserId* must have an OMVS segment with a UID assigned. ORALTER maps the user ID to a UID and stores that value as the file's new owner. In verbose mode (and not noRun mode), ORALTER issues a message notifying you of the mapped-to UID value.

Changing the owner (even to its current value) also turns off the set-user-ID bit and set-group-ID bit of the named file or directory. If SETUID or SETGID is specified on the same command, the bit(s) are turned back on.

The UNIXPRIV resource that allows an owner change is SUPERUSER.FILESYS.CHOWN.

In the absence of superuser privilege, a user can change the owner with access to CHOWN.UNRESTRICTED in the UNIXPRIV class:

- READ allows you to change the owner to any value other than 0.
- UPDATE allows you to change the owner to any value

PERMS(*perm-string*)

Specifies the permission bits (***u***ser, ***g***roup, and ***o***ther) to be assigned to the resource you are changing.

The UNIXPRIV resource that allows a permission change is SUPERUSER.FILESYS.CHANGEPERMS.

Perm-string can be specified in roughly the same formats that are allowed on the chmod shell command. These formats can be categorized into three different formats as follows:

Octal format

Permissions are expressed as a three-digit number, where each digit represents the read, write, and execute bits of each grouping (user, group, and other). A single group-ing is a base-8 number in which x is represented as 1, w is represented as two, and read is represented as 4. These individual values are added together to get the result.

For example, a combination of r-x is represented as the number 5, because the ones position (x) is on, the twos position (w) if off, and the fours position (r) is on. 1 + 0 + 4 = 5.

All three numbers, corresponding to the user, group, and other permissions must be specified. The file's existing permissions are replaced with this value.

For example, if you want the file owner to have all permissions, people connected to the file's owning group to have read and execute, but not write, and for nobody else to have any access, specify an octal value of 750.

Note that ORLIST displays permission bits in both octal and full symbolic format. This can help you get used to recognizing the relationship between the two notations.

Full symbolic

Permissions are expressed as nine characters (*uuugggooo*), three for each of the categories (user, group, other), all of which must be specified without imbedded spaces.

Each category consists of permissions in the form of "rwx", specifying a dash ("-") when that permission is not to be granted.

For example, to specify the same permissions as in the octal example above (750), specify `rwxr-x---.`

If you only want to change a subset of this information using this notation, then obtain the current settings (for example, by using ORLIST), and then modify only the positions you want changed.

For example, if the permissions are currently `rwxr-x---` and you wish to turn on the *other-read* bit, specify `rwxr-xr--.`

Alternatively, you can use the 'operator' format to change a subset of bits.

Operator format

Permissions are expressed by a set of categories (u, g, o, a), followed by an operation (=, +, -), followed by the bits you wish to change (r, w, x).

A category of "a" (all) is a shorthand for specifying "ugo" (which is also supported).

For example, you could accomplish the goal of the previous example (turning on the other read bit), by specifying just `o+r`. To remove write access to anybody except the file owner, specify `go-w`.

The categories can be specified in any order (e.g. ug, gou, and o are all valid). The same is true for the permissions bits (rwx, xrw, x, and xr, for example, are all valid).

The chmod command supports/tolerates some redundant forms (uug, rrw), some short-hand forms (+r), and some useless forms (u+) that will result in an error using ORALTER. Our goal is force precision in order to maximize understanding and reduce mistakes.

Note: to manage access control lists, use the OPERMIT command.

PROGRAM | NOPROGRAM

Either turn on or off the program-control extended attribute, which indicates that the program is defined to RACF program control or not.

These keywords require write access to the file being modified (via permission bit or acl entry – SUPERUSER.FILESYS authority is not sufficient) ***and*** READ access to BPX.FILEATTR.PROGCTL in the FACILITY class. If you receive an authorization error, look on the security console for an ICH408I message that identifies the nature of the authorization failure.

PROGRAM

Turns on the program-control attribute. PROGRAM is not allowed with RECURSIVE.

NOPROGRAM

Turns off the program-control attribute.

RECursive(ALL|<u>CURRENT</u>|FILESYS)

Applies the changes specified for the other keywords to the specified directory path name and sub-objects under it. “REC” is the shortest allowable abbreviation for this keyword. RECURSIVE is not allowed if the specified path name is a file. Symbolic and external links are excluded.

ALL

The keywords specified are applied to all sub-objects in the mounted file system structure.

CURRENT

The keywords specified are applied to all sub-objects in the specified directory only. This is the default if no sub-operand is specified.

FILESYS

The keywords specified are applied to all sub-objects within the same zFS file system data set. That is, mount points are not crossed.

SECLABEL(*seclabel-name*)

Specifies an installation-defined security label for this profile. A security label corresponds to a particular security level (such as CONFIDENTIAL) with a set of zero or more security categories (such as PAYROLL or PERSONNEL).

Setting the security label is only allowed if the user has RACF SPECIAL authority, and no security label currently exists on the resource. Once a security label is set, it cannot be changed.

SETGID | NOSETGID

Either turn on or off the set-group-ID bit.

If this bit is on for an executable file, when a user executes the file, the effective GID plus the saved GID for the process running the program is changed to the owning GID of the file. This change temporarily gives the process running the program access to data the owning group can access.

If the RACF profile named FILE.GROUPOWNER.SETGID exists in the UNIXPRIV class, then the set-group-ID bit for a directory determines how the group owner is initialized for new objects created within the directory:

- If the set-group-ID bit is on, then the owning GID is set to that of the directory.
- If the set-group-ID bit is off, then the owning GID is set to the effective GID of the process.

The UNIXPRIV resource that allows a set-group-ID bit change is SUPERUSER.FILESYS.CHANGEPERMS.

SETGID

Turns on the set-group-ID bit.

NOSETGID

Turns off the set-group-ID bit.

SETUID | NOSETUID

Either turn on or off the set-user-ID bit.

If this bit is on for an executable file, when a user executes the file, the effective UID plus the saved UID for the process running the program is changed to the owning UID of the file. This change temporarily gives the process running the program access to data the owning user can access.

The UNIXPRIV resource that allows a set-user-ID bit change is SUPERUSER.FILESYS.CHANGEPERMS.

SETUID

Turns on the set-user-ID bit.

NOSETUID

Turns off the set-user-ID bit.

STICKY | NOSTICKY

Either turn on or off the sticky bit.

For a file, the sticky bit causes a search for the program in the user's STEPLIB, the link pack area, or link list concatenation. For a directory, the sticky bit allows files in a directory or subdirectories to be deleted or renamed only by the owner of the file, by the owner of the directory, or by a superuser.

The UNIXPRIV resource that allows a sticky bit change is SUPERUSER.FILESYS.CHANGEPERMS.

STICKY

Turns on the sticky bit.

NOSTICKY

Turns off the sticky bit.

VERBOSE

Causes extra messages to be issued during exec processing. For example, messages confirming successful updates of each attribute will be displayed when not in noRun mode.

You can change the `verboseVal` variable value to 1 directly in the ORALTER exec so that you are in ‘verbose mode’ without having to specify the VERBOSE keyword every time. You can set the variable value to 11 to get the same messages for every object affected when the RECURSIVE option is specified (this option is not available using the keyword). Be careful! This can result in a large number of messages.

Examples

To change the file's owner, use the OWNER keyword:

```
ORALTER /u/bruce/myfile OWNER(JDAYKA)
```

To change the file's group owner, use the GROUP keyword:

```
ORALTER /u/bruce/myfile GROUP(RACFERS)
```

To change the file's owner, and preserve the set-user-ID bit:

```
ORALTER /u/bruce/myfile OWNER(JDAYKA) SETUID
```

To replace the file's permission bits, use the PERMS keyword:

```
ORALTER /u/bruce/myfile PERMS(rwxr-x---)
```

Or, using octal notation:

```
ORALTER /u/bruce/myfile PERMS(750)
```

To grant the file owner read and write access:

```
ORALTER /u/bruce/myfile PERMS(u+rw)
```

To remove write access from all but the file's owner:

```
ORALTER /u/bruce/myfile PERMS(og-w)
```

To turn on the file's APF and program-control bits, use the APF and PROGRAM keywords:

```
ORALTER /u/bruce/myfile APF PROGRAM
```

To turn off the file's APF and program-control bits, and other write access, for every file within a directory, use the RECURSIVE keyword:

```
ORALTER /u/bruce/myfile NOAPF NOPROGRAM PERMS(o-w) RECURSIVE
```

To turn on successful write access attempts to the file using the AUDITOR bits, use the GLOBALAUDIT keyword:

```
ORALTER /u/bruce/myfile GLOBALAUDIT(SUCCESS(w))
```

OPERMIT

Purpose

Use the OPERMIT command to manage access control lists (acls) for UNIX files and directories.

Authorization required

To modify or copy an object's acl, you must first be able to reach it. This requires search access to each directory component up to and including the parent directory. This authority can be granted with either:

- execute (search) access to the directory via permission bits or an acl entry
- READ access to SUPERUSER.FILESYS.DIRSRCH or SUPERUSER.FILESYS in the UNIXPRIV class
- The RACF AUDITOR or ROAUDIT attribute

If search access is lacking, OPERMIT will indicate the first directory component to which the issuer lacks access.

Upon reaching the object, one of the following authorities is required in order to create, modify, or delete an acl:

- File ownership
- READ access to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class
- UID 0

Syntax

Note: Simply invoking the exec with no parameters results in the command syntax and examples being displayed.

OPERMIT (or whatever name you have chosen for it)
[absolute-path-name-1]

[ACCess(*access-authority*) | DELETE]
[<u>ACL</u>] [FMODEL] [DMODEL] | [ALL]
[CLASS(FSSEC)]
[DEBUG]
[FROM(*absolute-path-name-2*)]
[FTYPE(ACL | DMODEL | FMODEL)]
[ID(*name* ...)]
[RECursive[(<u>CURRENT</u>|FILESYS|ALL)]]
[RESET]

absolute-path-name-1

Specifies the UNIX absolute path name of the file or directory you want to change. If the path is a symbolic link, the link is followed, and the target object is updated.

ACCess(*access-authority*) | DELETE

ACCess(*access-authority*)

Specifies the access authority you want to associate with the names that you identify on the ID operand. RACF sets the access authority in the type of access list(s) specified by the ACL, FMODEL, and DMODEL keywords, or the ALL key-word. If none of these are specified, ACL is defaulted.

Specify *access-authority* in the form of "rwx", specifying a dash ("-") when that permission is not to be granted. All three characters must be specified. If you only want to change a subset of this information, then obtain the current settings (for example, by using ORLIST), and then modify only the positions you want changed.

For example, if the permissions are currently `r-x`, and you wish to turn on the write bit, specify `rwx.`

DELETE

Specifies that you are removing the names you identify on the ID operand from an access list for the resource. RACF deletes the names from the type of access list(s) specified by the ACL, FMODEL, and DMODEL keywords, or the ALL key-word. If none of these are specified, ACL is defaulted.

[<u>ACL</u>] [FMODEL] [DMODEL] | [ALL]

Specifies the type(s) of acl you want to modify. ACL, DMODEL, and FMODEL can be specified in any combination, or ALL can be specified. That is, you can modify multiple acl types with a single command. If none of these options are specified, ACL is the default.

ACL

Specifies the access acl (yes, "access access control list" is redundant. Sorry!). This is the acl that actually determines access for a file or directory.

ALL

Specifies that the access acl, directory model acl, and file model acl are all to be modified. When ALL is specified, neither ACL, DMODEL, nor FMODEL can be specified. ALL can only be specified when *absolute-path-name-1* specifies a directory.

DMODEL

Specifies the directory model acl. This acl is applied as the access acl to new sub-directories created within the specified directory, and is copied as the directory model acl to these sub-directories. DMODEL can only be specified when *absolute-path-name-1* specifies a directory.

FMODEL

Specifies the file model acl. This acl is applied as the access acl to new files created within the specified directory, and is copied as the file model acl to new sub-directories created within the specified directory. FMODEL can only be specified when *absolute-path-name-1* specifies a directory.

CLASS(FSSEC)

Specifies a class name of FSSEC. This keyword and value are optional, and are supported so that OPERMIT syntax can match the RACF PERMIT command syntax as closely as possible.

DEBUG

Specifies that various internal information is displayed to help understand and debug exec processing. This can be particularly useful if you want to see what objects are affected when the RECURSIVE option is used. In fact, you can specify RECURSIVE and DEBUG without specifying other keywords so that you can see the set of objects without making any changes to them.

FROM(*absolute-path-name-2*)

Specifies the path name of the existing file or directory that contains the access list RACF is to merge into the access list(s) for *absolute-path-name-1*.

RACF modifies the access list for *absolute-path-name-1* as follows (which is how the PERMIT FROM command works for RACF profiles):

- Authorizations for *absolute-path-name-2* are added to the access list for *absolute-path-name-1*.
- If a group or user appears in both lists, RACF uses the authorization granted in *absolute-path-name-1*.
- If you specify a group or user on the ID operand and that group or user also appears in the *absolute-path-name-2* access list, RACF uses the authorization granted on the ID operand.

To specify FROM, you must have search access to *absolute-path-name-2*, as described above under “Authorization required”.

FTYPE(<u>ACL</u> | DMODEL | FMODEL)

Specifies the type of acl from which you wish to model, from *absolute-path-name-2* specified in the FROM keyword. FTYPE is ignored if FROM is not also specified.

ACL

Specifies the access acl is to be used as the source. If FTYPE is not specified, ACL is the default.

DMODEL

Specifies the directory model acl is to be used as the source. DMODEL can only be specified when *absolute-path-name-2*, specified on the FROM keyword, specifies a directory.

FMODEL

Specifies the file model acl is to be used as the source. FMODEL can only be specified when *absolute-path-name-2*, specified on the FROM keyword, specifies a directory.

ID(*name* ...)

Specifies the user IDs and group names of RACF-defined users or groups whose authority to access the resource you are granting, removing, or changing. If you omit this operand, OPERMIT ignores the ACCESS and DELETE operands.

A user value of *name* must have an OMVS segment with a UID assigned. A group value of *name* must have an OMVS segment with a GID assigned. OPERMIT maps a user ID to a UID and a group name to a GID and stores that value in the acl entry. When DEBUG is specified, OPERMIT displays the mapped-to ID value(s).

RECursive(ALL|<u>CURRENT</u>|FILESYS)

Applies the changes specified for the other keywords to the specified directory path name and sub-objects under it. "REC" is the shortest allowable abbreviation for this keyword. RECURSIVE is not allowed if the specified path name is a file. Symbolic and external links are excluded.

ALL

The keywords specified are applied to all sub-objects in the mounted file system structure.

CURRENT

The keywords specified are applied to all sub-objects in the specified directory only. This is the default if no sub-operand is specified.

FILESYS

The keywords specified are applied to all sub-objects within the same zFS file system data set. That is, mount points are not crossed.

RESET

Specifies that RACF is to delete the entire access list(s) from the object at the specified path name.

OPERMIT deletes the access list before it processes any operands (ID and ACCESS or FROM) that create new entries in an access list.

Examples

To permit the user ELLIE and the group SECADMNS to a file with only read permission, use the ID, ACCESS, and ACL keywords to modify the access acl:

```
OPERMIT /u/emily/myfile ID(ELLIE SECADMNS) ACCESS(r--) ACL
```

Since ACL is the default, it may be omitted:

```
OPERMIT /u/emily/myfile ID(ELLIE SECADMNS) ACCESS(r--)
```

To remove the acl associated with a file, use the RESET keyword:

```
OPERMIT /u/emily/myfile RESET
```

To permit the user ELLIE and the group SECADMNS to a file with only read permission, and ensure that these are the only access list entries in effect:

```
OPERMIT /u/emily/myfile RESET ID(ELLIE SECADMNS) ACCESS(r--)
```

To merge the acl from one object into the acl of another object, use the FROM keyword:

```
OPERMIT /u/emily/myfile FROM(/u/cameron)
```

To merge the directory model acl from one object into the file model acl of another object, use the FTYPE keyword:

```
OPERMIT /u/emily/myfile FMODEL FROM(/u/cameron) FTYPE(DMODEL)
```

To completely replace the acl of one object with the acl from another object:

```
OPERMIT /u/emily/myfile RESET FROM(/u/cameron)
```

To completely replace the acl of one object with the acl from another object, while adding a new acl entry:

```
OPERMIT /u/emily/myfile RESET FROM(/u/cameron) ID(PATRICK) ACCESS(r-x)
```

To change a directory’s access acl and apply the same change to sub-objects within the same directory, use the RECURSIVE keyword with the CURRENT sub-operand:

```
OPERMIT /u/emily ID(ELLIE SECADMNS) ACCESS(r--) RECURSIVE(CURRENT)
```

Since CURRENT is the default, it may be omitted:

```
OPERMIT /u/emily ID(ELLIE SECADMNS) ACCESS(r--) RECURSIVE
```

To make the same change apply to all sub-objects within the same file system data set, use the FILESYS sub-operand:

```
OPERMIT /u/emily ID(ELLIE SECADMNS) ACCESS(r--) RECURSIVE(FILESYS)
```

To make the same change apply to all sub-objects within the entire mounted file system hierarchy, use the ALL sub-operand:

```
OPERMIT /u/emily ID(ELLIE SECADMNS) ACCESS(r--) RECURSIVE(ALL)
```

To change a directory’s file model acl, use the FMODEL keyword:

```
OPERMIT /u/emily/myfile ID(ELLIE SECADMNS) ACCESS(r--) FMODEL
```

To change a directory’s directory model acl, use the DMODEL keyword:

```
OPERMIT /u/emily/myfile ID(ELLIE SECADMNS) ACCESS(r--) DMODEL
```

To change a directory’s access acl, directory model acl, and file model acl at the same time:

```
OPERMIT /u/emily/myfile ID(ELLIE SECADMNS) ACCESS(r--) ALL
```

To apply a file model acl change to the file model acl of the specified directory and subdirectories within the specified directory:

```
OPERMIT /u/emily ID(ELLIE SECADMNS) ACCESS(r--) FMODEL RECURSIVE
```

To replace a directory’s file model acl with that of another and apply it to sub-directories with the same file system data set:

```
OPERMIT /u/emily RESET FMODEL FROM(/u/cameron) RECURSIVE(FILESYS)
```

To update a directory’s access acl and synchronize it with its file and directory model acls and all sub-objects, use two (or more) commands. First, get the directory’s acl as you want it. Then, copy it to itself (and its model acls if you are using them), use the RESET keyword to clear any existing acl entries, and use the RECURSIVE keyword to apply the new acl to sub-objects in the same file system data set:

```
OPERMIT /u/emily ID(RICHARD VIRGINIA) ACCESS(rwx)
OPERMIT /u/emily FROM(/u/emily) RESET ALL RECURSIVE(FILESYS)
```

This is a good use case for ‘named acls’. That is, create a file somewhere whose only purpose is to contain an acl, for the purpose of applying that acl elsewhere in the file system. You could establish a directory to contain such named acls. The idea is to maintain the model over time, and as changes are made, re-apply that acl where desired. For example:

```
OPERMIT /etc/acls/ProjectX ID(GROUP1 GROUP2 …) ACCESS(rw-)
OPERMIT /etc/acls/ProjectX ID(GROUPA GROUPB …) ACCESS(r--)
OPERMIT /bin/ProjectX/source FROM(/etc/acls/ProjectX)
         ALL RECURSIVE(FILESYS)
```

Disclaimers, etc

Dedication to John C. Dayka

This download is dedicated in loving memory to my big brother in mainframe security, John Dayka, on the first anniversary of his passing.

For my first three years at IBM, fresh out of college, I had been working in VM development in Kingston, NY. That mission was then being moved to Endicott, NY, and we had the option of moving with it, or joining a different group. There was a job fair. John was there representing RACF/VM. I was interested because my final VM project was security-related. The senior people on that project had worked with the RACF people. John was spoken of highly by them, and I was flattered that he took the time to recruit newbies like myself.

John explained that RACF/VM was a unique and interesting product. As a modified port of the RACF/MVS product, it embodied general security concepts, implemented by a combination of VM and MVS programming models. John's enthusiasm convinced me to join the team, and I've been involved in security, and RACF in particular, ever since.

John was the technical team leader. It became quickly apparent that he was the undisputed, indispensable expert. He was always the smartest person in the room. Despite his pressures and workload, he took the time to educate me, and others, on the nuances of the product. John taught me a whole bunch of MVS concepts, some of which I finally started to understand within the last two years, I think.

At some point, I got promoted, and was horrified to discover that I was at the same level as John. To IBM's credit, they quickly rectified that situation. Over the ensuing 29 years, as John advanced through the ranks, he pulled me along in his wake.

On his journey, he would often invite me to participate in career development opportunities such as technical discussions with senior level experts, client meetings, business travel, and a couple of patents, none of which it had even occurred to me to ask for. For all those years, I had the additional pleasure of working in close *physical* proximity to John; across multiple sites and buildings, separated only by thin walls, some of which lacked ceilings and doors. This afforded me the luxury of profiting from his wit, wisdom, creativity, and tact, even when I was not invited.

When I refer to John as a big brother, I didn't mean the kind of big brother who would chase you down and give you wedgies, but the big brother who would protect you from exactly that, metaphorically speaking. When John was in the room, the pressure was off. If you had the seed of a good idea, John encouraged you to put yourself out there and show the world. If the audience wasn't receiving it well, John was there to absorb the slings and arrows. “Now hold on a minute …”. John would jump to your defense, and the respect he commanded would compel people to step back and reconsider their position. He had a natural affinity to deflect any criticism into a productive discussion of what outcome the audience would *like* to see. I could only sit in awe, not taking any offense at all, but rather admiring John’s ability to take control of a situation and hoping I could someday do that myself with half as much grace. While I would like to think that I had a unique relationship with John in this way, I know that there are many others who feel the same way as I do.

One of our last collaborations was brain-storming with clients to find ways of addressing separation-of-duties issues regarding UNIX security management on the platform. It was instructional, and inspirational, to watch John try to solve problems in a creative manner, even if done outside the traditional development process. This download is a direct result of those discussions, and of John’s relentless, infectious energy, enthusiasm, and encouragement.

I wish John were around so I could show him this download. I like to image it going something like this:

I pop into his office to demonstrate an early prototype. He drops whatever he’s doing and gives me his full attention. John says: “Bruce, that’s fanTAStic!”, whether it was or not. We proceed to discuss various aspects of UNIX security, and John casually mentions Client Pain Point X. As we continue our discussion, his phone rings, as it inevitably would. “We’ll put a bookmark here”, he says.

I wander back to my office and tinker around some more while eavesdropping on his conversation through our shared wall. After some amount of time, I bring it back to him and show how I’ve addressed Client Pain Point X. John says: “Bruce, youda man! That’s brilliant!”, as if it was my idea. His encouragement makes me want to improve it even more.

I hope that John would be pleased with this download, and I hope that you find some value in it as well.