IBM eServer™

Session R05
RACF and z/OS UNIX System Services

Networking and Security Technical Conference
October 25-28, 2004
Anaheim, California

Peggy LaBelle
IBM Corporation, RACF Development
(845) 435-5910
plabelle@us.ibm.com

Disclaimer

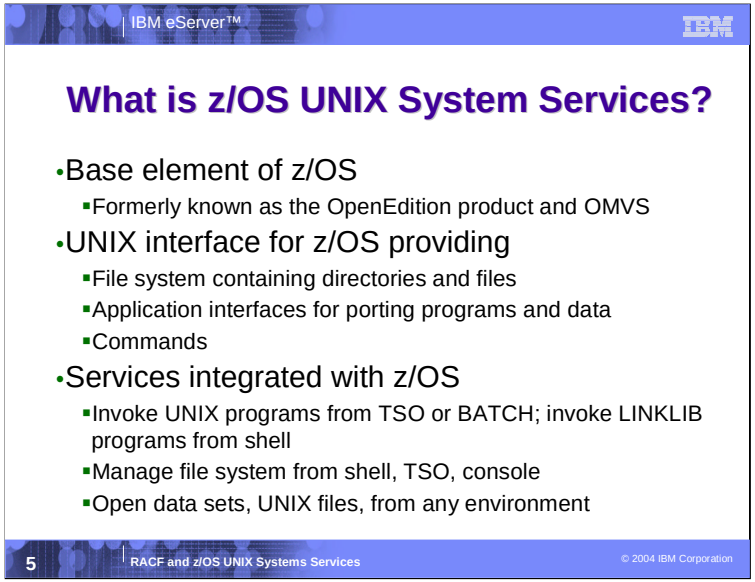
- The information contained in this document is distributed on as "as is" basis, without any warranty either express or implied. The customer is responsible for use of this information and/or implementation of any techniques mentioned. IBM has reviewed the information for accuracy, but there is no guarantee that a customer using the information or techniques will obtain the same or similar results in its own operational environment.
- In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used. Functionally equivalent programs that do not infringe IBM's intellectual property rights may be used instead. Any performance data contained in this document was determined in a controlled environment and therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.
- It is possible that this material may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM Products, programming or services in your country.
- IBM retains the title to the copyright in this paper as well as title to the copyright in all underlying works. IBM retains the right to make derivative works and to republish and distribute this paper to whomever it chooses.

Trademarks

- **The following are trademarks or registered trademarks of the International Business Machines Corporation:**
  - OS/390
  - z/OS
  - RACF
- **UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.**

Agenda

- z/OS UNIX introduction
- z/OS UNIX users
  - UNIX vs. z/OS identity
  - Defining a UNIX user and group
  - And more…
- z/OS UNIX file systems and MVS datasets
  - File ownership, file permissions, ACLs
  - Executable programs
  - And more…
- Auditing z/OS UNIX



Makes application development easier:

•Standard (open) programming interface

•Interoperability in networks

•Portable programs

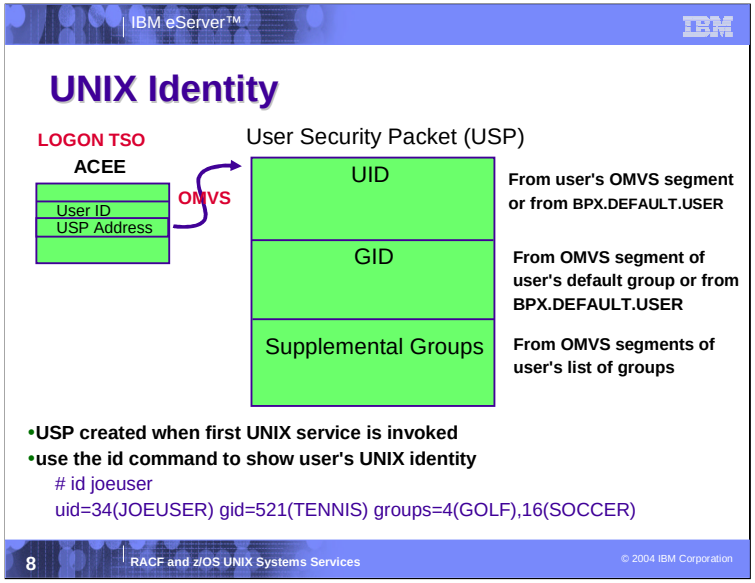
•Portable data

Required by products such as TCP/IP, Lotus Domino, LDAP, EIM, PKI

How is it related to RACF?

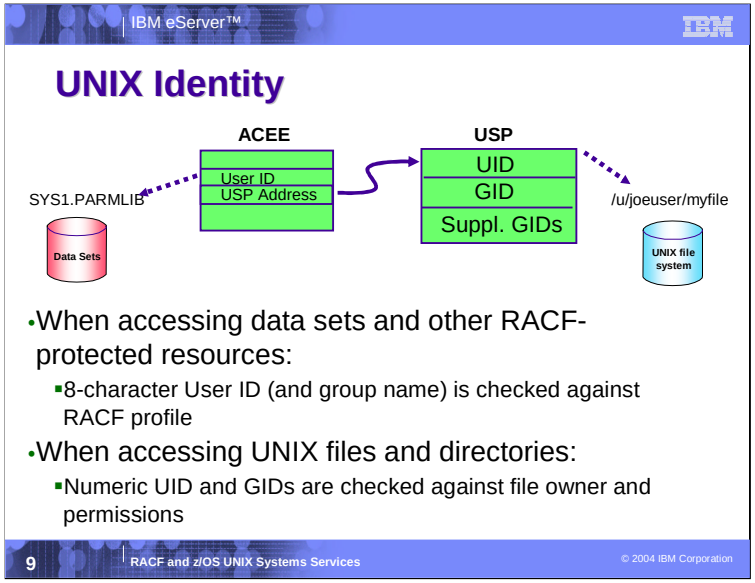
- External security product is required
- User identification and authentication
- Protection of files
- Protection of services (su, chmod, chown, etc)
- Auditing of security events

z/OS Unix System Services Identities



When a UNIX service is invoked, the ACEE (z/OS security context) is supplemented with a USP containing the user's UNIX identity

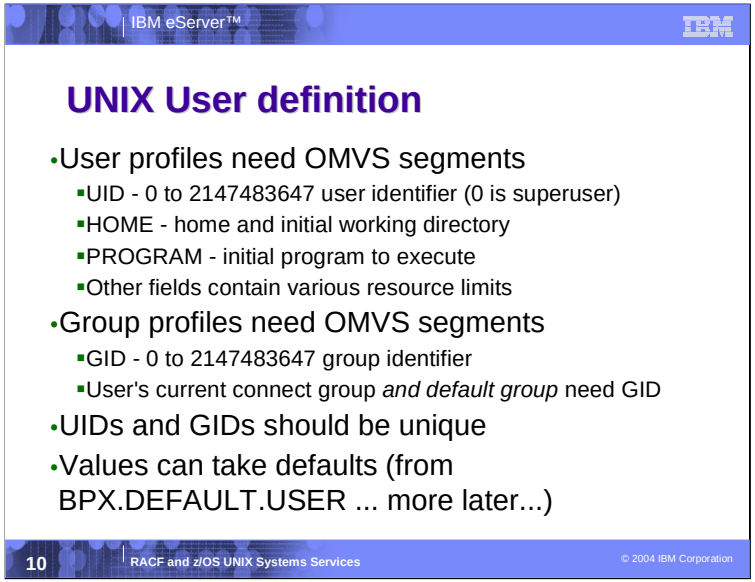
UNIX identity consists of a numeric UID taken from the OMVS segment of the USER profile, a numeric GID taken from the OMVS segment of the GROUP profile associated with the user's current connect group (usually the user's default group, unless (s)he specified a different group at logon), and a list of GIDs taken from each group (with an OMVS segment) to which the user is connected, up to 300.



When a data set is accessed, the user ID from the ACEE is used for authority checking using RACF profiles.

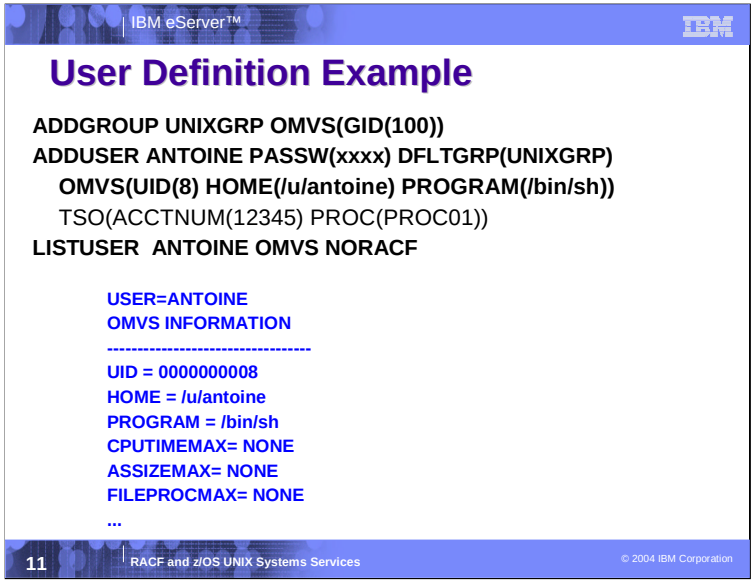
When a UNIX file is accessed, the effective UID, effective GID, and supplemental GIDs are used for authority checking using the security information which is stored with the file in the file system.

BUT when UNIXPRIV is checked, the user ID is used!



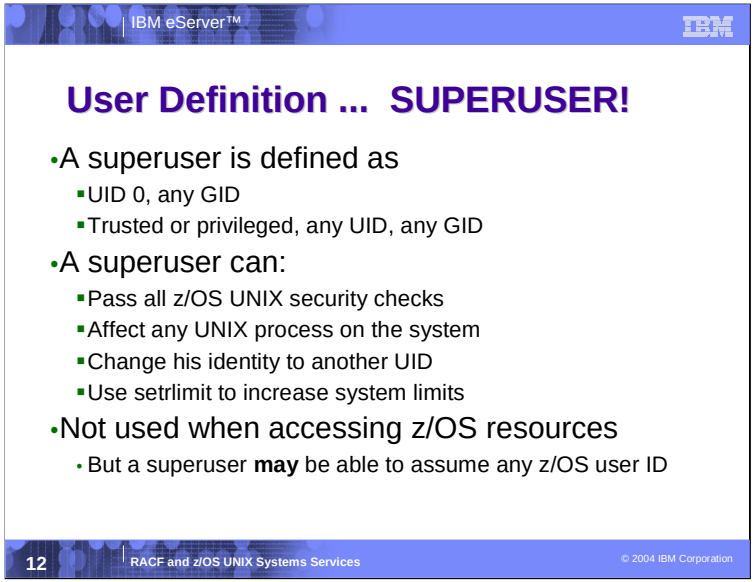
UNIX limits in OMVS segment are: max address space size, cpu time, files per process, memory map size, and threads per process..they are discussed shortly...

Also, default GID can be taken from BPX.DEFAULT.USER
(In the absence of BPX.DEFAULT.USER) current connect group MUST have a GID. If default group does not have a GID, you get an error message when adding the UID to the user, and when entering the shell. **You *can* enter the shell, but (at least) getpwnam() won't work, and so 'ls -l' won't be able to map to names**.



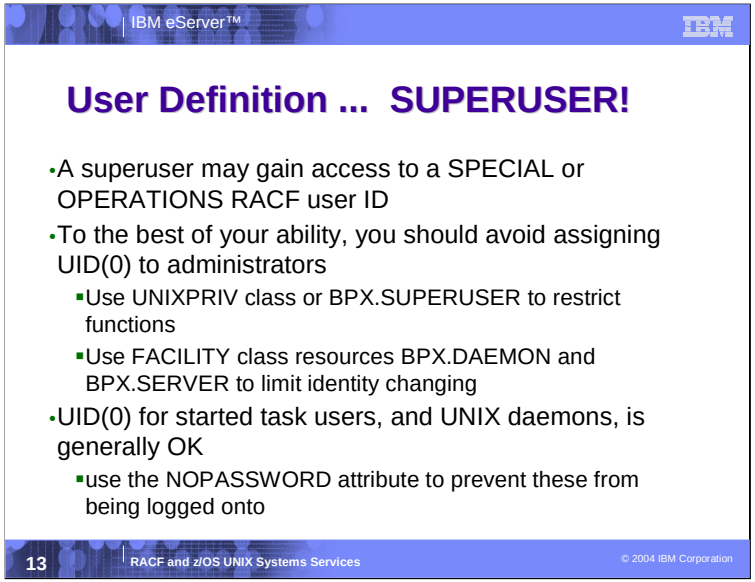
The limit fields are discussed shortly..

There is no reason to prevent general users from updating their own HOME and PROGRAM fields;use the FIELD class. But by all means, do not let them update their own UID!!! Letting them update their own limits would also not be a very good idea.

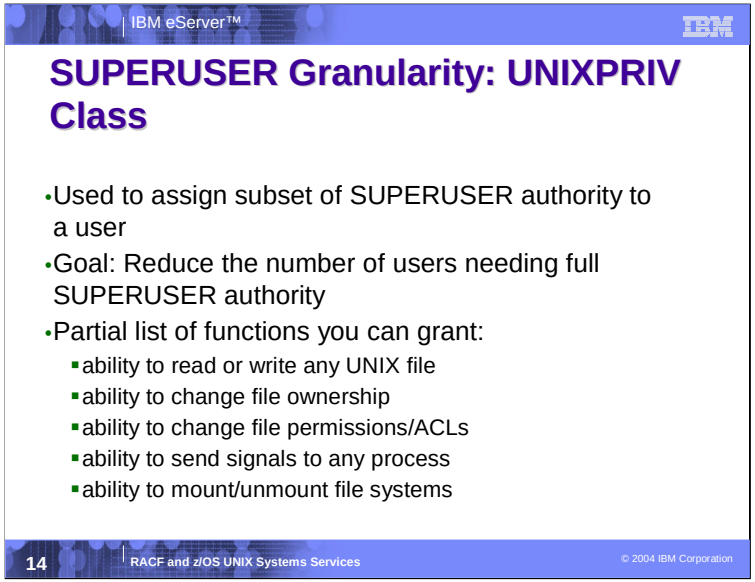


A superuser is a user running with UID 0. A started task that is trusted or privileged is also considered a superuser.

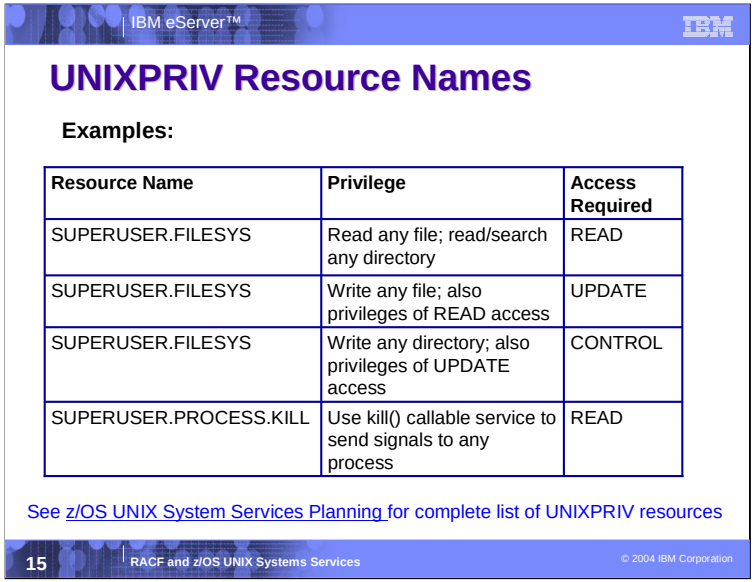
Changing to another UID can be further controlled using BPX.SERVER and BPX.DAEMON. But if you're not using these profiles, a superuser can switch to any identity
Note that this **can** (indirectly) allow a superuser to access z/OS resources with another user ID



A superuser is essentially SPECIAL and OPERATIONS if you are not protecting servers and daemons (discussed later). In this scenario, the superuser can simply write a program which does a setuid() to any user ID in the system, thus allowing them to access that user's UNIX files and z/OS datasets. By setuid()ing to an OPERATIONS user, one could conceivably access any data set. By setuid()ing to a SPECIAL user, one could conceivably excute any RACF command using the R_admin interface.

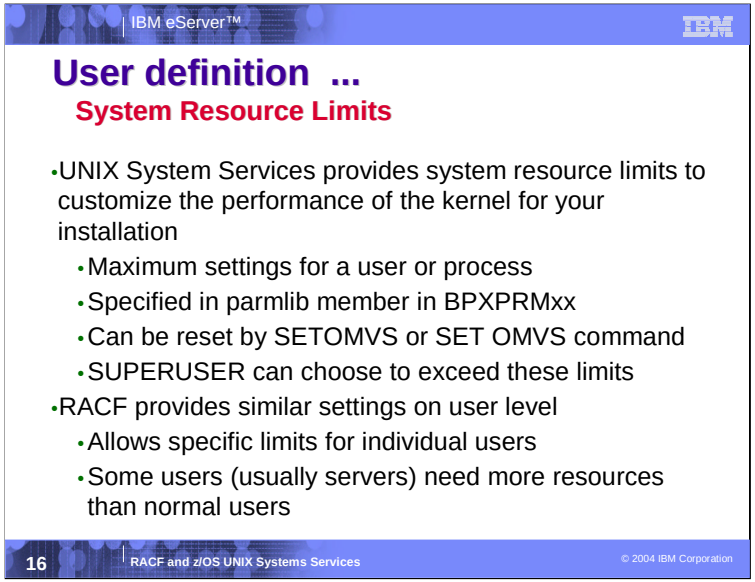


Contrast this with authority to BPX.SUPERUSER. With BPX.SUPERUSER, you issue the su command to get into "superuser mode", do your task, and exit. With UNIXPRIV, you always have that authority, so be careful not to make mistakes!



File execution (as opposed to directory searching) is not covered by UNIXPRIV

Other UNIXPRIV resource names
SUPERUSER.FILESYS.CHOWN - READ access allows user to use the chown command to change ownership of any file.
SUPERUSER.FILESYS.MOUNT - READ allows user to mount file system with nosetuid option. UPDATE allows user to mount file system with setuid option.
SUPERUSER.FILESYS.QUIESCE - READ/UPDATE allows user to issue quiesce/unquiesce commands for a file system mounted with nosetuid/setuid.
SUPERUSER.FILESYS.PFSCTL - READ access allows user to use the pfsctl() callable service.
SUPERUSER.FILESYS.VREGISTER - READ allows a server to use the vreg() callable service to register as a VFS file sesrver.
SUPERUSER.SETPRIORITY - READ access allows a user to increase own priority



Choices before OS/390 V2R8:

increase system limit for all users

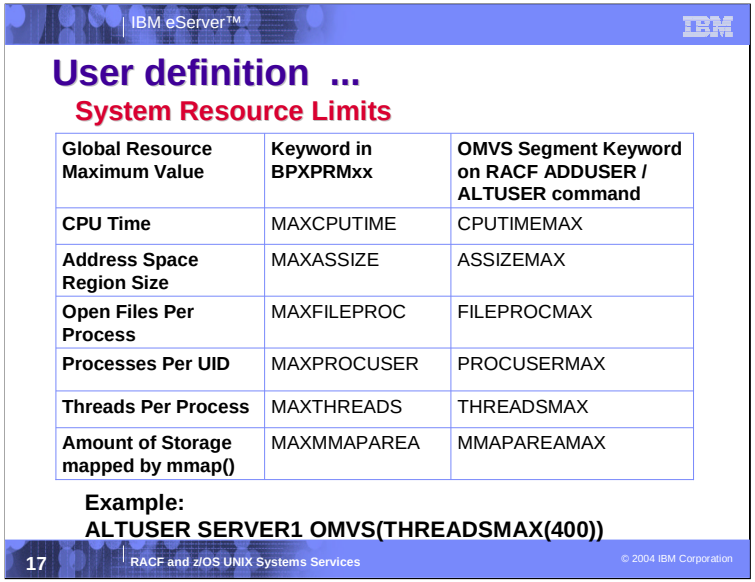
bad for system reliability, performance

give server UID(0) or BPX.SUPERUSER authority and modify server code to request a higher limit

requires modification to server code

With OS/390 V2R8 and later:

assign higher limits specifically to server user IDs (or others) that need them



As usual, TSO command abbreviations are allowed, so you don't necessarily need to type all of these fully.
MAXPROCUSER - may impact the number of users attempting to use kernel services through use of the default UID established in the BPX.DEFAULT.USER profile.

OMVS segment keywords on ADDUSER and ALTUSER allow specific limits for individual users:

CPUTIMEMAX(cpu-time)

ASSIZEMAX(address-space-size)

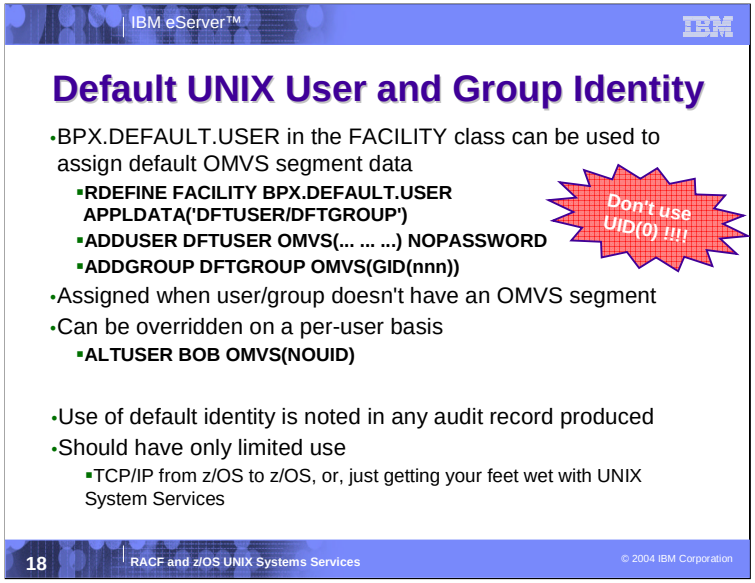
FILEPROCMAX(files-per-process)

PROCUSERMAX(processes-per-UID)

THREADSMAX(threads-per-process)

MMAPAREAMAX(memory-map-size)

Listed via LISTUSER



Consider making default user NOPASSWORD so nobody can try to log on to it

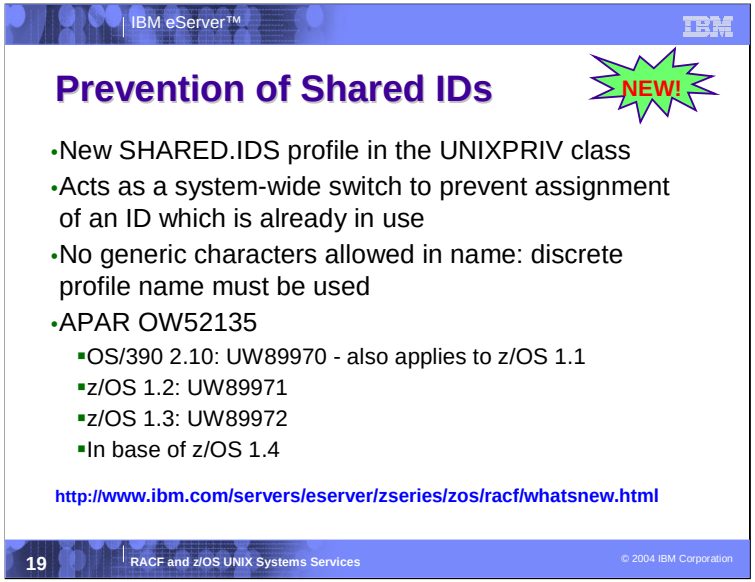
All OMVS fields are eligible for use

UID - choose a number that jumps out visually (not 0!!!)

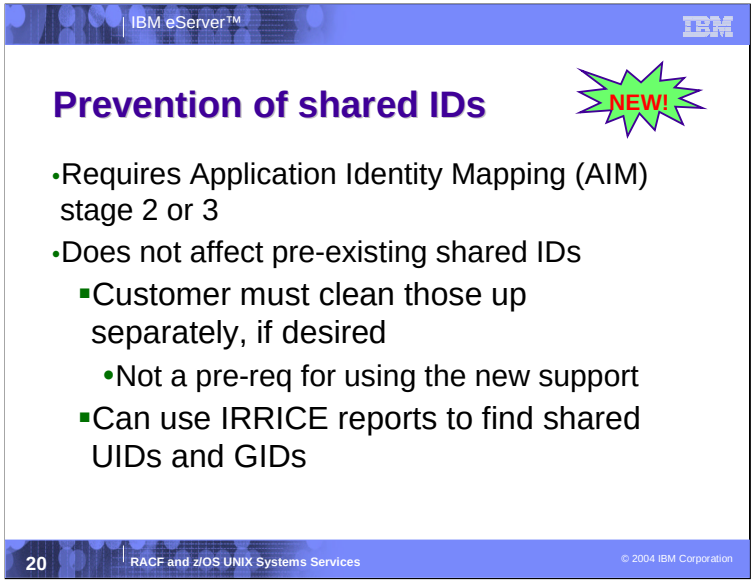
HOME - all default users will have access to each other's files. Consider putting it in /tmp. Also consider making it root, where they will not have write access

PROGRAM - usually /bin/sh. If you don't want default users to be able to use the shell, set it to /bin/echo

resource limits like CPUTIMEMAX, PROCUSERMAX, and THREADSMAX are shared among all users using the default segment, so you might want to make the default limits higher than you otherwise would



The APAR does not include ISPF panel support. It is included in z/OS V1.4.



IRRIRA00 utility – RACF Internal Reorganization of Aliases Utility Program (OS/390 V2R10)

In final stage, stage 3, provides an alternative to the use of mapping profiles to associate RACF user and group names with

z/OS UNIX ids

LOTUS Notes ids

Novell Directory Services ids.

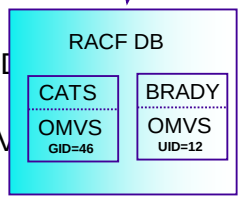
The IRRIRA00 utility converts from the use of UNIXMAP profiles to the use of alternate database indices. It is a fairly clean and straightforward process, but some customers have complained about:

1) the time it takes to move to stage 1 with a large RACF database. IRRIRA00 doesn't give you any 'progress reports'

2) the fact that AIM chokes when there are somewhere in the vicinity of 130 UID(0) users. If you have 130 super users, you should really think long and hard about that. That's 130 users with essentially SPECIAL and OPERATIONS!!! Use BPX.SUPERUSER, UNIXPRIV, and set up server and daemon level protection (more later).

Prevention of Shared IDs ... Example

NEW!

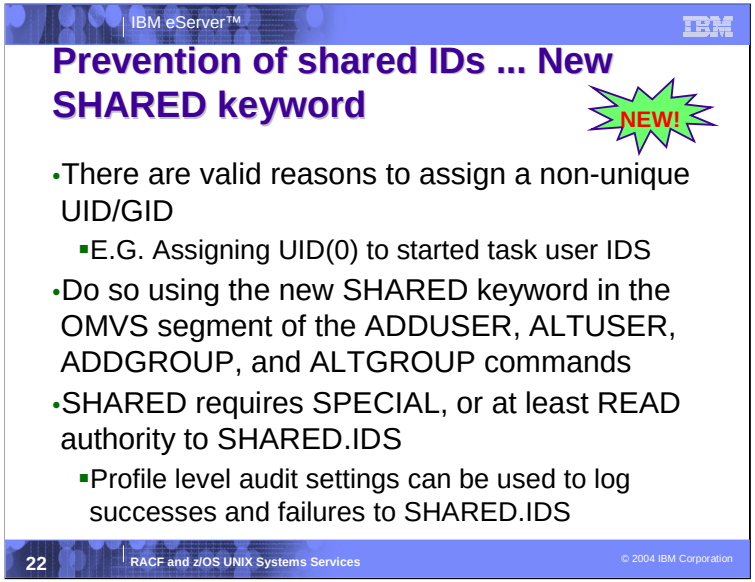


- RDEFINE UNIXPRIV SHARED.ID UACC(NONE)
- SETROPTS RACLIST(UNIXPRIV REFRESH
- ADDUSER MARCY OMVS(UID(12))

IRR52174I Incorrect UID 12. This value is already in use by BRADY.

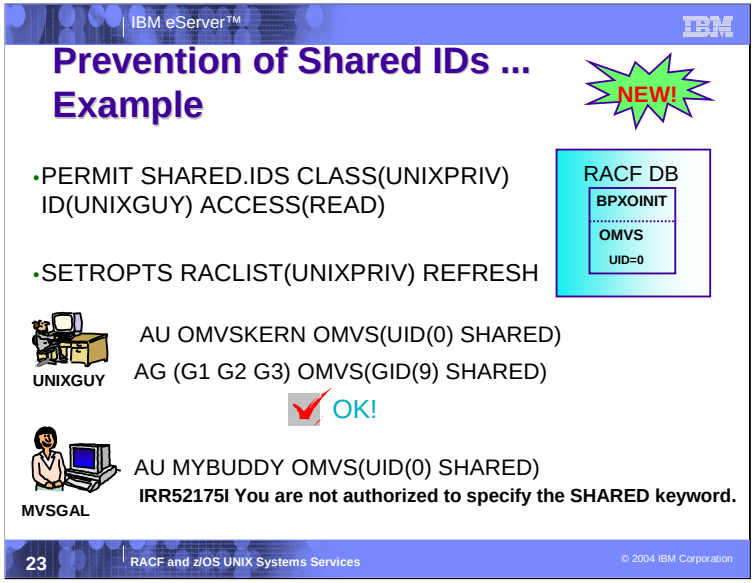
- ADDGROUP DOGS OMVS(GID(46))

IRR52174I Incorrect GID 46. This value is already in use by CATS.



Can generally only log successes in the UNIXPRIV class. We make an exception for SHARED.IDS because it is the only way to log the failure...you won't get a command related (e.g. ADDUSER, ALTGROUP) audit record.

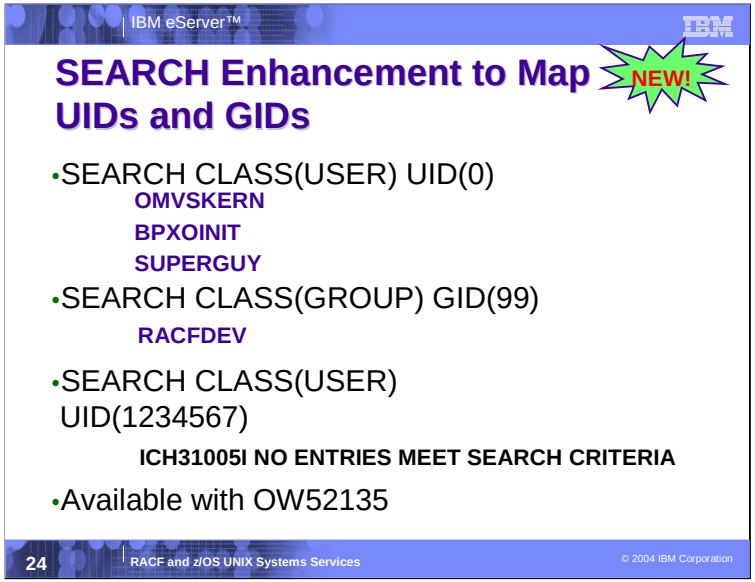
No FIELD class checking for SHARED operand because there's no corresponding field in the RACF database! This is a change from existing fields within non-base segments which always correspond to a template field in the RACF database. Processing for the SHARED keyword is just pertinent for the life of the command, and is not an ‘attribute’ which needs to be stored in the USER profile.



Even if MVSGAL would otherwise have been authorized to define the user (e.g. via CLAUTH(USER), FIELD access, group authority, etc), she may not assign a shared UID or GID unless she is SPECIAL, or has access to SHARED.IDS.

By the way, I feel compelled to say that we recommend you permit groups to RACF profiles instead of users, but it's so much simpler to show examples using users!!!

The SHARED keyword will be ignored when: SHARED.IDS is not defined, UID/GID is not also specified, the ID value specified is unique, and for ALTUSER/ALTGROUP when the specified ID is the value already assigned (regardless of whether it is unique or not).



No authority to the user/group profile is required in order to list it using the new keyword. This is consistent with UNIX mapping interfaces.

The new keywords require AIM. If you don't have AIM, then continue to RLIST UNIXMAP profiles.

All other SEARCH keywords are ignored when you use UID or GID (except of course the CLASS keyword, which must specify USER or GROUP as appropriate. Any other class will cause the UID/GID keywords to be ignored)

No more excuses! You will start to see more new function which requires AIM.

Automatic UID/GID Assignment

NEW!

- New AUTOUID keyword in the OMVS segment of the ADDUSER and ALTUSER commands
- New AUTOGID keyword in the OMVS segment of the ADDGROUP and ALTGROUP commands
- Derived values are guaranteed to be unique
- Must use in conjunction with SHARED.IDS

ADDUSER MELVILLE OMVS(AUTOUID)

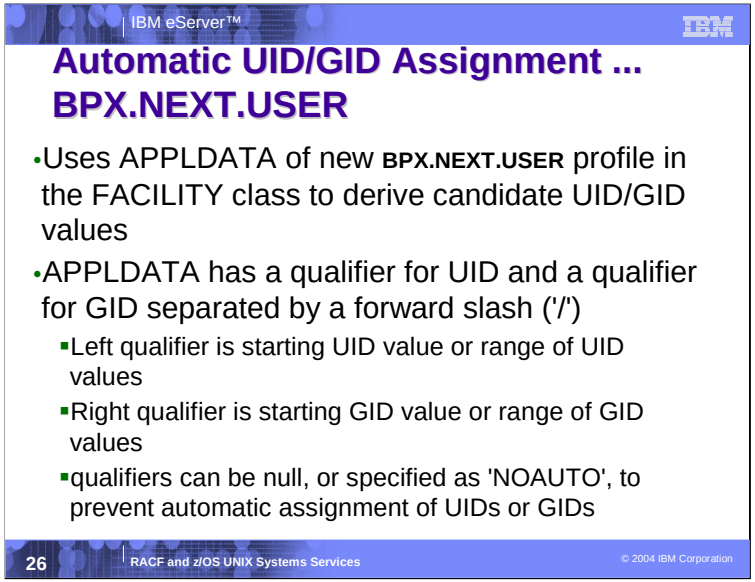
IRR52177I User MELVILLE was assigned an OMVS UID value of 4646.

ADDGROUP WHALES OMVS(AUTOGID)

IRR52177I Group WHALES was assigned an OMVS GID value of 105.

Anyone authorized to assign a UID/GID, can use AUTOUID/AUTOGID

As with the SHARED keyword, there is no RACF database template field for AUTOUID or AUTOGID, and thus no FIELD class protection; except of course for the UID and GID fields themselves.



APPLDATA verified at time of use, not when defined

FACILITY class need not be active or RACLISTed

When AUTOUID or AUTOGID is issued, RACF

•extracts the APPLDATA from BPX.NEXT.USER

•parses out the starting value

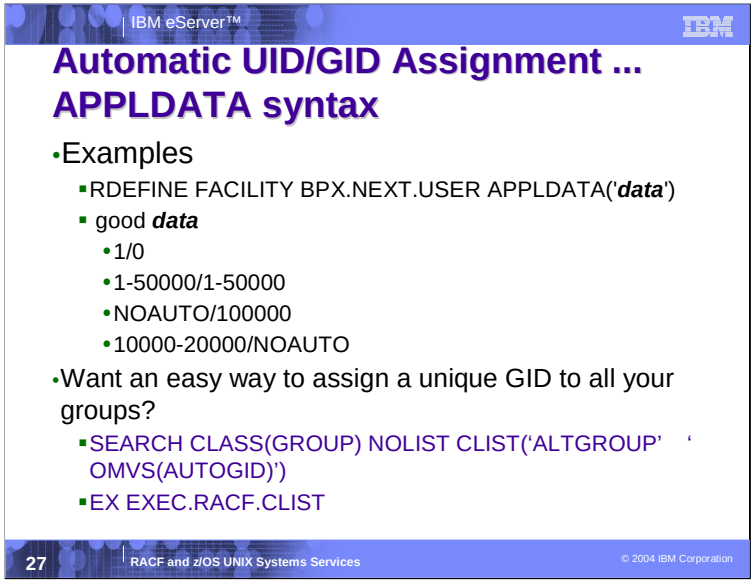
•checks to see if it is already in use

•If so, the value is incremented and checked again until an unused value is found

•assigns the value to the user or group

•replaces the APPLDATA with the new starting value

The administrator can change the APPLDATA at any time using



A qualifier can be specified as 'NOAUTO', or simply omitted, to prevent automatic ID assignment.

Might want this if you have a convention for users, such as employee serial number

Values must be valid UIDs or GIDs. If specifiying a range, end of range must be greater than start of range. No white space allowed.

Validity is checked at time of use, not at time of definition (So, RDEFINE/RALTER won't fail, but subsequent AU, ALU, AG, ALG will). Give it a quick test to be sure.

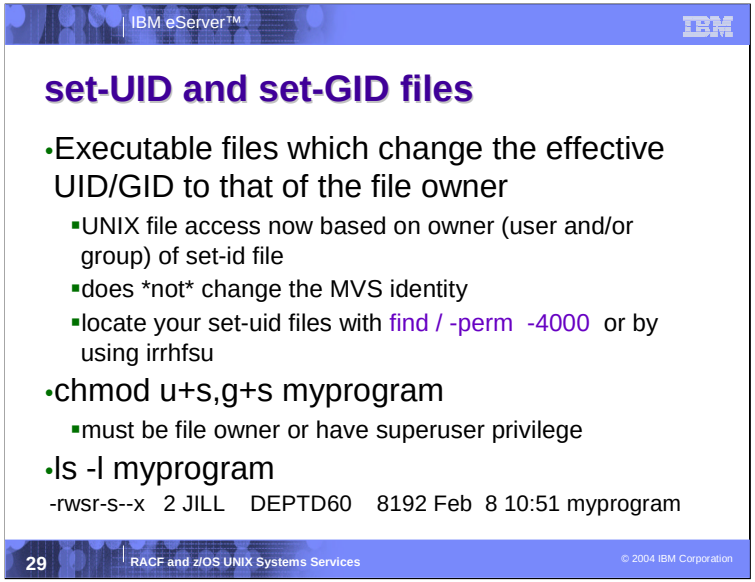
No more complaining about having to assign GIDs to all your groups!

Could also be used for users, but, there are other things in the OMVS segment of USER profiles which will need addressing (e.g. HOME, PROGRAM). ISHELL can do this for you.

AUTOUID/AUTOGID will not reassign a new value if one already exists

Ways of Assuming Another UNIX Identity

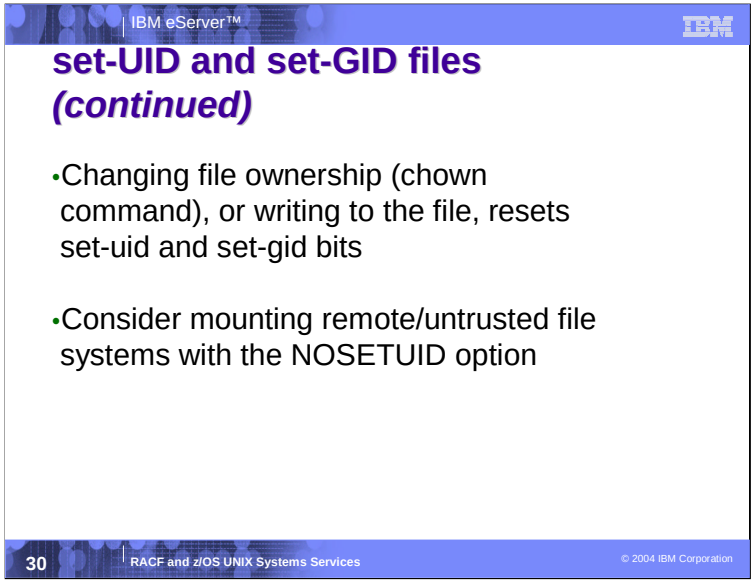
- Executing a set-id file
  - changes effective UID/GID to that of file owner
- Issuing the su command
  - used to switch to 'superuser mode' or to another user entirely
- Various C language functions such as setuid(), setgid(), pthread_security_np()
  - Used by UNIX servers and daemons



When a set-UID or set-GID file executes, the effective UID/GID is changed to that of the file owner.

You can locate set-uid files with 'find / -perm -4000' or by using irrhfsu.

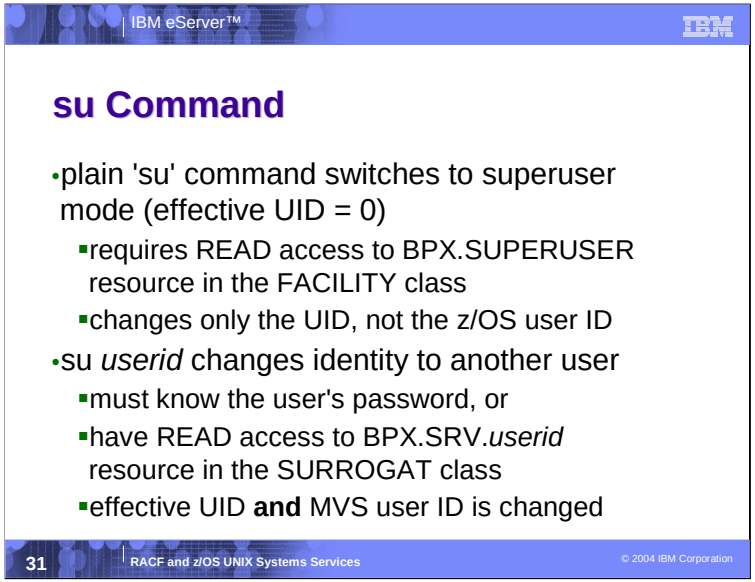
This is similar to Program Access to Data Sets (PADs).



The set-uid and set-gid bits are reset when the owner of the file is changed or the file itself has changed.

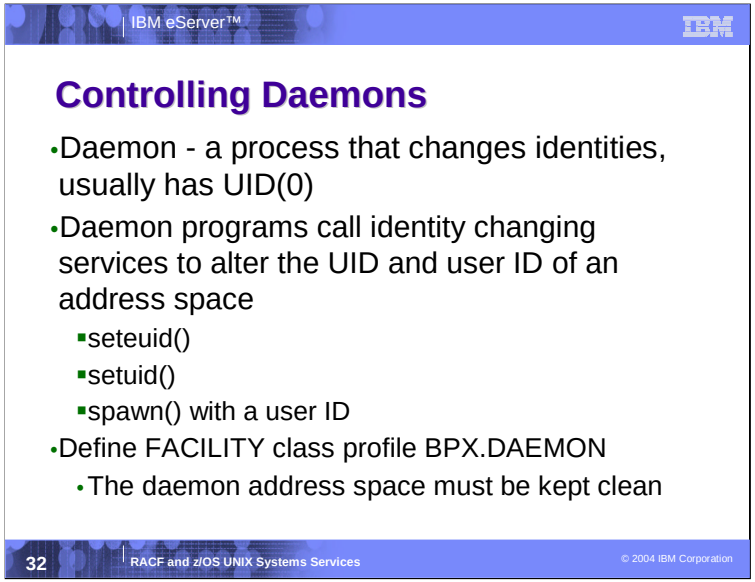
When a file system is mounted with the NOSETUID option, any set-uid and set-gid bits are ignored. Use this on untrusted or remote file systems, since they may have set-uid 0 (superuser) executables that could damage your system.

set-uid files can be dangerous in that some user who is superuser on his desktop risc box can create set-uid root programs, and if these files are mounted on z/OS, they will retain their UID 0 owner, and set-uid status. MOUNT has NOSETUID options to avoid this sort of thing.



Note that su w/ userid completely assumes the target user's identity: both user ID and UID (switching to superuser keeps your userid). Use "-s" to avoid a password prompt if you have SURROGAT authority, or just hit enter at the password prompt
If you have SURROGAT authority, you can issue **su -s *userid*** to skip the password prompt. If you forget to specify **-s**, then just hit enter at the prompt and your SURROGAT authority will be honored.

Essentially, su creates a new shell environment with the authority of the new user.



Daemon programs can issue...
•setuid() - set uid
•seteuid() - set effective uid - invokes z/OS SAF services to change the z/OS identity of the address space
•spawn() - create and start a child process that runs a program in a new Address Space (fork and exec) - security info from the parent AS will be propagated to the child unless _BPX_USERID environment variable specifies otherwise (R9 C/C++ RTL pg 1409)
... to change the OS/390 identity in an address space or process in order to run work on behalf of a user. *C/C++RTL*
Daemon authority - user ID that is authorized to change to any z/OS identity that has an OMVS segment without knowing the target userid's password.
Or ' BPX.SRV.target_userid' authority can change identies w/o a pw . Class SURROGAT *(web notes)*

Only Superusers with access to BPX.DAEMON authority can invoke these functions without authenticating the user, or without having surrogate authority. If not defined, an uncontrolled executable can be loaded .
May want to protect privileged z/OS users by not allowing them to have an OMVS segment or a UID.
ALU privuser OMVS(NOUID) *(R8 Unix Planning p 250)*
Clarification: any user (uid) can use setuid() if they either authenticate or have surrogate authority. This does not change when BPX.DAEMON is defined. BPX.DAEMON scopes which superusers can use setuid without authentication/surrogate and enforces program control.

If BPX.DAEMON is defined, the __passwd() service will enforce program control

All programs loaded must be controlled

PROGRAM profiles covering all programs from z/OS libraries (UACC READ is OK)

Controlled attribute for programs from the HFS

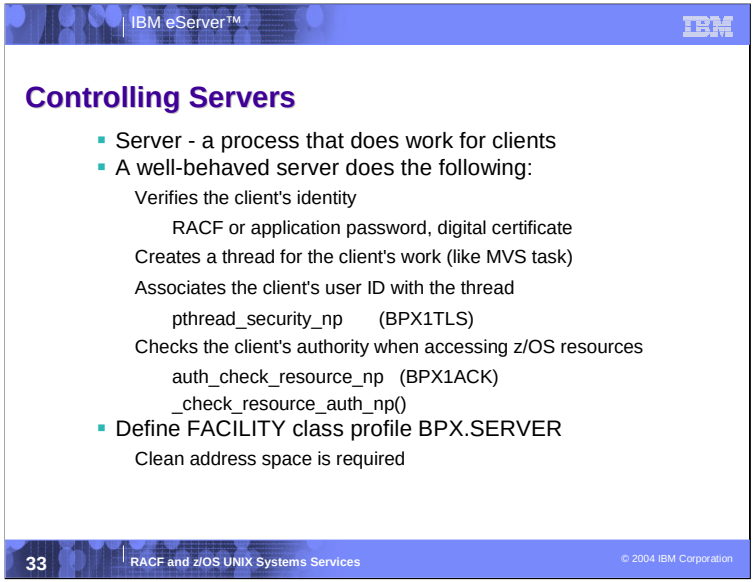
Set with *extattr +p*

Issuer needs authority to BPX.FILEATTR.PROGCTL

Turned off automatically if file is changed

Ignored if HFS mounted with *nosetuid* or *nosecurity*

IBM-supplied daemons shipped in /usr/bin with sticky bit on so SYS1.LINKLIB copy will be used, or shipped with +p extended attribute



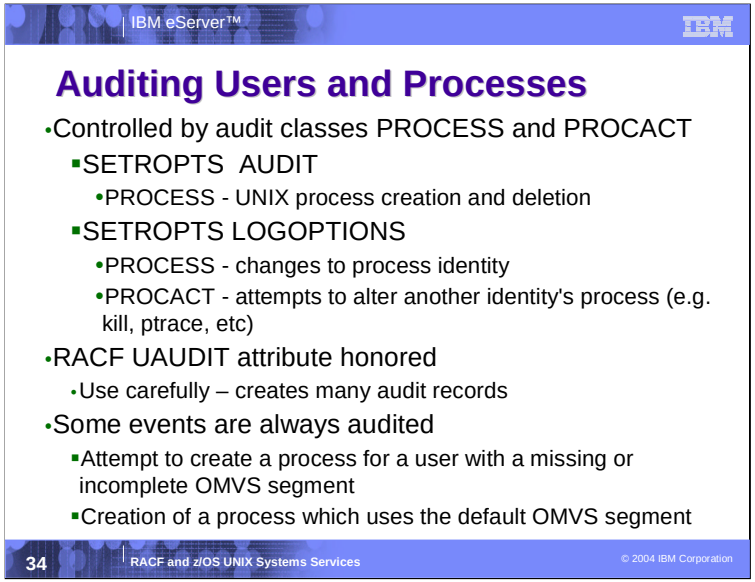
Ensure control over Server to ensure they do what you want. Servers designed on other platforms have their own security...
Not all Servers authenticate their clients or pass out resources properly

pthread_security_np = Create/Delete the security environment for the callers thread *(RACF InitACEE Callable Svc)*

auth_check_resource_np = Callable service to determine a User's Access to Protected z/OS Resource *(R9 Unix ASM Callable Services p 59)*
Caller must have READ access to BPX.SERVER Facility class profile (Or UID=0 if BPX.SERVER not defined)

_check_resource_auth_np() = C function call to determine Access to z/OS Resources
Caller must have read permission to FACILITY class BPX.SERVER
(Or UID=0 if BPX.SERVER not defined) (R 9 C/C++RTL)

BPX.SERVER profile is useful for programs that will establish a task-level(z/OS) or thread-level(UNIX) security environment.
Examples: IBM http Server (formerly Web Server)
DCE Application Server
Daemon and server programs must be protected
Sticky bit on in file system copy - get copy from LPA which is program controlled.
SYS1.LINKLIB is protected with PROGRAM and DATASET profiles
program control **ext**ended **attr**ibute **"extattr +p"** for HFS-resident programs
nosetuid **= no program controlled entities**
PADS - Program Access to Data Sets - make the use of a protected program a condition for access to a dataset or load module.
SETR WHEN(PROGRAM(pgm_name))
extattr command will fail if BPX.FILEATTR.* is not defined, or if the FACILITY class is inactive.



UAUDIT is mentioned for completeness...to demonstrate that it applies to UNIX as well. However, be careful as it will generate **lots** of audit records.

The default UID auditing has two aspects:

1) creation of the process is always audited

2) subsequent events, *if audited*, specify that a default UID was used

UNIX Auditing ...

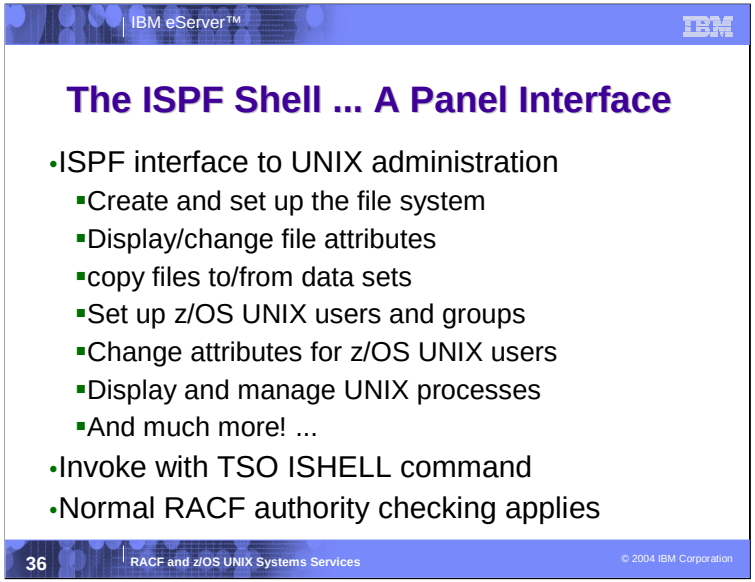
The results

- Type 80 SMF records
- ICH408I messages for resources and services

```
ICH408I USER(SYS) GROUP(TST) NAME(OOPS)
CLASS(PROCESS)
OMVS SEGMENT NOT DEFINED
```

- RACFRW information is incomplete
- Use SMF Data Unload utility (IRRADU00)

See: z/OS Security Server RACF Auditor's Guide and z/OS Security Server Macros and Interfaces



Setup -> User ... can be used to set up a new OMVS user...HFS dataset and all

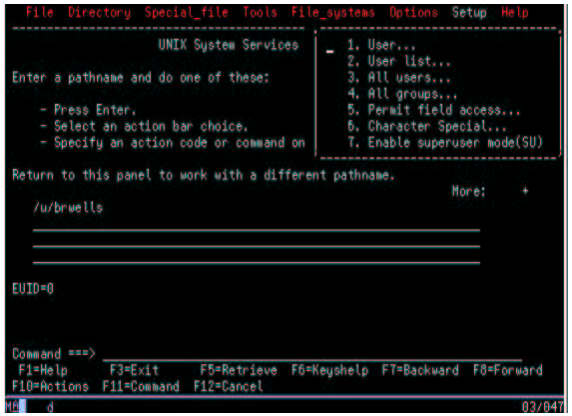
Setup -> User list ... will show all users and their UIDs. This can be sorted by UID.

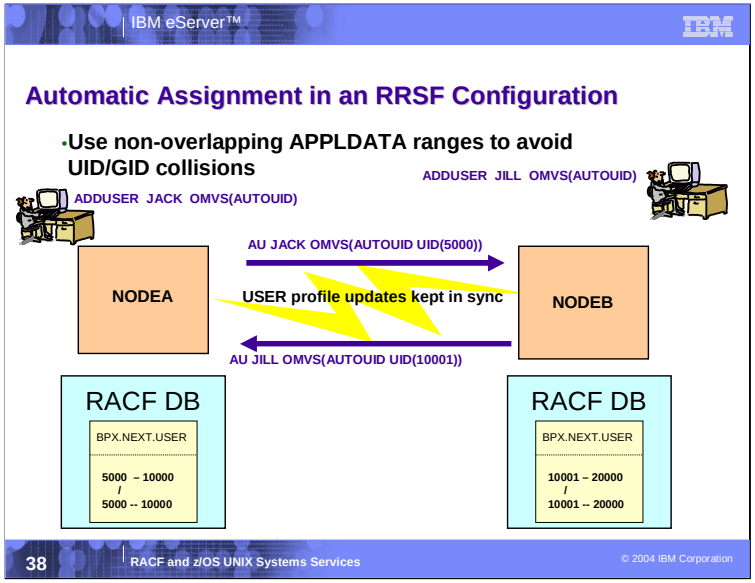
Setup -> All users ... can assign unique UID, initial program, and home directory based on a specified prefix. The UID will start at one higher then the current high value. You will also be asked if you want to assign GIDs to all groups during this setup. If you don't, and a given user's group has no GID, processing will stop.

Setup -> All groups ... can assign a unique GID to all groups. There is no "Group list" function, and so you can't see a sorted list of current GID values.

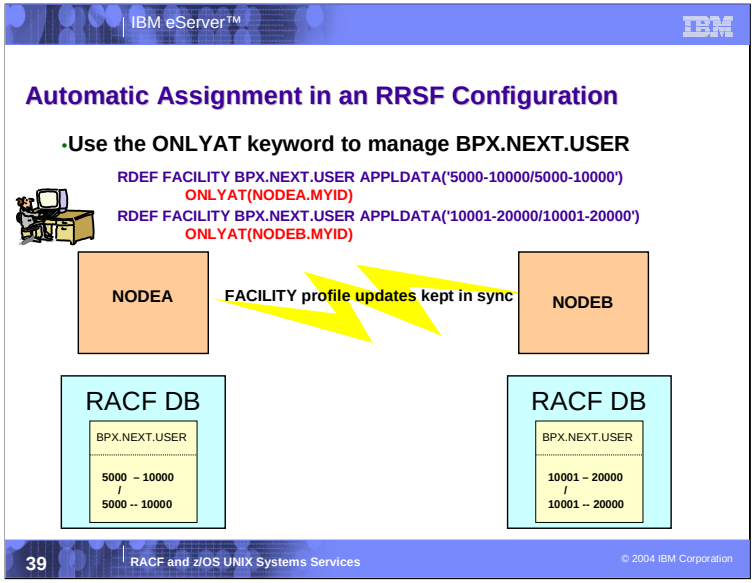
Setup -> Permit field access ... will do the steps shown in the appendix.

The ISPF Shell ... A Panel Interface





In an RRSF environment where user updates are kept in sync across the network, you want to avoid UID/GID collisions when AUTOUID/AUTOGID is used on multiple nodes. This is accomplished by specifying unique ranges of ids in the BPX.NEXT.USER APPLDATA for each node, thus making sure the various nodes do not derive the same value as another node. Assuming that the nodes are kept in sync, then even explicit UID/GID assignment should not result in collisions, unless the same value is explicitly assigned on different nodes at almost the same time, and they "cross in the mail." Even then, the assignment should fail on the remote node, and the administrator will be notified, and get a chance to fix their error.



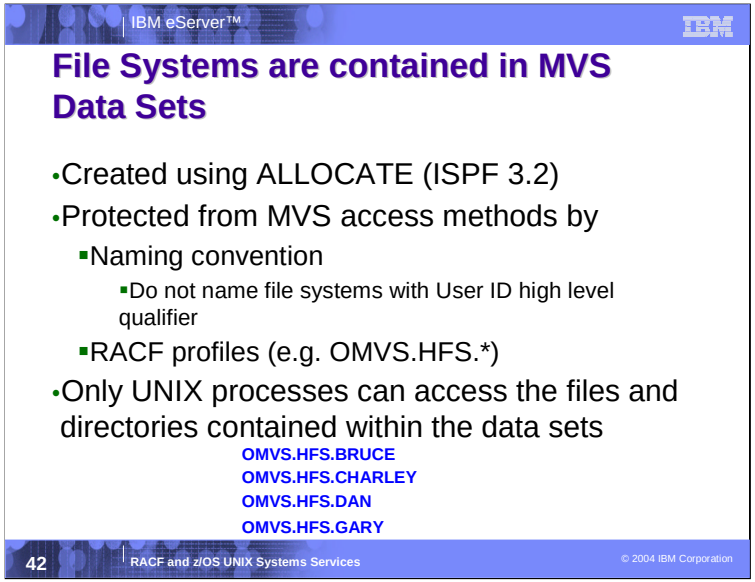
This is only a consideration if you are using RRSF automatic command direction for the FACILITY class. Since you went to so much trouble to plan for non-overlapping BPX.NEXT.USER ranges on each of the nodes, you wouldn't want NODEA's update being propagated to the other nodes and wiping out their local changes! So, you must use the ONLYAT keyword, even when changing the profile on which you are logged on. ONLYAT tells RACF not to propagate the command outbound. Note that the AT keyword is not sufficient, since it will still be subject to propagation.

RACF updates the APPLDATA itself when deriving unique UIDs and GIDs. But RACF is careful to prevent the propagation of these internal updates.

Recap

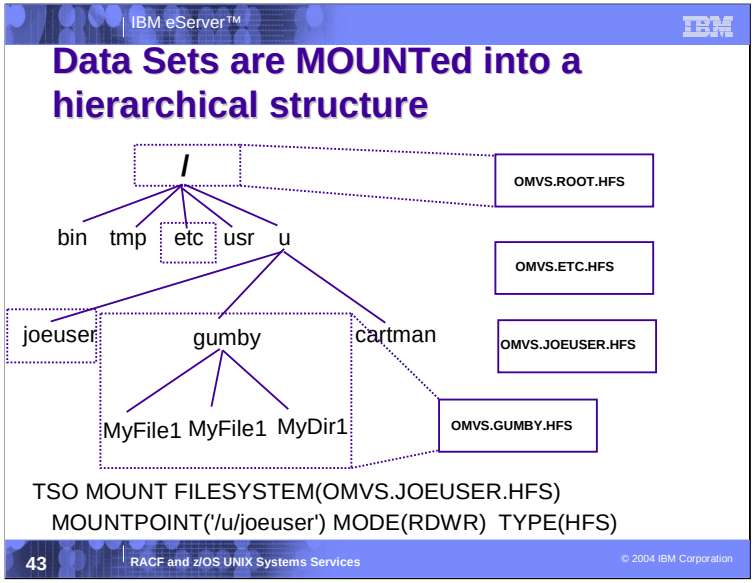
- UNIX systems require an integer value as a user or group identity
- This identity is contained in the RACF database, which acts as the 'user and group registry'
- Various mechanisms can effect a change in user/group identity of a process
- Traditional UNIX security mechanisms are extended on z/OS
- UNIX security events are audited in a familiar RACF fashion

z/OS Unix System Services File Systems



Since the operating system underlying UNIX System Service is MVS, the UNIX file systems are stored within MVS datasets. You want to prevent end users from accessing these datasets through traditional MVS access methods. You should protect them with RACF profiles.
Can't access the files and directories from the 'outside'. That is, using traditional MVS access methods.

Must access the files using UNIX semantics from a UNIX process.



Mount steps involve:

Allocate the HFS data set

mkdir a home directory for the user in /u

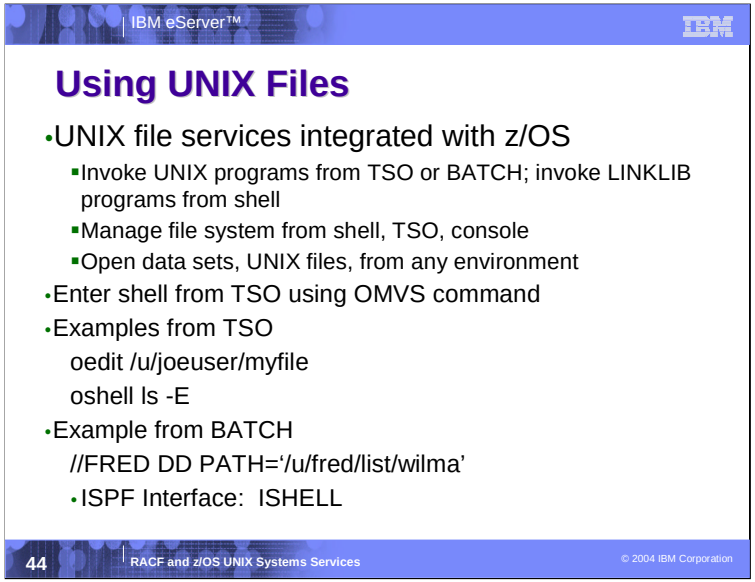
MOUNT the HFS data set (file system)

chown the /u/joeuser directory to joeuser

Mount table specified in BPXPRMxx

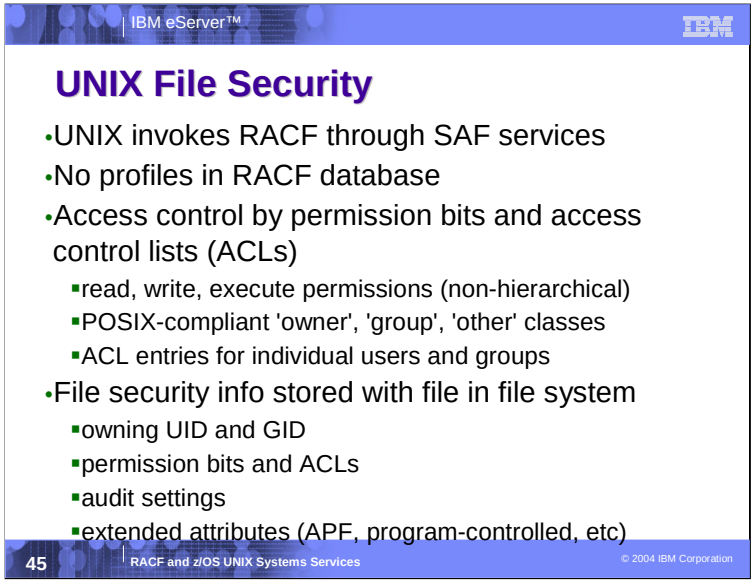
file systems can be mounted dynamically with the TSO MOUNT command or the UNIX mount command

view file system with DISPLAY OMVS,F or df command

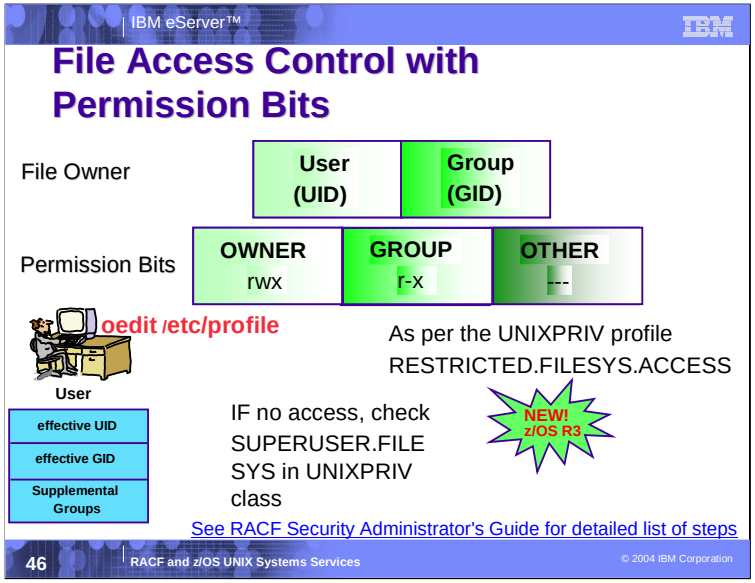


There are many ways to access UNIX files. You can enter a UNIX shell with the OMVS command and issue UNIX commands.

Or you can manipulate UNIX files from the TSO command line or from JCL.



RACF, of course, gets involved in providing security for UNIX files. No RACF profiles, however, are used to provide security. All the security information for a file is kept with the file.



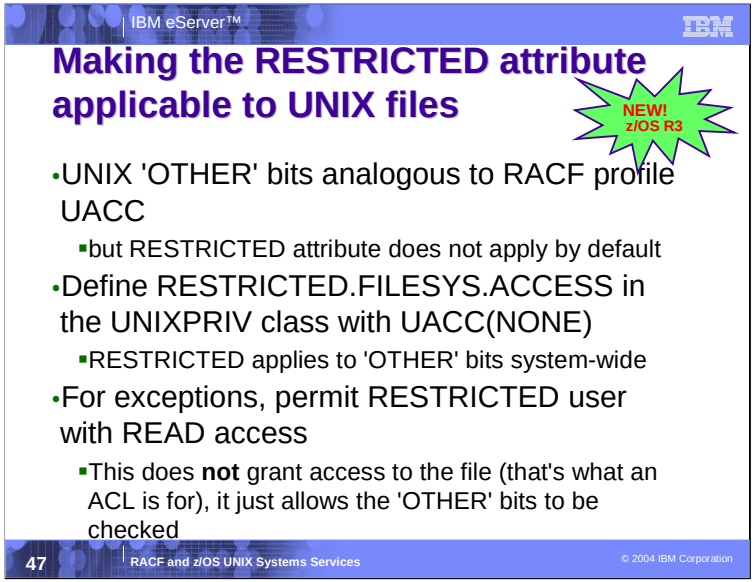
This chart illustrates the checking which is done to determine file access.

It does not illustrate the complete list of steps, but rather, demonstrates the basic POSIX access algorithm, supplemented by RACF's UNIXPRIV checking. This does not include ACL checking. That will be covered later.

The z/OS V1R3 SAG was updated with a comprehensive list of authorization steps, like what has always been in there for RACF profile checking. It's worth a 5-minute read.

See speaker note for the upcoming flowchart slide for an overview of the steps which were omitted from this slide.

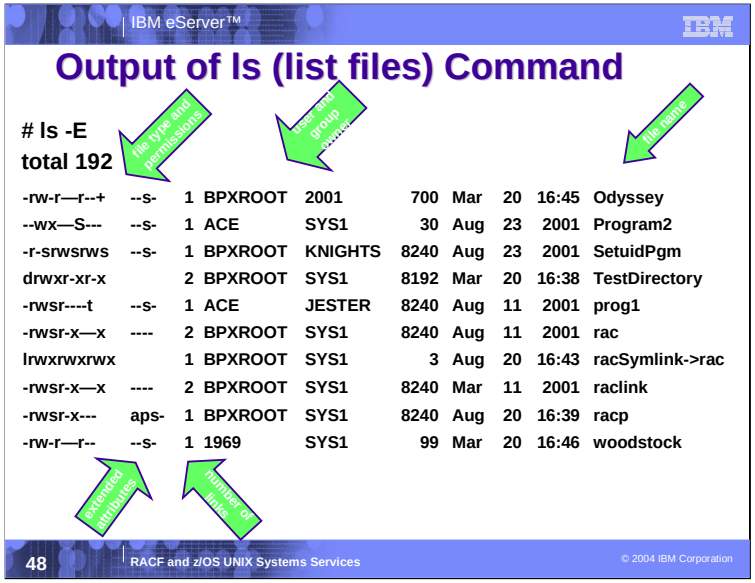
OW53183 (R3): UNIXPRIV used for shell test operators



The RESTRICTED attribute for a user prevents the user from getting access based on the UACC in a RACF profile. In z/OS R3, the profile RESTRICTED.FILESYS.ACCESS profile in the UNIXPRIV class allows that to apply to UNIX files as well.
Note that RESTRICTED.FILESYS.ACCESS will be checked for RESTRICTED users regardless of whether an ACL exists, so this function can be exploited regardless of whether you plan to use ACLs or not.

Using UACC(READ) on RESTRICTED.FILESYS.ACCESS doesn't work, since by definition, a RESTRICTED user cannot be granted access via UACC!!! This would be a meaningless thing to do anyway, given that you simply would not define RESTRICTED.FILESYS.ACCESS if you want 'other' bits to be checked for RESTRICTED users.

Even today, a RESTRICTED user can be permitted to SUPERUSER.FILESYS and thus be granted access to files, though I cannot think of why anyone would do that! In any case, it will continue to work that way.



owning UIDs and GIDs are translated to RACF user IDs and group names, if possible. UNIXMAP or AIM will speed this up, especially when there are unassigned UIDs/GIDs

The file Odyssey is owned by a group which could not be mapped. woodstock is owned by a user which could not be mapped. This can happen from chown being issued with a numeric, NFS mounting a remote file system, or a user having been deleted

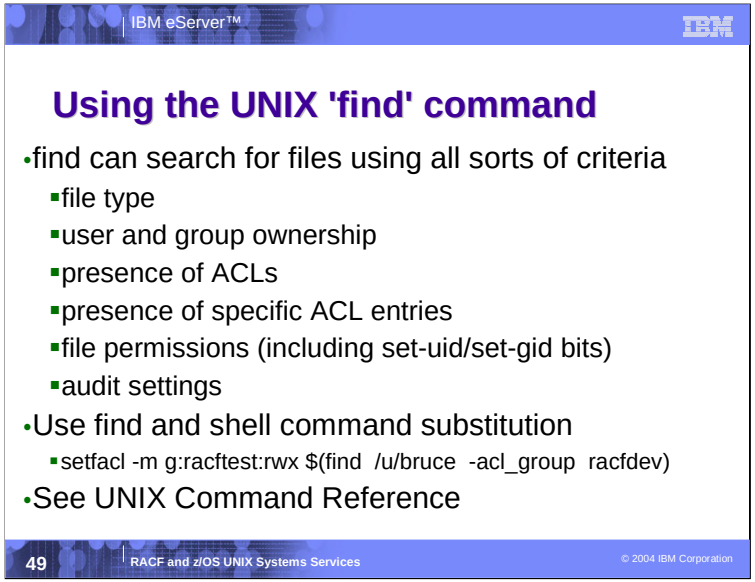
SetuidPgm has the setuid bit on, Program2 has the setgid bit on, but the execute bit is not on.

prog1 has the sticky bit on. It will be fetched from LINKLIB

racp has the program control and apf bits on (need AC=1)

raclink is a hard link to rac, racSymlink is a symbolic link

+ in the permissions means that an ACL entry exists



When you need to update many UNIX files with the same updated, the UNIX find command is very useful. Use the find command to ‘select’ the files, and then use shell command substitution to make the output of the find command into input for another command (such as setfacl).

Since ACLs are not 'shared', there will normally be lots of them dispersed throughout your HFS.

the find command, in conjunction with shell substitution, will help you to manage them over time

E.G.

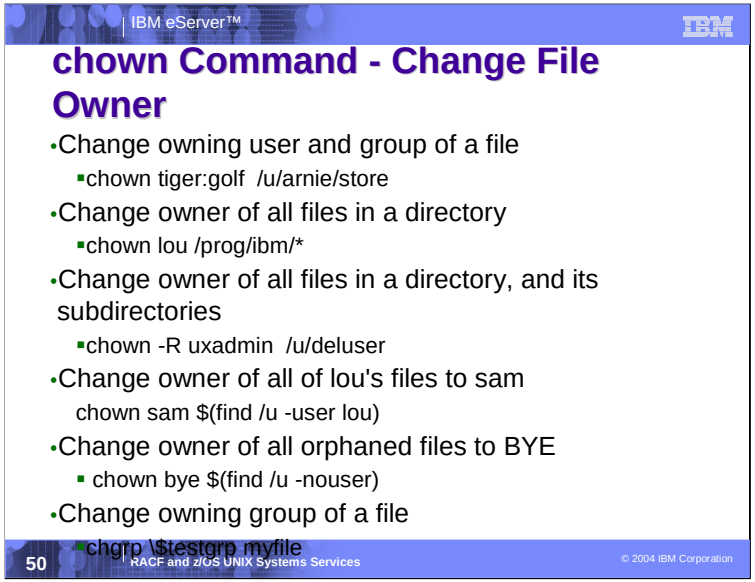
to add an entry to every directory within a subtree

to permit the group RACFTEST to every ACL which has an entry for RACFDEV (this is the example shown)

to permit a user to all files owned by JOEUSER

to delete all ACLs within a given subtree

See the UNIX Planning Guide for more examples



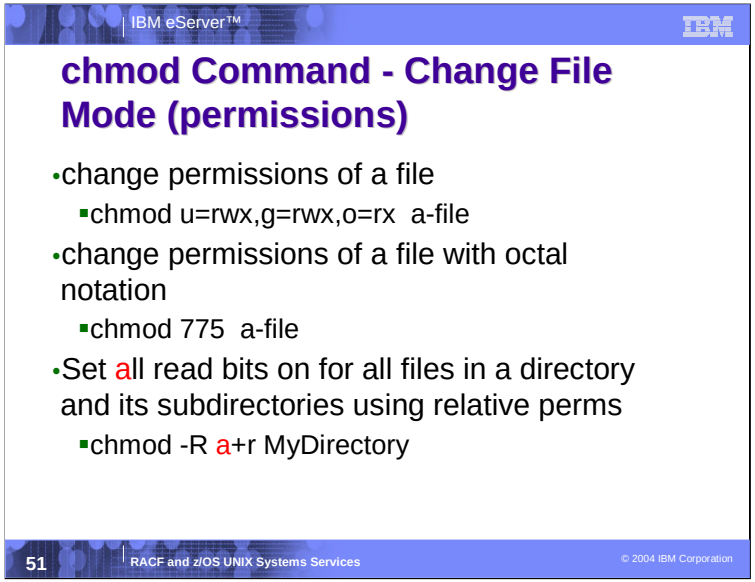
This shows various formats of the chown/chgrp command, and demonstrates some features of this particular command, and of the UNIX shell in general

the * is a shell wild card operator. In this example, the shell will apply the chown command to any file/directory within the /prog/ibm directory (but will not recurse to lower levels)

chown, and some other commands (e.g. chmod, but not sefacl) have a 'recursive' option, -R.

shell command substitution can be used to apply a command to a list of file names returned by 'find'

The dollar sign is a special shell character, so if it is used in a group name, it must be 'escaped' by preceding it with a backslash.



This slide demonstrates some of the basic chmod options. You can use symbolic form or octal notation.

Bit definition is:

4000 Set-user-ID bit

2000 Set-group-ID bit

1000 Sticky bit

0400 User read

0200 User write

0100 User execute (or list directory)

0040 Group read

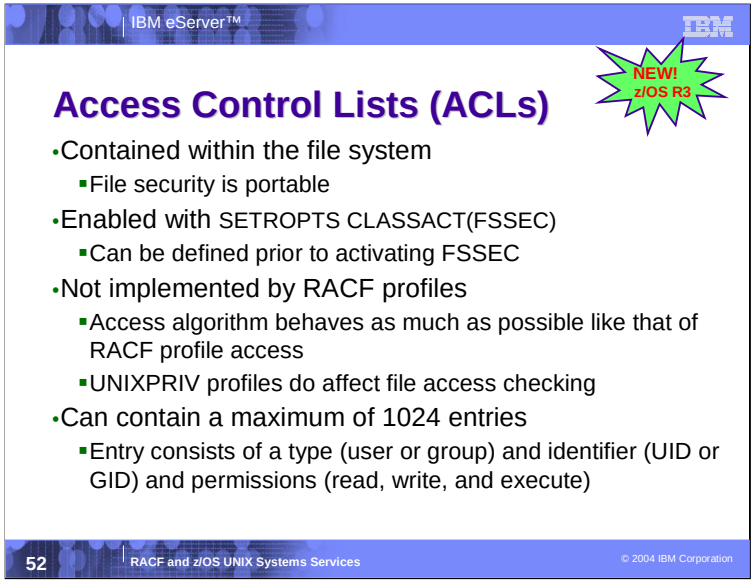
0020 Group write

0010 Group execute

0004 Other read

0002 Other write

0001 Other execute



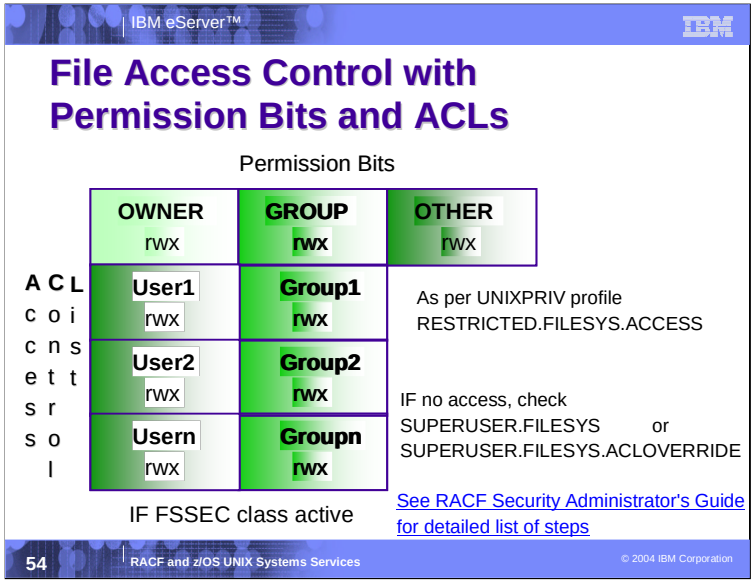
ACL is Deleted automatically with file (even on downlevel systems)

Managed by RACF using SAF callable services

R_setfacl, makeFSP, ck_access, query_file_security_options

Access Control Lists (ACLs) ...

- Are displayed with the getfacl UNIX command and created, modified, and deleted with the setfacl UNIX command
  - Must be UID(0), file owner, or have READ access to UNIXPRIV resource SUPERUSER.FILESYS.CHANGEPERMS
- Support inheritance
  - 3 types of ACLs
    - Access ACL
    - Directory Default ACL
    - File Default ACL



This chart extends the previously shown access algorithm to include ACLs.
Order of checking

1) Owner bits
2) User ACL entries
3) Group bits
4) Other bits

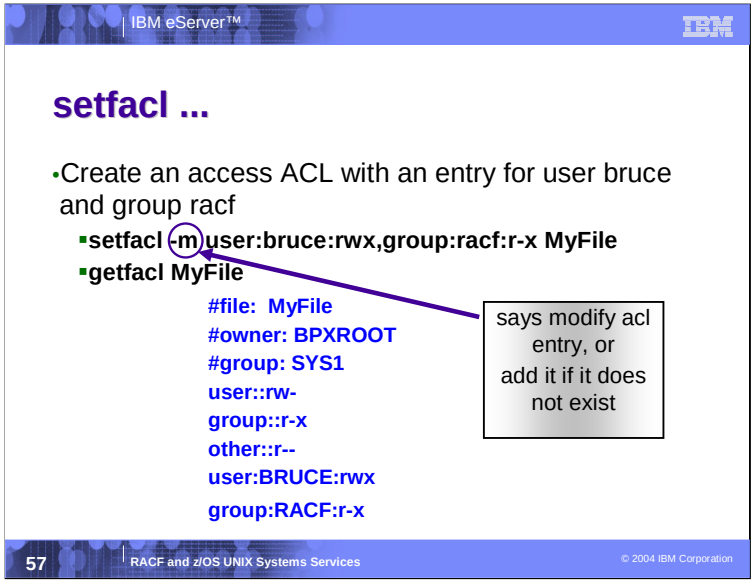
getfacl and setfacl commands

- Can also be used to display/modify the POSIX permission bits
  - allows use of a single interface
    - chmod only necessary to set sticky, set-uid, and set-gid bits
- ACL can be set from contents of a file
  - thus, output of getfacl can be piped into setfacl via stdin
  - Reduces typing
  - Allows use of "named ACLs"

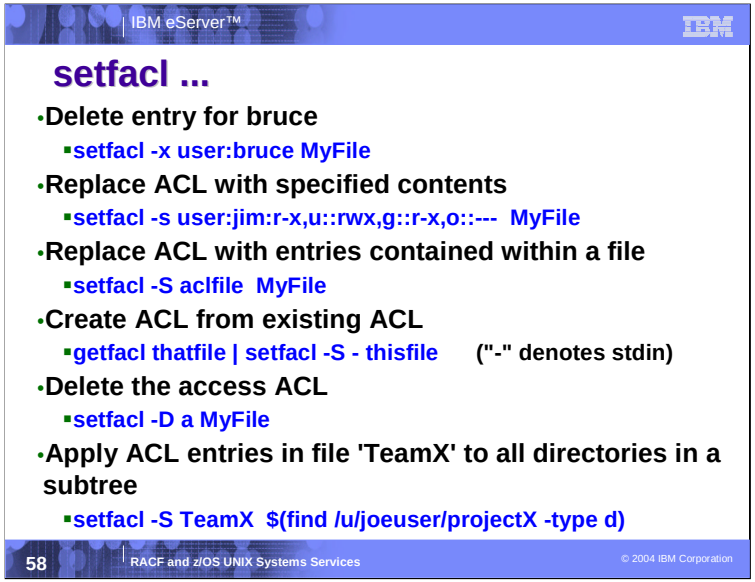
This shows the output of the getfacl command on a file with no ACL entries.

–m Specifies that the comment header (the first three lines of each file's output) is not to be displayed.

–o Displays only the extended ACL entries. Does not display the base ACL entries.



setfacl can also be used to change the posix permission bits. This lets you use getfacl/setfacl without having to also do ls and chmod commands (though ls and chmod must be used to see/update the set-uid-, set-gid, and sticky bits)
to update the posix bits, use the 'u' qualifier for the owner bits, the 'g' qualifier for the group bits, and the 'o' qualifier for the other bits, and omit the middle qualifier for the user or group name. For example, the moral equivalent of **chmod 755 myfile** would be **setfacl -m u::rwx,g::r-x,o::r-x myfile**



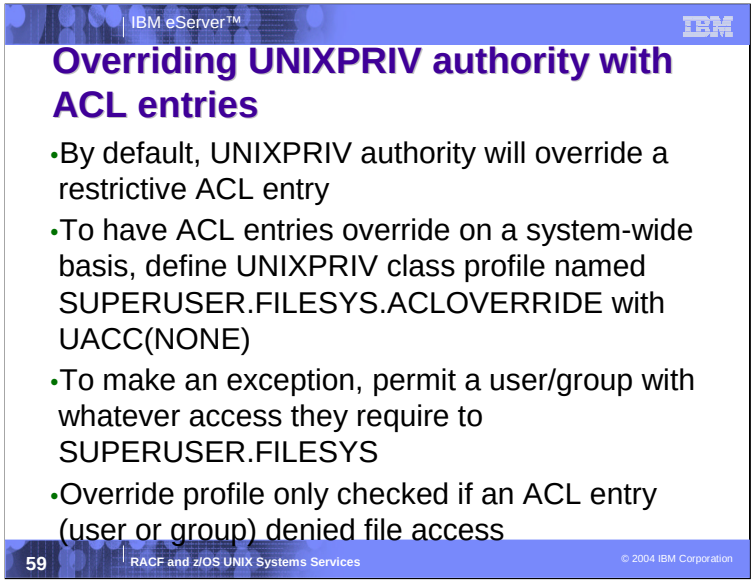
the -x option removes the specified ACL entries. The permission qualifier need not be specified

the -s option means replace the current contents of the ACL with the entries specified on the command line. -s requires that you also specify the posix permissions

the -s, -m, and -x options have capital letter options which take their input from a file. Note that I can maintain an ACL in a file (a 'named ACL', if you will) and apply that ACL to files at will.

'-' as a file name is a special notation for stdin, which allows you to pipe output from a getfacl command into setfacl

And of course, our old friend 'find' is our only mechansim to apply setfacl to a set of files.

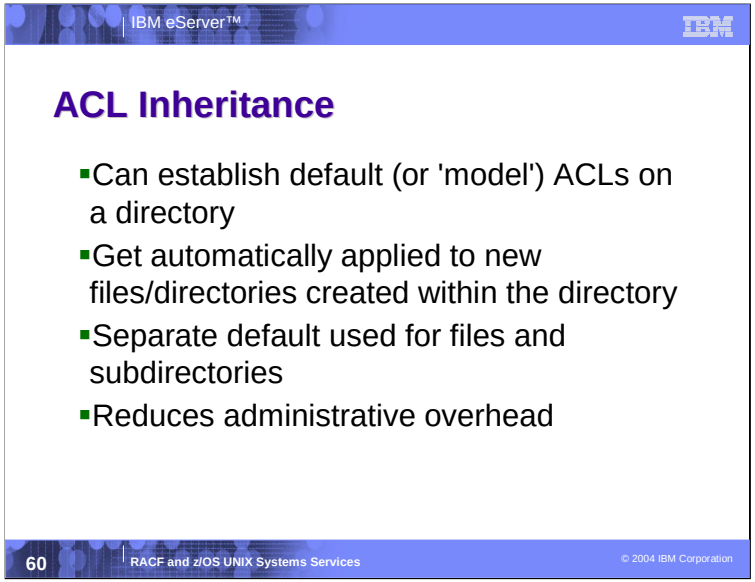


This will **not** override UID(0)/trusted/privileged

Can use this to provide a mechanism of scoping UNIXPRIV access authority to certain file system subtrees
Define the override profile with UACC(NONE)

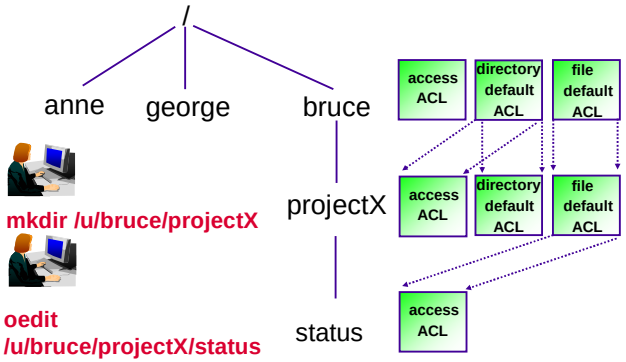
Define ACL on top directory of a given subtree and permit the user (or group) with limited access permissions

if permitting by group, make sure the 'group' bits, or another group entry does not provide access



ACL inheritance allows you to automatically apply ACL protection to newly created objects

ACL Inheritance ...



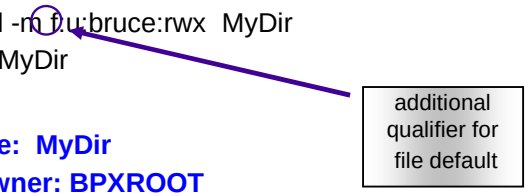
getfacl and setfacl ...

- Create a directory default ACL
  - setfacl -m default:user:bruce:rwx MyDir
  - or: setfacl -m d:u:bruce:rwx MyDir
  - getfacl -d MyDir

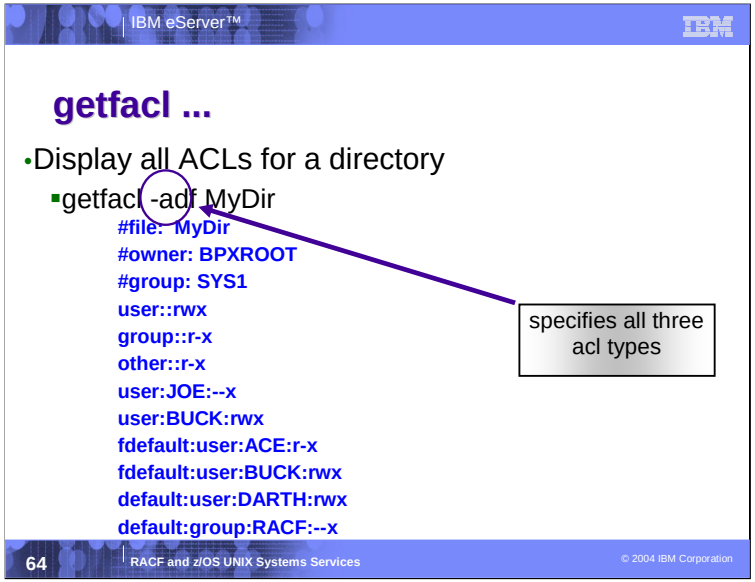

```
#file:  MyDir
#owner: BPXROOT
#group: SYS1
default:user:BRUCE:rwx
```

getfacl and setfacl ...

- Create a file default ACL
  - setfacl -m fdefault:user:bruce:rwx MyDir
  - or: setfacl -m fu:bruce:rwx MyDir
  - getfacl -f MyDir



#file: MyDir
#owner: BPXROOT
#group: SYS1
fdefault:user:BRUCE:rwx

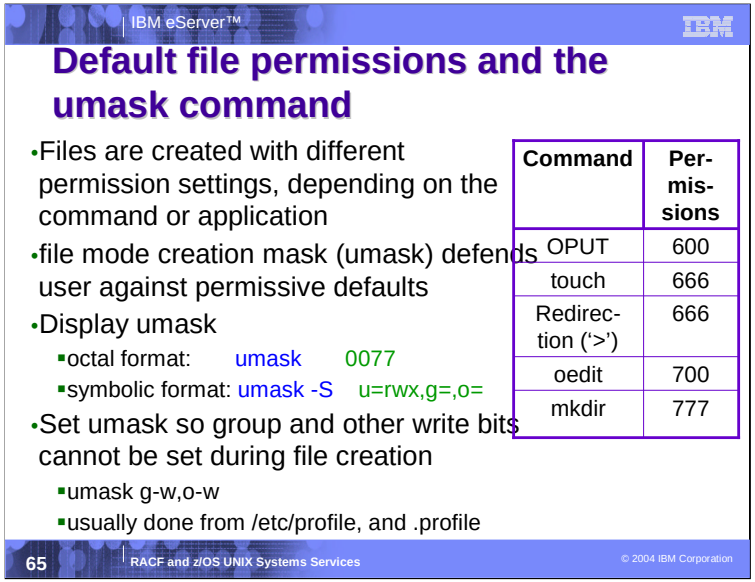


In –adf:

a is access ACL

d is directory default ACL

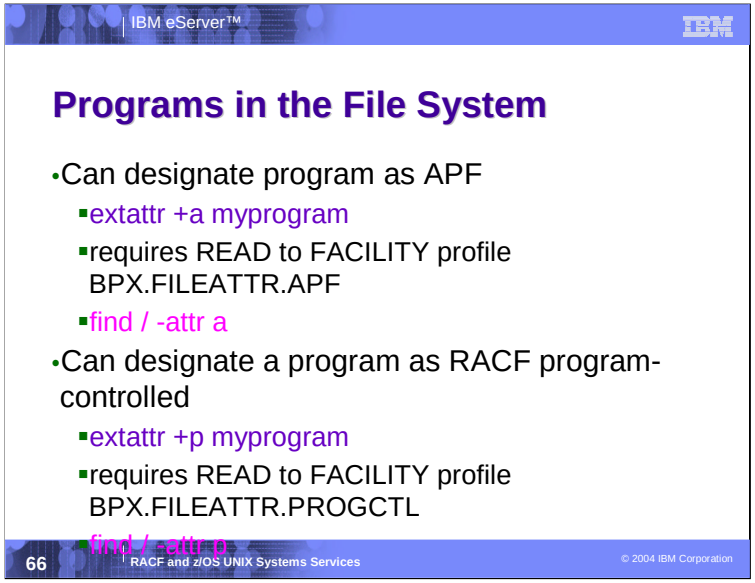
f is file default ACL



The initial file permissions are as specified in the mode parameter of the open() call made by the creating application

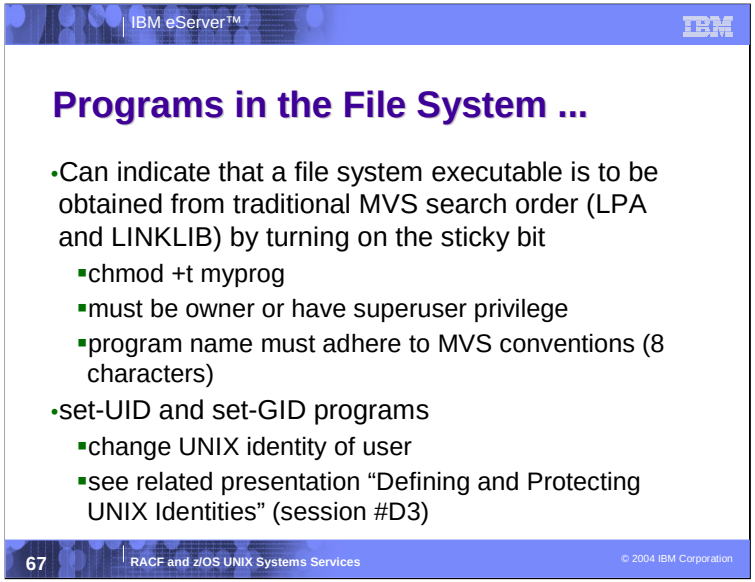
umask specifies which permission bits are to be masked off during file creation. Typically, this is used to mask off the group and other 'write' bits.

This is typically done on a system-wide basis by /etc/profile, and on an individual user basis by the .profile script in the user's home directory. For you VMers out there, these files are analogous to SYSPROF EXEC and PROFILE EXEC, respectively.
umask **cannot** be used to initialize bits **on**



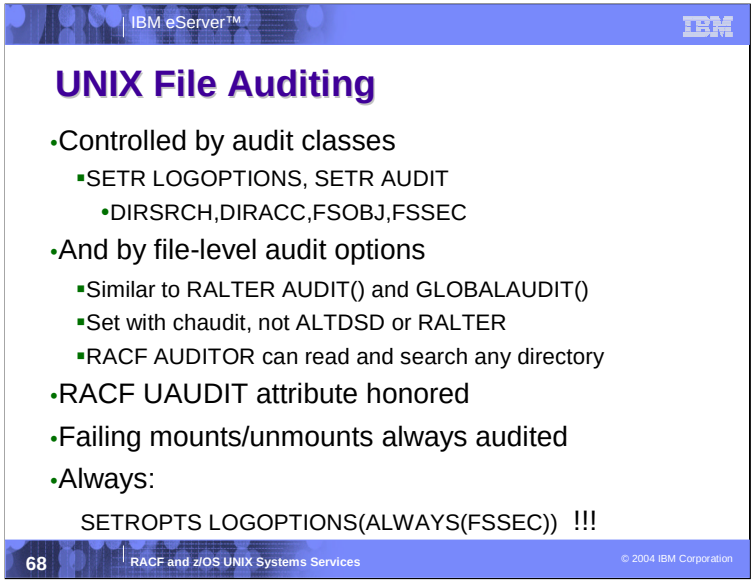
Whenever an APF or program-controlled file is written to, the extended attributes are reset, for integrity reasons. Only an authorized user (e.g. SMPE) can turn the bits back on.

Note that UID(0) is not sufficient to turn these attributes on! This aids in enforcing z/OS UNIX level security for servers and daemons.



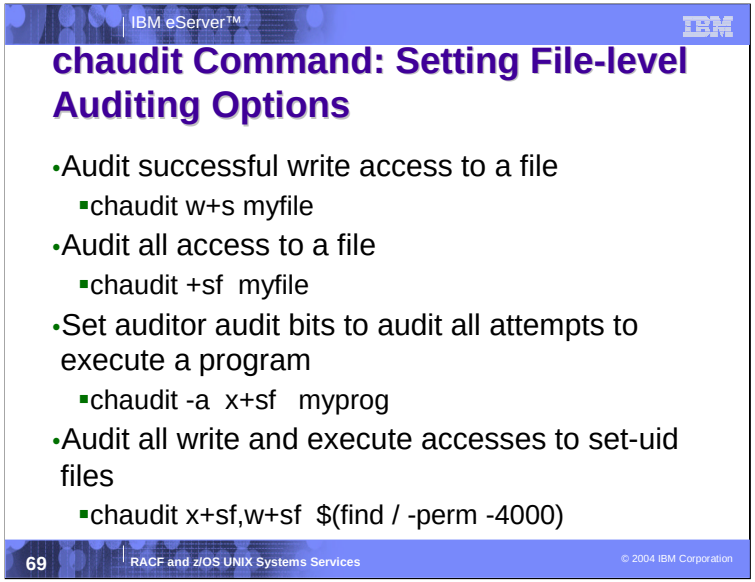
Whenever a set-uid or set-gid file is written to, the set-uid and set-gid bits are reset, for integrity reasons.

Similarly, if the file owner is changed (user and/or group), the appropriate set-id bit is reset



There are three separate access-related classes: DIRSRCH, DIRACC, and FSOBJ. These log accesses to files and directories, and split into separate classes for auditing granularity. Especially DIRSRCH. You do not want to log successful directory searches. On the other hand, you may want to do so with directory accesses (DIRACC) or file accesses (FSOBJ). SETR CLASSACT has no effect on these classes.

FSSEC logs attempts to alter security information (chown, chmod, setfacl, etc). Kind of like auditing changes to RACF profiles. Unlike the 3 classes above, you will not get this logging by default. Also unlike the above, SETR CLASSACT(FSSEC) has meaning...it enables ACLs.

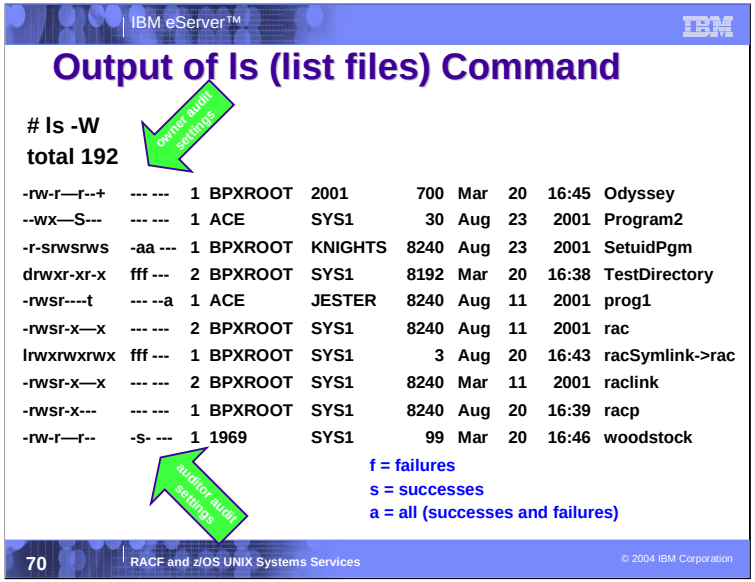


a user with the RACF AUDITOR attribute can change the AUDITOR file-level settings for any file

a superuser can change the owner file-level settings for any file

a general user can set the owner file-level settings only for files which (s)he owns

There is no UNIXPRIV profile which controls audit settings



ls -W shows the audit settings

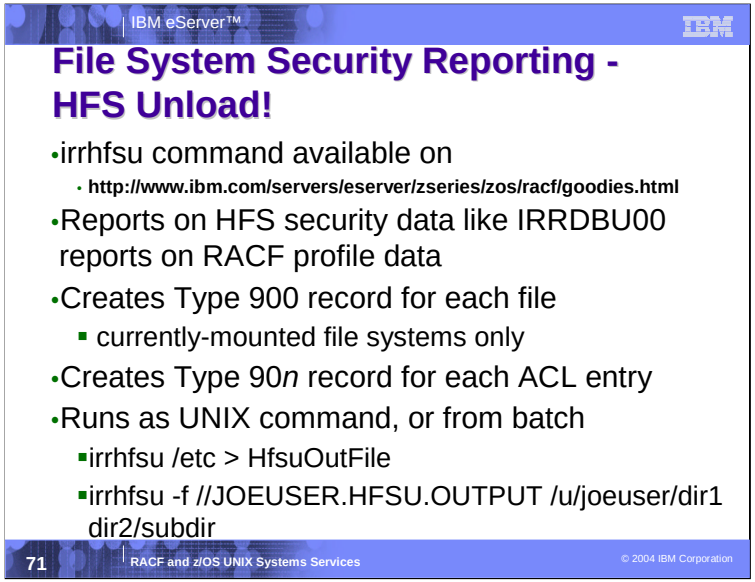
the first column of audit settings shows the owner options, the 2nd column shows the auditor options.

'f' indicates that failed accesses for the relative permission (e.g. read, write, execute access) are being logged

's' indicates that successes are being logged

'a' indicates that all accesses (successes and failures) are being logged

the find command can locate files with certain audit settings



For ACLs:

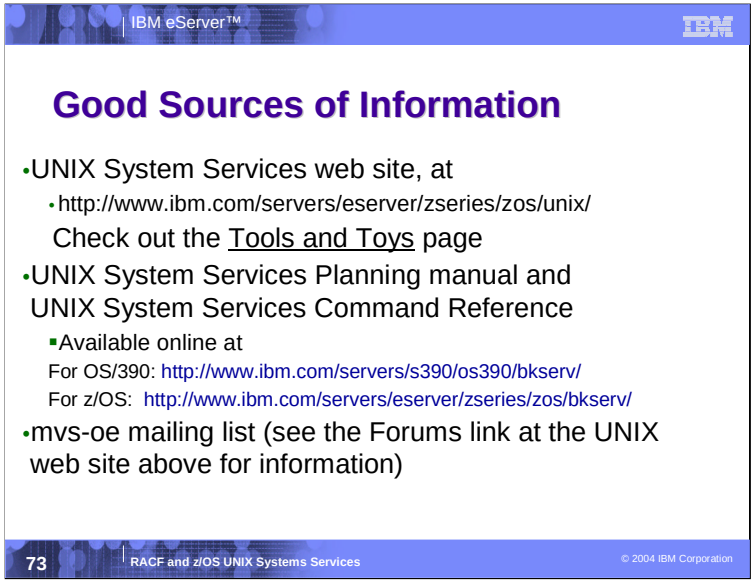
Type 901 describes an access ACL entry

Type 902 describes a file default ACL entry

Type 903 describes a directory default ACL entry

Recap

- UNIX file systems are contained in MVS datasets
- File systems are mounted at 'mount points' (directories) to create a hierarchical file system
- File security information is contained within the file system (not in the RACF database), and is managed using UNIX commands and interfaces
- Access Control Lists allow you to specify a list of users who can access a file
- Programs in the file system can be APF authorized and program-controlled
- Actions are auditable through RACF and SMF and can be reported on using the SMF Data Unload utility (IRRADU00)



BPX1SEC1 SAMPLIB - TSO Clist of RACF commands for security setup

Good Sources of Information ...

- RACF web site, at
  - http://www.ibm.com/servers/eserver/zseries/zos/racf/

  See Downloads page for HFS Unload
- RACF Security Administrator's Guide (UNIX chapter) and RACF Auditor's Guide
  - Available online at

  For OS/390: http://www.ibm.com/servers/s390/os390/bkserv/

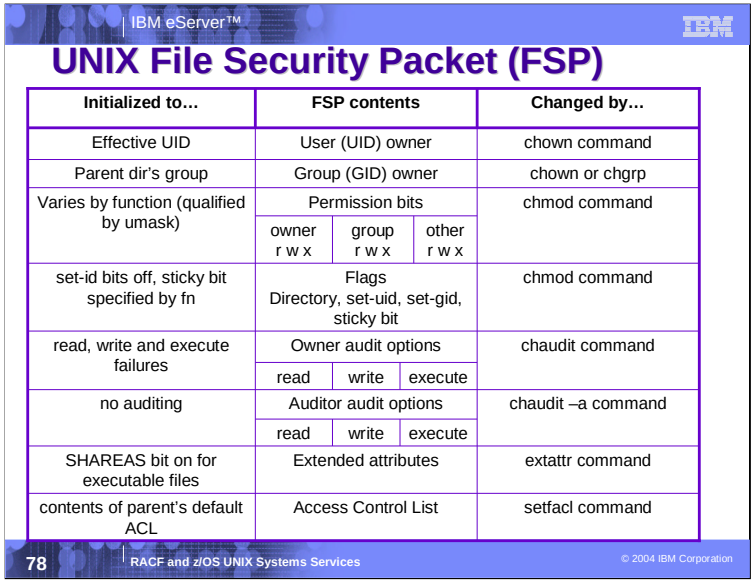
  For z/OS: http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
- racf-l mailing list (see the front-matter in any RACF book for information)

Questions?

Supplemental Material

Terminology

- POSIX - Portable Operating System Interface
- FSP - File Security Packet
- process - UNIX 'address space'
- USP - User Security Packet
- ACL - Access Control List
- SAF - System Authorization Facility
- APF - Authorized Program Facility



the extended attributes are also security-related, though technically speaking, they do not reside in the FSP.

The apf attribute marks an HFS program as APF authorized. It is set with the extattr +a progname command, and requires authority to BPX.FILEATTR.APF in the FACILITY class

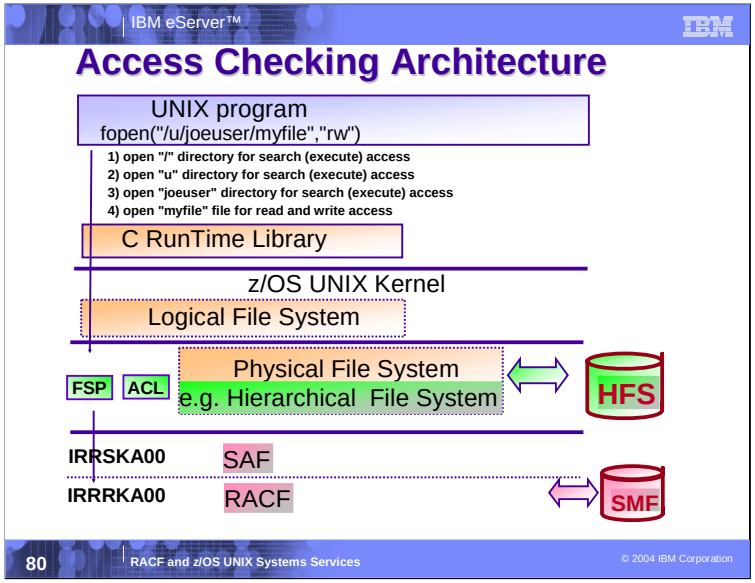
the progctl bit marks an HFS program as being program controlled. It is set with the extattr +p progname command, and requires authority to BPX.FILEATTR.PROGCTL in the FACILITY class

setuid, setgid and APF bits get reset when the file is changed

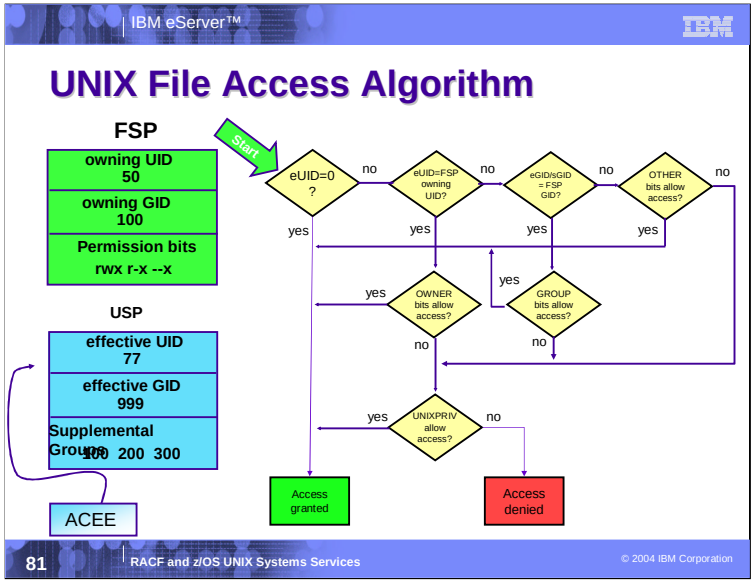
UNIX File Security Packet (FSP) ... who can change what?

Security Field	Required Authority
Owning UID	▪UID 0 ▪File owner if CHOWN.UNRESTRICTED is defined in the UNIXPRIV class ▪READ access to UNIXPRIV resource SUPERUSER.FILESYS.CHOWN
Owning GID	▪UID 0 ▪Owner, if a member of new group ▪File owner if CHOWN.UNRESTRICTED is defined in the UNIXPRIV class ▪READ access to UNIXPRIV resource SUPERUSER.FILESYS.CHOWN
File mode (permissions and flags) and ACL	▪UID 0 ▪File owner ▪READ access to UNIXPRIV resource SUPERUSER.FILESYS.CHANGEPERMS
Owner audit options	▪UID 0 ▪File owner
Auditor audit options	▪RACF Auditor
Extended attributes	READ access to FACILITY class resource named: ▪APF – BPX.FILEATTR.APF ▪Program control – BPX.FILEATTR.PROGCTL ▪Shared library – BPX.FILEATTR.SHARELIB

With z/OS V1R2, SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class will authorize the use of chmod against any file/directory.



This shows the underlying architecture for an access check. One file access is actually translated into a search access for each directory along the way before the file access check is performed.



This algorithm shows the basic flow, but omits

TRUSTED/PRIVILEGED checking for superuser

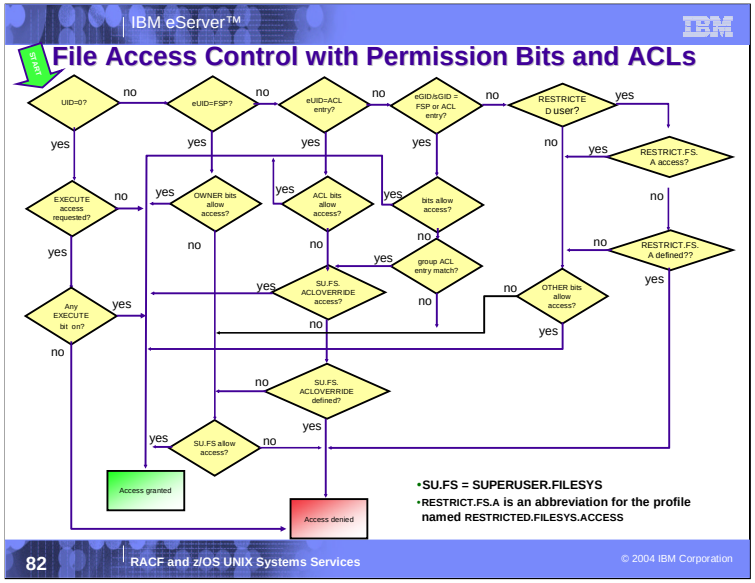
superuser exception for execute access (at least 1 execute bit must be on)

exception for RACF AUDITOR (can read and search any directory)

unauthenticated client processing (both client and server must be authorized to the file; the same algorithm is applied to both)

Note that the RESTRICTED attribute *can* apply to UNIX files (OTHER bits treated like RACF UACC)

What access is allowed in this example? r-x because one of the user's supplemental groups owns the file



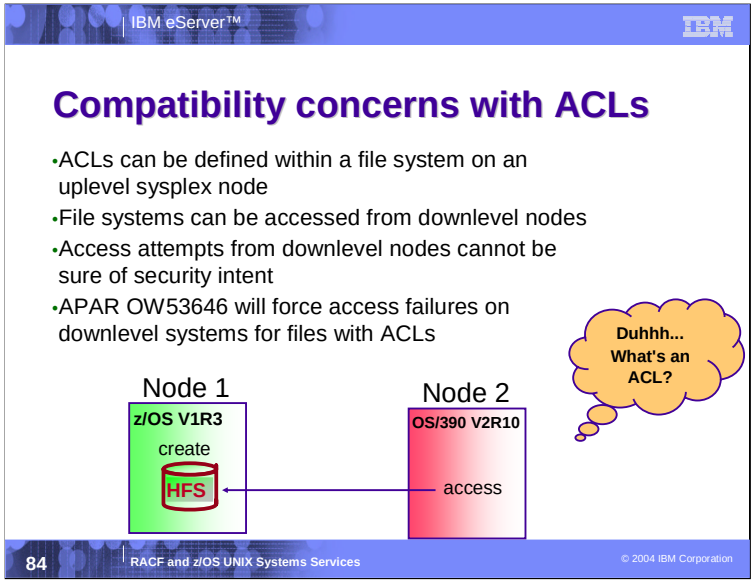
The superuser/execute check (far left of the diagram) is extended to check the ACL for any entry containing an execute bit, if no execute bit is on in the base permissions.

This diagram is still not 100% complete (missing the same stuff which is missing from the 1st flowchart).

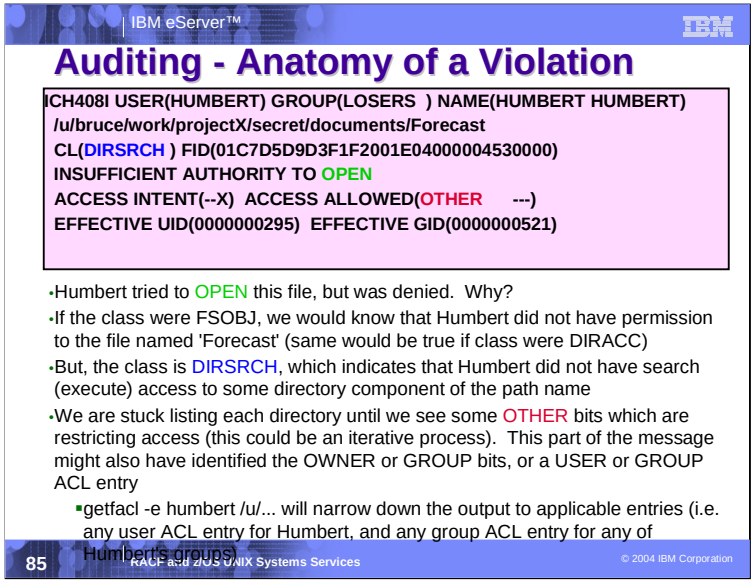
The V1R3 Security Administrator's Guide has been updated with an exhaustive (exhausting?) set of steps performed during Unix file authorization checking

UNIX File Group Ownership

- UNIX files have an owner (UID) and an owning group (GID)
- Previously owning group inherited from directory
- Can now choose
  - Assign from owner of directory, as before
  - Assign from effective GID of user creating the file
- RDEFINE UNIXPRIV FILE.GROUPOWNER.SETGID
  - Directory set-gid bit on - assign from director
  - Directory set-gid bit off - assign from user



In a SYSPLEX environment, the node on which the file system is mounted is called the 'server'. The node which is accessing a file in this filesystem is called the 'client'. The actual access check is performed by the client, based on the FSP (which contains the base permission bits), and now also the ACL, which is retrieved from disk by the server and shipped to the client. In this scenario, the server (Node 1) has all the information (i.e. FSP and ACL) necessary to make an access decision, but the client (Node 2) does not know to ask for the ACL in the first place, and certainly does not know how to apply ACL checking rules. Only the FSP is returned from Node 1.

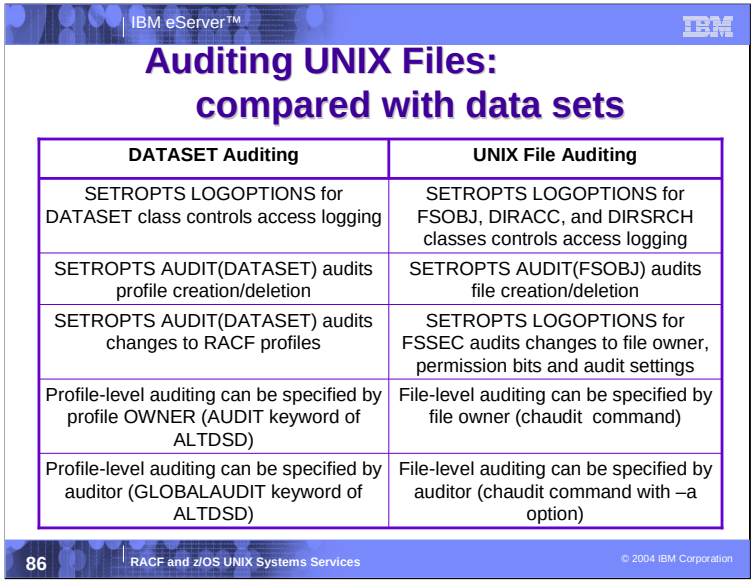


Remember, you only get violations if the access attempt was audited, which it will be by default

Note that failed attempts to change file security info (e.g. chown) will **not** be logged (FSSEC class) by default. FSSEC will be in the SETROPTS LOGOTIONS DEFAULT list, but, there are no file-level audit options which control changes to file security (they **only** apply to accesses).

The effective UID and GID are shown, in case the user has changed their identity (using a set-uid file, for example). Much confusion results otherwise.

the auditid tool from the UNIX tools and toys page can be used to match pathnames to FIDs.

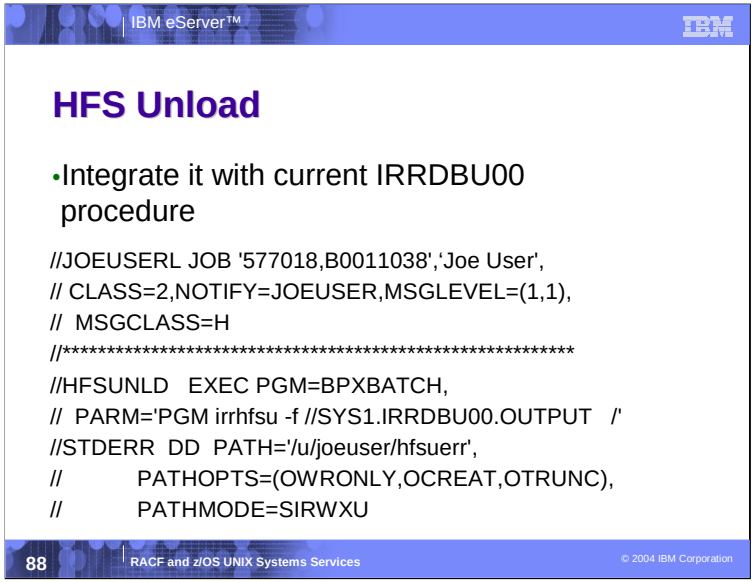


You'll find that UNIX file auditing has been designed to closely mirror the auditing that RACF does with system-wide settings (SETROPTS) and file-level audit options:

- files have owner audit settings and AUDITOR audit settings
- by default, failed accesses are logged
- SETROPTS LOGOPTIONS ALWAYS and NEVER for the FSOBJ, DIRACC, and DIRSRCH classes override file-level options
- SETROPTS LOGOPTIONS SUCCESSES and FAILURES for the FSOBJ, DIRACC, and DIRSRCH classes 'merges' with file-level options

Auditing UNIX Files: compared with data sets ...

DATASET Auditing	UNIX File Auditing
LOGOPTIONS with ALWAYS and NEVER overrides profile settings	Same for file settings
LOGOPTIONS with SUCCESSES or FAILURES merged with profile-level settings	Same for file settings
LOGOPTIONS with DEFAULT uses the profile-level settings	Same for file settings
Default profile setting is READ failures for owner options (implies UPDATE, CONTROL, and ALTER failures too), and no settings for auditor options	Default is read, write, and execute failures for owner settings (note that UNIX permissions are not hierarchical – they are separate settings for each access type)
Display profile options with LISTDSD	Display file options with ls -W



This job appends the HFS unload output to your Database unload output.