Z/OS UNIX Security

This paper provides information relating to z/OS UNIX functions and how they impact security. It can be used as a reference point and a checklist to ensure UNIX functions in z/OS 1.2 and above have security applied. This paper assumes that UNIX is already configured and able to communicate with RACF.

The *RACF Security Administrator's Guide* (RACF and z/OS UNIX chapter) and the *UNIX System Services Planning* manual (Establishing UNIX Security chapter) contain information to assist an administrator setup their UNIX security..

1. **SAMPLIB member BPXISEC1**
   This sample TSO/E CLIST is provided with z/OS and has RACF commands needed for the security setup. It is recommended that you review this CLIST and use it to setup the security environment.

2. **Define user IDs and group IDs**
   a. **UID**
   For users IDs to use UNIX the IDs must have a **UID** parameter defined. This can be done either on the RACF adduser command or the RACF alteruser command. The UID is a subparameter of the OMVS parameter.

   Example; au userid omvs(uid(xx)) or alu userid omvs(uid(xx))

   This UID should be unique for each user, although user IDs can share a UID it is not recommended because the sharing of UIDs allows each user access to all of the resources associated with the other users of the shared user ID. You might want to share a UID if the user IDs belong to the same person. To determine a unique ID, the ISPF shell can be used to generate a list of all user and then sort the list by UID. Or the administrator can keep track of assigned UIDs with their own methods, such as a flat file or data base.

   b. **HOME**
   The user ID should also have a home directory assigned which is assigned through the **HOME** subparameter. This is where the files created by the user will reside. The recommended home directory for a user is **/u** followed by the user ID, for example **/u/mary** would be the home directory for the MARY ID.

   If a home directory is partially specified (for example, **/mary**) problems may occur during processing. The home directory must be created by the security administrator before the user logs on to the system.

   Example; au user omvs(home('/u/mary') **not** au user omvs(home('/mary'))

c. **PROGRAM**

Specify an initial program for each user ID through the **PROGRAM** subparameter of the OMVS parameter. This parameter gives control to the user's program when the user logs in or invokes the OMVS command.

Example: au user omvs('program('/bin/sh')

To list a user'ss ID to view their UNIX information; lu mary omvs noracf

d. **Field-Level Access**

To allow a user to see or change OMVS fields in a RACF user profile, field-level access must be setup and users authorized to specified fields in any profile or to specified fields in the user's own profile (see *UNIX System Services Planning* for all fields that can be defined) .

Example : setr classacct(field) raclist(field) - activate class and setup for refresh ability
rdef field user.omvs.uid uacc(none) - define user fields to **FIELD** class
rdef field user.omvs.home uacc(none)
rdef field user.omvs.program uacc(none)

pe user.omvs.uid cl(field) id(&RACUID) ac(read)
pe user.omvs.home cl(field) id(&RACUID) ac(update)
pe user.omvs.program cl(field) id(mary) ac(update)

&RACUID gives all users the ability to look at the specified field in their profiles. An access of update would allow them to change this field. **So do not give update authority to the user.omvs.uid field because users could then give themselves superuser authority (UID 0).**

Specifying a specific user ID gives that person the ability to see the program fields for all users and update authority would allow them to change the field.

e. **Groups (GID)** .

Connect the user to a RACF defined group that has a OMVS GID, <u>if not connected to one the user will not be able to access UNIX resources</u>. If the group specified is not the users RACF default group you may make a RACF change so the group selected becomes the user's default group. Or the user can simply remember to specify this group when they logon and intend to perform UNIX functions. The GID is used in z/OS UNIX security checks. You can assign the same GID to multiple RACF groups, but it is not recommended since you would loose control at an individual group level. RACF groups that have the same GID assignment are treated as a single group during security checks.

Example; au mary dfltgrp(admin) name('mary doe') password(xxxxxx)
omvs(uid(97) home('u/mary') program('/bin/sh'))
tso(acctnum(xxxxxxx) proc(proc1) sysoutclass(a))

admin group must already have a GID assigned.

To list the group to check the GID; lg admin omvs noracf

For special considerations about using the RACF list-of-groups checking (GRPLIST) option for access to hierarchical file system (HFS) files and directories, see **z/OS Security Server RACF Security Administrator's Guide**.

3. **Superusers**
The number of humans (as opposed to started tasks) that have superuser authority (UID 0) should be limited to the minimum needed to perform the work and take care of the system.

There are three ways of assigning superuser authority;
a. Using the **UNIXPRIV** class profiles - allows a user to have some privileged functions
b. Using **BPX.SUPERUSER** - allows a user to request that they be given superuser
c. Assigning a UID of 0 to a user ID

Assigning a user ID a UID of 0 gives them superuser authority all the time and allows that ID to bypass all security checks in UNIX which allows the ID access to all files in the system, install products, manage processes and perform other administrative activities. Therefore it is strongly recommend that there be very few humans who have this authority. See **UNIXPRIV** and **BPX.SUPERUSER** sections for more information.

Sometimes it may be appropriate to assign a UID of 0 for a specific user ID. If this must be done it is recommended that the person have two user IDs. One ID associated with a UID of 0 that will be used when performing system maintenance, etc. And one user ID that has a UID other then zero which the user will use when performing all other regular activities.

4. **UNIXPRIV**
You can define profiles in the **UNIXPRIV class** to grant RACF authorization for certain z/OS UNIX privileges. By using this RACF class, you can specifically grant certain superuser privileges to users who do not have superuser authority.

Resource name in the UNIXPRIV class are associated with z/OS UNIX privileges. You define profiles in the UNIXPRIV class protecting those resources in order to use RACF authorization to grant access to those privileges. The UNIXPRIV class must be active and the SETROPTS RACLIST must be in effect for the UNIXPRIV class.

The list of resource names available in the UNIXPRIV class, the UNIX privilege associated with this resource and the authority required to grant the privilege are all listed in the *UNIX System Services Planning* manual (see Establishing UNIX Security chapter).

For example if a user needs to be able to mount and unmount file systems the administrator can define SUPERUSER.FILESYS.MOUNT to the UNIXPRIV class and grant the user ID READ authority. The user can now perform this task but it does not have access to all the other superuser authority.

Example: setr classact (unixpriv) - activate the class in RACF
setr raclist(unixpriv) - set RACLIST processing
rdef unixpriv superuser.filesys.mount uacc(none) - define entry
pe superuser.filesys.mount cl(unixpriv) id(userids or groups) ac(read)
setr raclist(unixpriv) refresh

5. **BPX.SUPERUSER**
To use this method for controlling superuser authority you must define **BPX.SUPERUSER** to the RACF class **FACILITY** and them permit the users that need to have superuser authority. When users need to use the superuser authority they can switch to superuser mode using the **su** command or the "Enable superuser mode (SU) option in the ISPF shell.

Example: setr classact(facility) - activate the class in RACF
setr raclist(facility) - set RACLIST processing
pe bpx.superuser cl(facility) id(xxxx) ac(read) - grant ID access **or**
pe bpx.superuser cl(facility) id(group) ac(read) - permit a group access
setr raclist(facility) refresh

pe bpx.superuser cl(facility) id(mary) delete - remove user ID when superuser authority is no longer needed

Each user you define to this resource should have a unique UID associated with their RACF ID. The same applies to each person connect to any group that you might grant superuser authority (such as a system programmers group). See "**Defining user IDs and group IDs**" for more information on this subject. For further details see the *UNIX System Services Planning* manual, chapter on Establishing UNIX Security.

6. **Protected User IDs**
User IDs can be defined as protected user IDs by specifying NOPASSWORD on the ID. These Ids are used for started procedures associated with z/OS UNIX, such as the kernel, the initializations started procedure, etc..

NOPASSWORD creates a protected user ID that can not be used to log on to the system nor can it be revoked by incorrect password attempts. Using NOPASSWORD also prevents the ID from being used with TSO which prevents a user logon from interfering with the OMVS user ID.

7. **Setting UNIX user limits**
   You can set certain UNIX user limits and control the amount of resources these users consume. The resource limits for the majority of UNIX users are specified in the BPXPRMxx parmlib member. These limits apply to all users except those with a UID of 0. Sometimes there are certain users that need to exceed the limitations imposed by BPXPRMxx. Rather then give out superuser authority so they can bypass the limitation you can individually set higher limits for these users. The limits are stored in the OMVS segment of the user profile. Example of limits overrides would be for maximum CPU time, or address space size. The UNIX System Services Planning manual list all the limits you can override.

8. **Protecting file system resources**
   HFS files and directories can be protected by permission bits that are stored within the file system. These bits allow read, write or search authority for a directory and read, write or execute for a file. There are three sets of bits; owner, owning group, and everyone else, so separate authorities can be specified for each group. The UNIX command **chmod** can be used to change individual bits without affecting the other bits. The owner, superuser can change these settings, or authority could be granted through UNIXPRIV. For information on setting these bits and granting/changing this authority reference the *UNIX System Services Planning* manual.

   There also exist access control lists (**ACLs**) that are use in conjunction with the permission bits. ACLs allow access control for files and directories by individual UIDs and GIDs. The checking for ACLs is done by RACF but they are administered through UNIX commands**.**
   ACLs are created, modified and deleted using the **setfacl** UNIX command and they are displayed using the **getfacl** shell command. ACLs are created and checked by RACF, if you are using a security products other then RACF you need to check that documentation to see if it supports ACLs.

   There are two types of ACLs; base ACL are the same as permission bits and they can be changed using chmod or setfacl; extended ACL are entries for individual users or group and they are managed via setfacl. The file owner, superuser or someone authorized through UNIXPRIV can change these ACLs.

   To use ACLs the RACF class FSSEC must be active. If this class is not active the standard permission bits are used to govern access to the file/directory.

Example1; file /etc/inetd.conf, user ID Joe and group admins require read and write authority

setr classact(FSSEC) raclist(FSSEC)
setfacl -m user:joe:rw-,group:admits:rw- /etc/inetd.conf
(-m modifies or adds ACLs if they do not exist)

```
getfacl /etc/inetd.conf  - display file
#file: /etc/inetd.conf
#owner: BPXROOT
#group: SYS1
user::rw-
group::r--
other::r--
user:JOE:rw-
Group:ADMINS:rw-
```

Example 2; file /etc/inetd.conf, User ID Joe and group Admins require read and write authority and set base permission bits to prevent access by anyone other than the owner

setfacl -s user::rw-,group::---,other::---,
user:joe:rw-,group:admits:rw- /etc/inetd.conf
(-s replaces the contents of an ACL with the entries specified on the command line. It requires that base permissions be specified.)

Reference the UNIX Planning manual and the UNIX Command manual for additional details on ACL and how to use the command.

9. **Auditing**

Audit classes DIRSRCH, DIRACC, FSOBJ, FSSEC, PROCESS, PROACT are set via RACF LOGOPTIONS. No users or groups are defined to these classes. Activating them via CLASSACT has no effect. These classes control audit in z/OS UNIX.

10. Special UNIX Privileges without being a SuperUser

Grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. UNIXPRIV profiles identified and described in USS System Planning. Access required to perform function is shown in parentheses. Where no access is shown the access is READ.

a. **CHOWN.UNRESTRICTED**

Allows all users to use the chown command to transfer ownership of their own files. (NONE)

b. **RESTRICTED.FILESYS.ACCESS**
Specifies that RESTRICTED users cannot gain file access by virtue of the 'other ' permission bits. Can be overridden for a specific user/group. (NONE)

c. **SUPERUSER.FILESYS.ACLOVERRIDE**
Specifies that ACL contents override the access that was granted by SUPERUSER.FILESYS. (NONE) Can be overridden for a specific user/group. (User/group must have the same access that would be required to SUPERUSER.FILESYS while accessing the file)

d. **SUPERUSER.FILESYS**
Allows user to read any local file, and to read or search any local directory. (READ)
Allows user to write to any local file, and includes privileges of READ access. (WRITE)
Allows user to write to any local directory, and includes privileges of UPDATE access. (CONTROL or higher)

e. **SUPERUSER.FILESYS.CHANGEPERMS**
Allows users to use the chmod command to change the permission bits of any file and to use the setfacl command to manage access control lists for any file. (READ)

f. **SUPERUSER.FILESYS.CHOWN**
Allows user to use the chown command to change ownership of any file. (READ)

g. **SUPERUSER.FILESYS.PFSCTL**
Allows user to use the pfsctl() callable service. (READ)

h. **SUPERUSER.FILESYS.VREGISTER**
Allows a server to use the vreg() callable service to register as a VFS file server. (READ)

i. **SUPERUSER.IPC.RMID**
Allows user to issue the ipcrm command to release IPC resources. (READ)

j. **SUPERUSER.PROCESS.GETPSENT**
Allows user to use the w_getpsent() callable service to receive data for any process. (READ)

k. **SUPERUSER.PROCESS.KILL**
Allows user to use the kill() callable service to send signals to any process. (READ)

l. **SUPERUSER.SETPRIORITY**
Allows user to increase own priority. (READ)

m. **SUPERUSER.SETPRIORITY**
Allows user to increase own priority. (READ)

Z/OS UNIX Security

11. **BPX.* FACILITY class profiles**
    Reference the *UNIX System Services Planning* manual and the *Security Server RACF Security Administrator's Guide* for additional details on BPX profiles.

    a. **BPX.DAEMON**
    In z/OS it is recommended that the administrator define **BPX.DAEMON** to the RACF class FACILITY. Doing this means that the system has z/OS UNIX security and you have more control over superusers then systems that do not run with this definition.

    This serves two functions; any superuser permitted to this profile has the daemon authority to change MVS identities without knowing the target user ID's password (target ID must have an OMVS segment defined).

    If BPX.DAEMON is not defined then all superusers have deamon authority. To limit which superusers have daemon authority, define this profile and permit only selected superusers to it.

    The second function BPX.DAEMON provides ensures that any program loaded into an address space that requires deamon level authority must be defined to program control. z/OS UNIX has the following deamons; inetd (network daemon), rlogind (remote login), cron(clock daemon), uucpd (UUCP daemon). SYSLOG daemon is shipped with Communications Services and is documented in their library.

    You must activate program control in RACF and files that require program control must have the program control flag set. (UNIX Planning and RACF manuals explain how to perform these tasks).

    Example: setr classact(facility) raclist(facility) - make sure FACILITY class is active
    rdef facility bpx.daemon uacc(none)
    pe bpx.daemon cl(facility) id(racfadm) ac(read) - administrator needs access to restart daemons
    pe bpx.daemon cl(facility) id(*omvskern*) ac(read) - UNIX kernel needs access

    b. **BPX.DAEMON.HFSCTL**
    Controls which users with daemon authority are allowed to load uncontrolled programs from MVS libraries into their address space.

    c. **BPX.DEBUG**
    Controls users who can use ptrace (via **dbx)** to debug programs that run with APF authority or with BPX.SERVER authority**.**

d. **BPX.DEFAULT.USER**
Identifies the user ID and group name to be used when setting up default OMVS segments.

e. **BPX.FILEATTR.APF**
Controls which users are allowed to set the program control attribute in an HFS files . Programs marked with this attribute can execute in server address spaces that run with a high level of authority**.**

f. **BPX.FILEATTR.PROGCTL**
Controls which users are allowed to set the program control attribute in an HFS file. Programs marked with this attribute can execute in server address spaces that run with a high level of authority.

g. **BPX.FILEATTR.SHARELIB**
Indicates that extra privilege is required when setting the shared library extended attribute via the chattr() callable service. This prevents the shared library region from being misused.

h. **BPX.JOBNAME**
Control which users are allowed to set their own job names by using the BPX_JOBNAME environment variable or the inheritance structure on spawn. Users with READ or higher permissions to his profile can define their own job names

i. **BPX.SAFFASTPATH**
Enables faster security checks for file system so it improves performance but prevents auditing of successful events.

j. **BPX.SMF**
Checks if the caller attempting to cut an SMF record is allowed to write an SMF record.

k. **BPX.SERVER**
Restricts the use of the pthread_security_np service. Users requiring this service must be defined to BPX.SERVER with read or write authority. It deletes or creates the security environment of the caller's thread. It also restricts the use of the BPX1ACK service, which determines access authority to z/OS resources

l. **BPX.SRV.*userid***
Allows users to change their UID if they have access to BPX.SRV.userid, where uuuuuuuu is the MVS user ID associated with the target UID. BPX.SRV.userid is a RACF SURROGAT FACILITY class profile.

m. **BPX.STOR.SWAP**
Controls which users can mark address spaces nonswappable.

n. **BPX.SUPERUSER**
Allows users to switch to superuser authority. See SUPERUSER, item 3 and 5 above.

o. **BPX.WLMSERVER**
Controls access to the WLM server functions.