Breaking Up on the Rocks: Why 777 Could Destroy Your Digital Security

David Schuchart

Product Owner, Broadcom Mainframe Security

david.schuchart@broadcom.com

Session Agenda

- What is Unix System Services (USS) in 60 seconds
- Why should you care about USS security
- Permission bits – the most basic USS security
- How can we do better than permission bits – ACLs and the sticky bit
- ESM-based USS controls to think about

z/OS UNIX System Services (USS)

Component of z/OS operating system optimized for mainframe architecture
- Official name: z/OS UNIX System Services
- Other names: z/OS UNIX, USS, OMVS

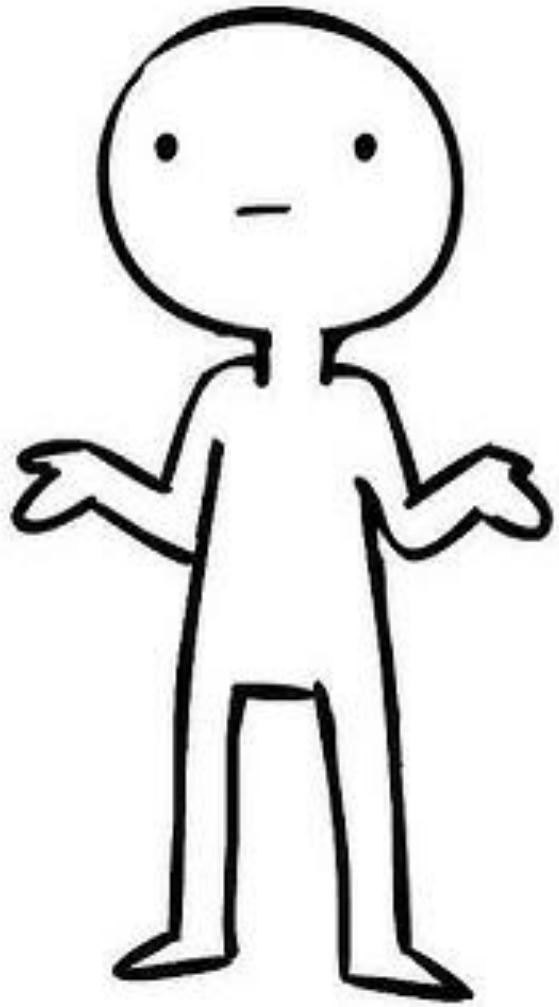
Certified UNIX system
- Predecessor of z/OS UNIX was OpenEditionMVS
- Initial release: 1998
- Latest release: 2023 Version 3.1 (V3R1)

File System
- Hierarchical
- Based on underlying VSAM data sets

OMVS – z/OS UNIX Shell environment, can also use 3.17 or ISHELL

Why should you care about USS security?

- Many mainframe solutions have a USS component, and USS is an integral subsystem to z/OS
- You can APF authorize programs in USS
- Shared UIDs/GIDs mean shared identities
- Toxic access combinations can allow impersonation of any user
- If nothing else, an auditor will eventually ask



PERMISSION BITS – THE MOST BASIC FORM OF USS SECURITY

Permission Bits – File level

-rwxrwxrwx (-uuugggooo)

- uuu – Owner permissions
  - First character: read or print the file
  - Second character: change or delete from the contents of the file
  - Third character: run the file – refers to executable file type
- ggg – Group permissions
  - First character: read or print the file
  - Second character: change or delete from the contents of the file
  - Third character: run the file – refers to executable file type
- ooo – Others permissions
  - First character: read or print the file
  - Second character: change or delete from the contents of the file
  - Third character: run the file – refers to executable file type

Permission Bits – Directory level

drwxrwxrwx (duuugggooo)

- uuu – Owner permissions
  - First character: list the contents of the directory
  - Second character: change, add or delete the contents of the directory
  - Third character: enter/traverse the directory, access directory contents
- ggg – Group permissions
  - First character: list the contents of the directory
  - Second character: change, add or delete the contents of the directory
  - Third character: enter/traverse the directory, access directory contents
- ooo – Others permissions
  - First character: list the contents of the directory
  - Second character: change, add or delete the contents of the directory
  - Third character: enter/traverse the directory, access directory contents

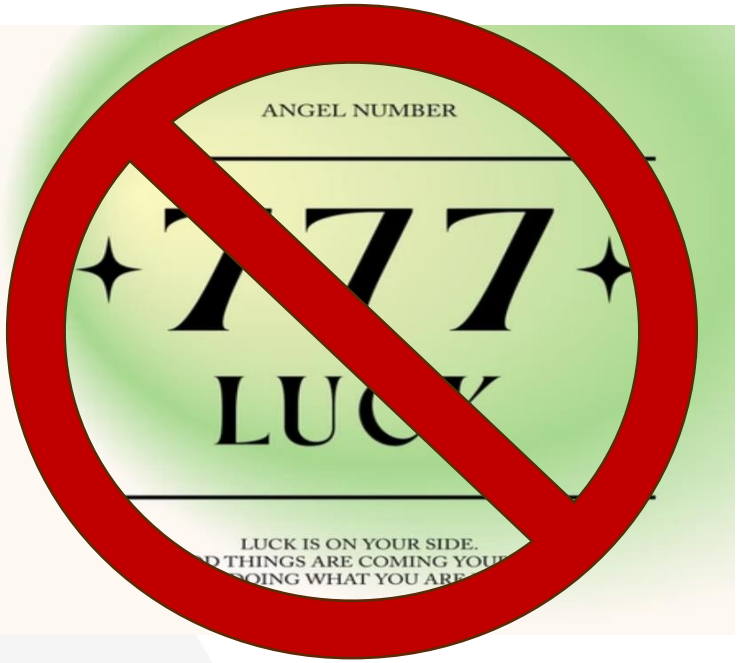
Permission Bits – Octal values

- There are three sets of permissions for each level of ownership: owner class, group class, and others class.
- The permissions can be interpreted as octal values: read (4), write (2), execute (1). Sum up the three numbers for each level of ownership to calculate the octal value.

Permission bits	Octal Notation
-rwxrwxr-x	
-421421401	775
drwxr-x---	
d421401000	750
-rw-r-----	
-420400000	640

Check for 777- it's bad!

- As 777 is "anybody can do anything", it's the worst permission bit combo to have set
  - Review to make sure anything that is 777 absolutely requires it (I'd argue this should never be the case)
- Find usage of 777 permissions by issuing the following Unix command:
  - find *where_to_start_search* –type *file_or_dir* –perm 777
  - find / -type f -perm 777 (find all files with 777 permissions starting from root)
- Use caution on where you start the search
  - Starting from root can take a long time and generate a lot of output
- Can redirect the output to a file for easier review or run the command in BPXBATCH
  - find / -type f -perm 777 > some_file.txt



HOW CAN WE DO BETTER THAN PERMISSION BITS – ACLS AND THE STICKY BIT

UNIX Access Control Lists (ACLs)

- Owner/group and permission bits are “basic” USS security, but don’t allow for much granularity
- ACLs used in conjunction with permission bits to provide more control
  - Base ACL - permission bits
  - Extended ACL - additional access controls, could be set for an individual user or a group
- Implementation varies per ESM:
  - HFSACL (ACF2/TSS) / FSSEC (RACF)
    - Turn on to enable ACL checking in USS
    - There is an FSSEC option in ACF/TSS, but it is for SMF record cutting
- 3 kinds of ACLs
  - Access ACLs – provide access control to files/directories
  - File default ACLs – set the default ACL on files in a directory
  - Directory default ACLs – set the default ACL on subdirectories

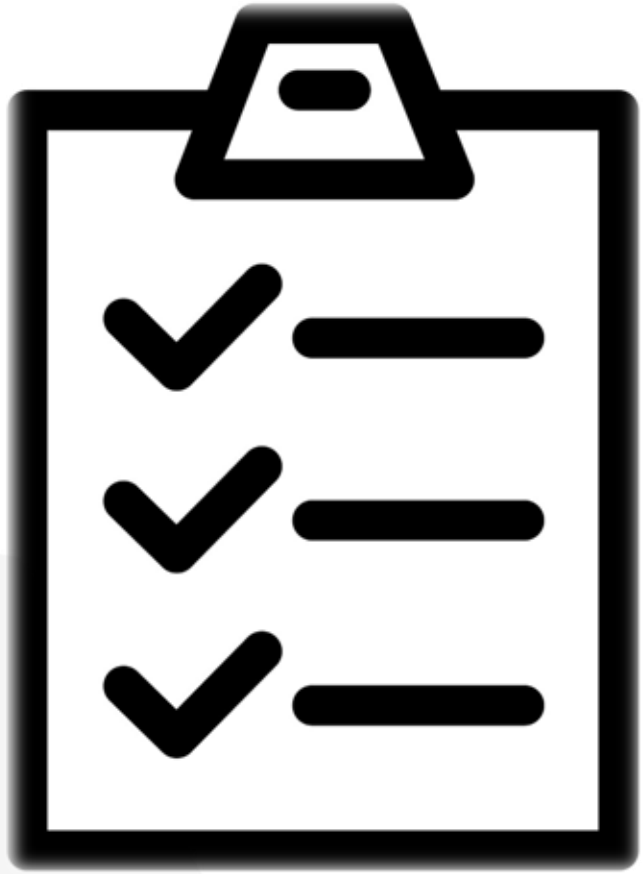
Managing ACLs

- View the ACL
  - getfacl – “get file access control list”
- Create or modify the ACL
  - setfacl – “set file access control list”
- Examples:
  - Modify the ACL (or create if it doesn’t exist) for /some/file to allow user123 read and write, allow group123 read and write

```
setfacl -m user:user123:rw-,group:group123:rw- /some/file
```

  - View the ACL for /some/file

```
getfacl /some/file
    #file: /some/file
    #owner: SOMEOWNER
    #group: SOMEGROUP
    user::rw-
    group::r-
    other::r-
    user:USER123:rw-
    group:GROUP123:rw-
```



Permission Bits – Sticky Bit ‘t’

```
$ cd /u/user02/myapp/bin
$ ls -l apprun
-rwxr-x--t   2 user02   devgrp         475 Oct 18 2017 apprun
```

- Set at the directory level
- Controls permission to remove or rename files or subdirectories
  - Useful for those fringe situations where 777 is “kind of” OK (think: /tmp)
- When on, the user can remove or rename only if one of below is valid
  - Owns the file or subdirectory
  - Owns directory
  - Has superuser authority
- Sticky bit also exists for files, but is more performance-tuning related

ESM-BASED USS CONTROLS TO THINK ABOUT

ESM Settings that Affect USS Security

- Reminder: turn HFSACL on and use ACLs!
- HFSSEC (ACF2/TSS)
  - Bypasses “normal” USS security access validation and instead checks ACF2/TSS rules
- FSACCESS (ACF2/TSS/RACF)
  - Controls access at the filesystem level
  - Another layer of control, but very coarse-grained

SURROGAT and BPX.SRV.*userid*

- In CLASS – SURROGAT (generally scrutinize any surrogate access)
- Allows users to change their UID if they have access to BPX.SRV.*userid*, where *userid* is the zOS user ID associated with the target UID.
  - There should be little/no usage of this due to passwordless identity switching. Any usage/access should be well-understood and documented.
  - Particularly problematic if able to su –s to UID(0)

```
------------------------------------------------------
 *    Welcome to ACBD UNIX System Services            *
------------------------------------------------------
$ su -s userid
```

FACILITY CLASS BPX resources

- BPX.SUPERUSER
  - Allows su to UID(0)
- BPX.SUPERUSER + BPX.DAEMON
  - UID(0) + BPX.DAEMON allows for change of zOS identity without knowing the target user's password (though the target ID does need an OMVS segment)

```
badnews.sh

   echo '_BPX_USERID=target_userid tsocmd "'"$1"'"' | su
./badnews.sh some_tso_command
```

  - Will issue `some_tso_command` as if using zOS identity `target_userid`
- BPX.FILEATTR.*
  - Allows setting of extended file attributes, pay close attention to this being masked and/or access to BPX.FILEATTR.APF, which allows for APF authorization of AC=1 linked modules in USS

The UNIXPRIV class

- Allows for granular access to superuser-like functions
- CHOWN.UNRESTRICTED
  - Allows self-reassignment of ownership via chown command
- SUPERUSER.FILESYS.*
  - Several extended resources to review that can control users' ability to:
    - Read/write any local file/directory
    - Change ownership of any file
    - Change permission bits of any file
    - Mount/unmount file systems
- SUPERUSER.PROCESS.KILL
  - Allows use of the kill() callable service (end a process)

Recap – things to do

- Consider USS security just as important as z/OS/ESM security
- Review permission bits in your USS environment, fix any instances of 777 found
- Consider turning on and using ACLs for more granular access
- Review ESM-based controls/rules that affect USS security operations, particular the resources in the FACILITY (BPX.*) and UNIXPRIV classes
- Reach out for additional help

Join a Mainframe Technical Exchange

SHARE
EDUCATE • NETWORK • INFLUENCE

Apr. 21-23, 2026	June 23-25, 2026	Oct. 20-22, 2026
European MTE Prague, Czech Republic	Virtual MTE Held Virtually	North American MTE Plano, TX

- Network with peers and Mainframe technical experts
- Participate in technical how-to sessions and hands-on workshops
- No registration fee!

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation

THANK YOU!

Contact:

David Schuchart

Product Owner, Mainframe Security

david.schuchart@broadcom.com