z/OS UNIX Security Overview

New York, Tampa Bay, Dallas, and Raleigh RACF User Group

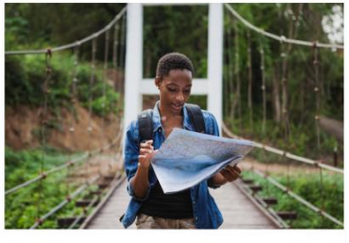
May 15, 2024

Bruce R. Wells

brwells@us.ibm.com

Navigating the presentation

Navigating the documentation

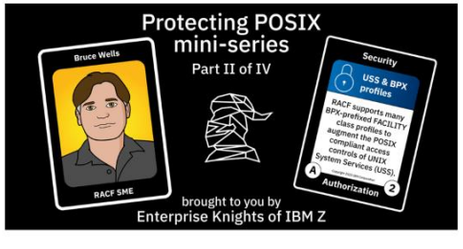


My UNIX brain-dump of record

1. z/OS UNIX System Services File System Security
2. z/OS UNIX System Services File Security
3. z/OS UNIX System Services Users and Groups
4. The UNIX superuser

UNIX file *system* security

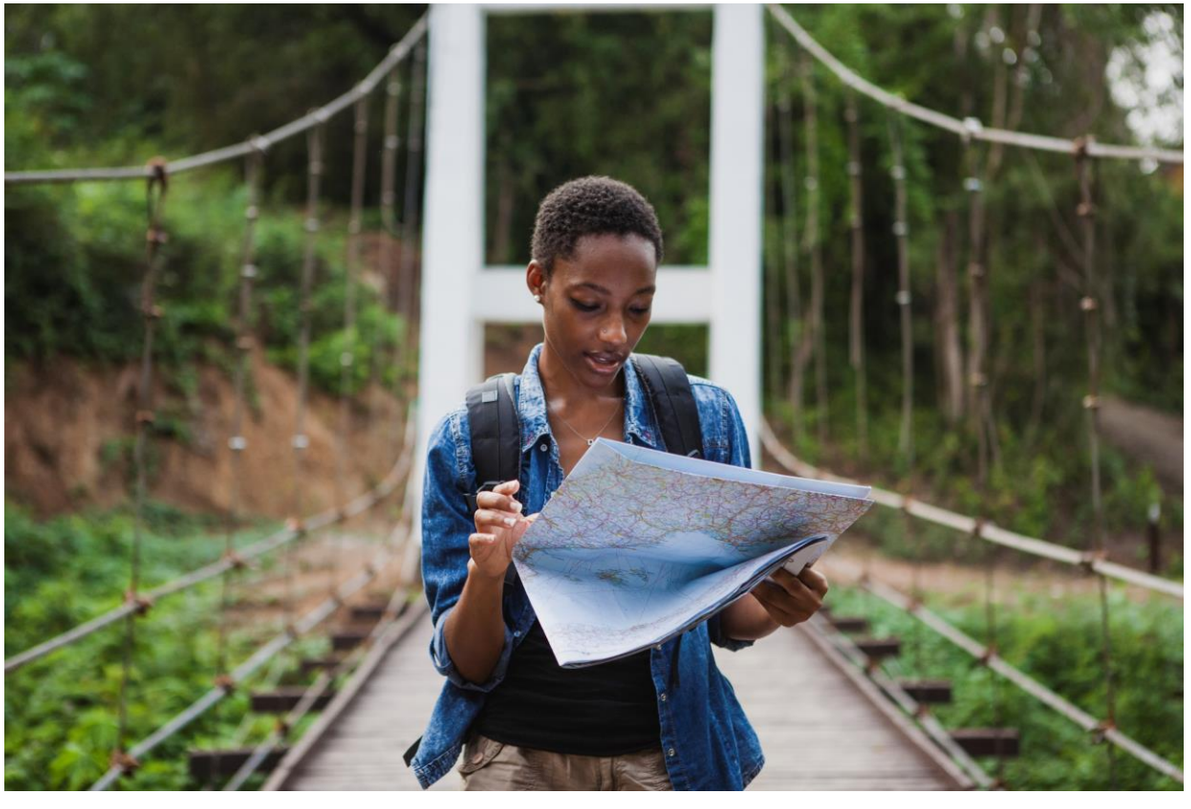
UNIX *file* security



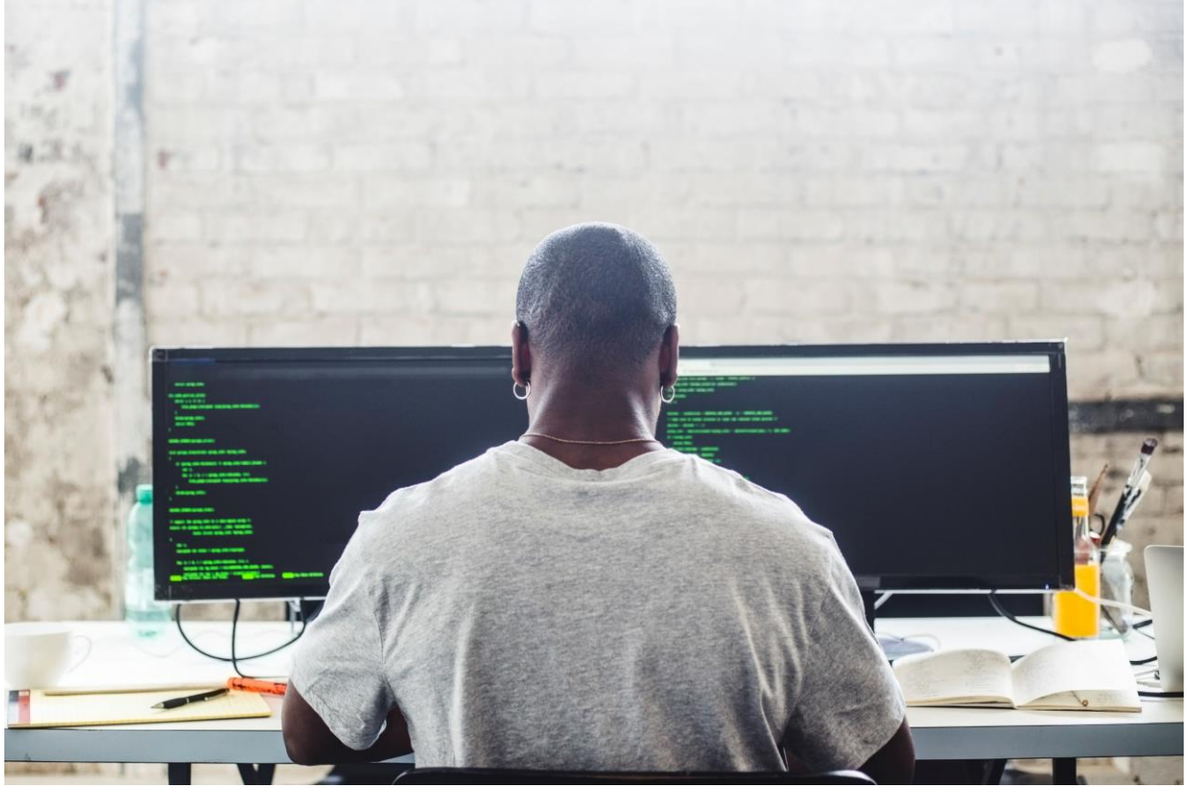
UNIX users and groups

UNIX superusers

Navigating the documentation



For security administrators

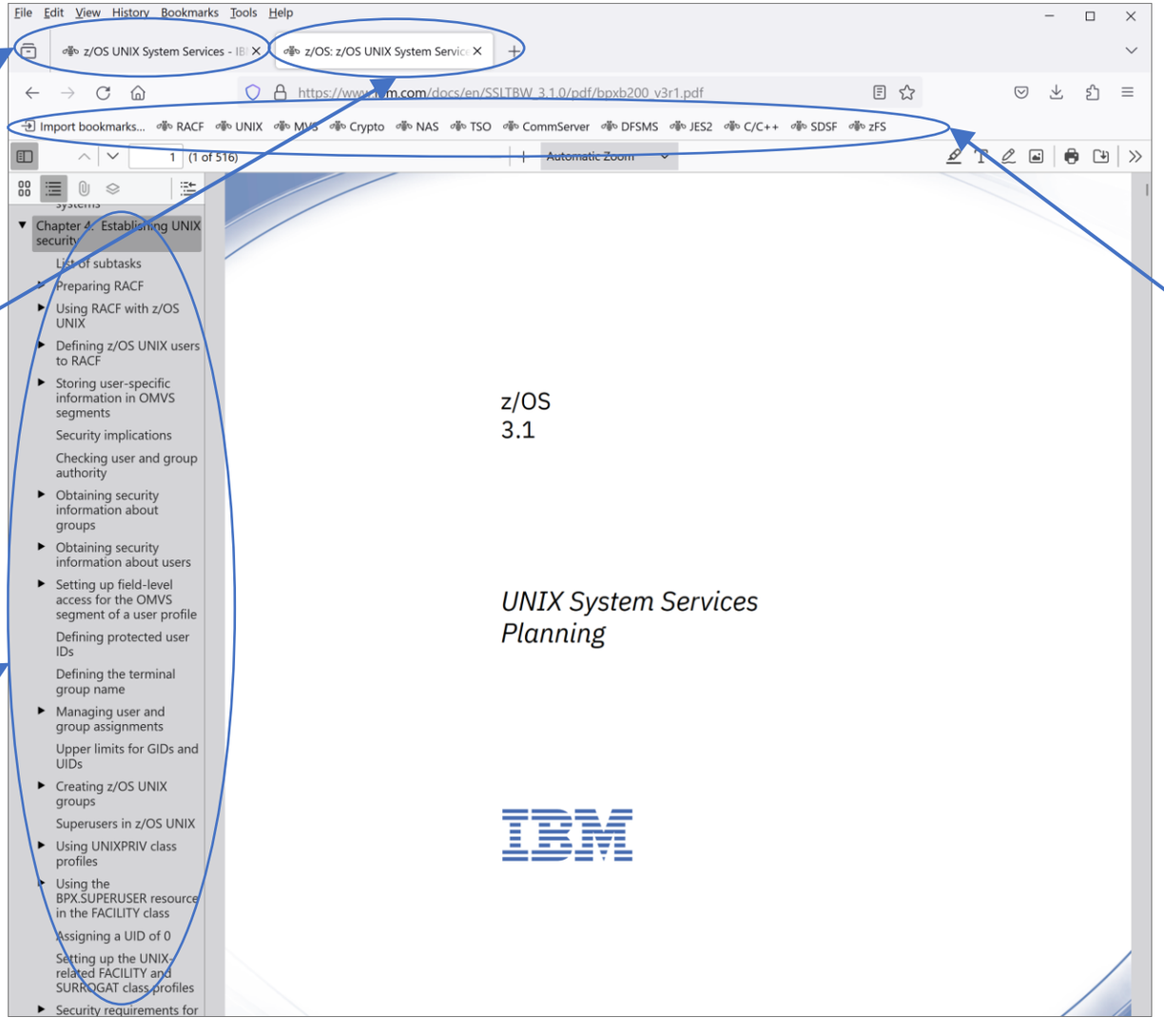


UNIX System Services Planning: Security chapter

The bookshelf in which the displayed book resides

The displayed book

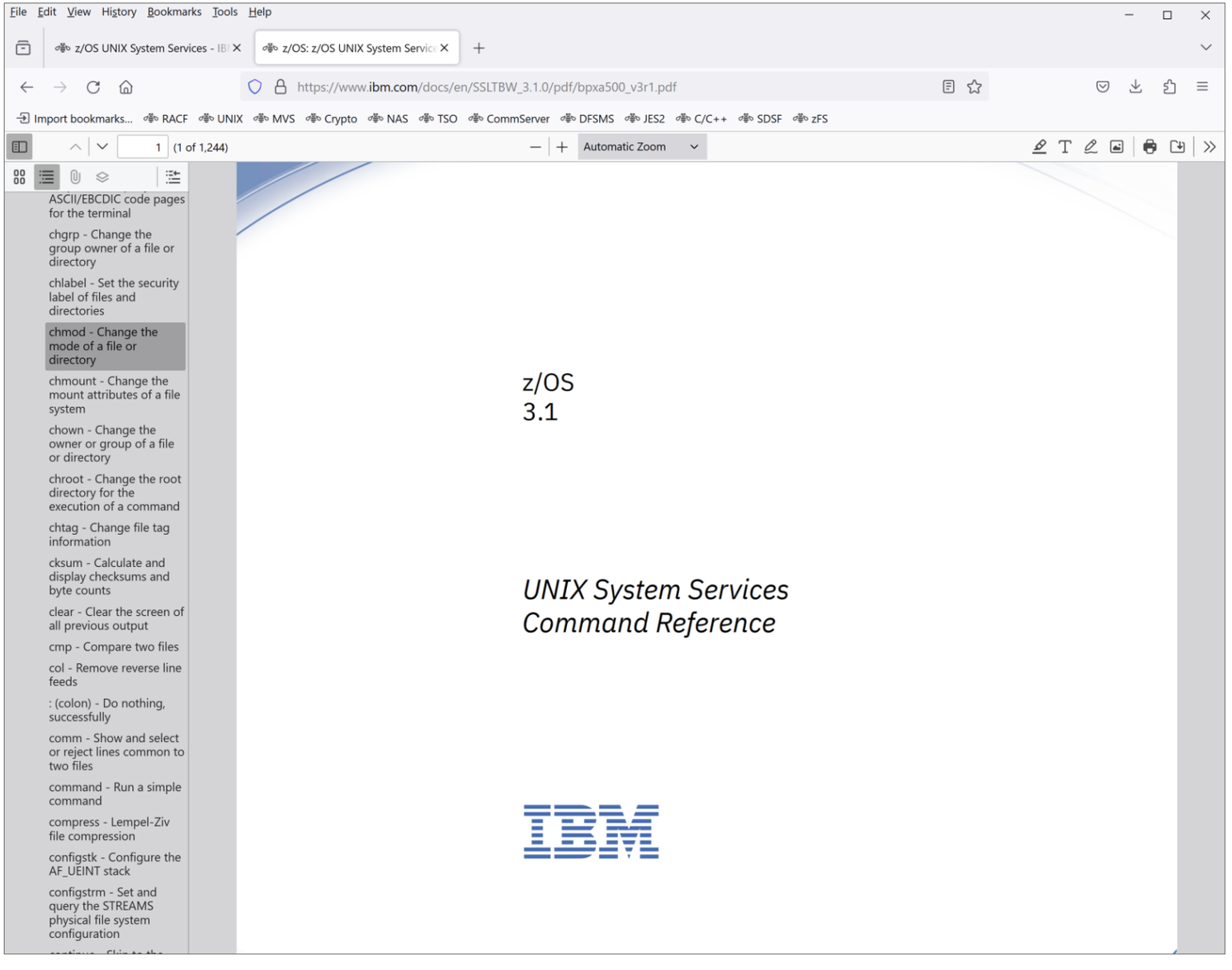
The relevant content



The bookshelves I reference the most are Firefox bookmarks.

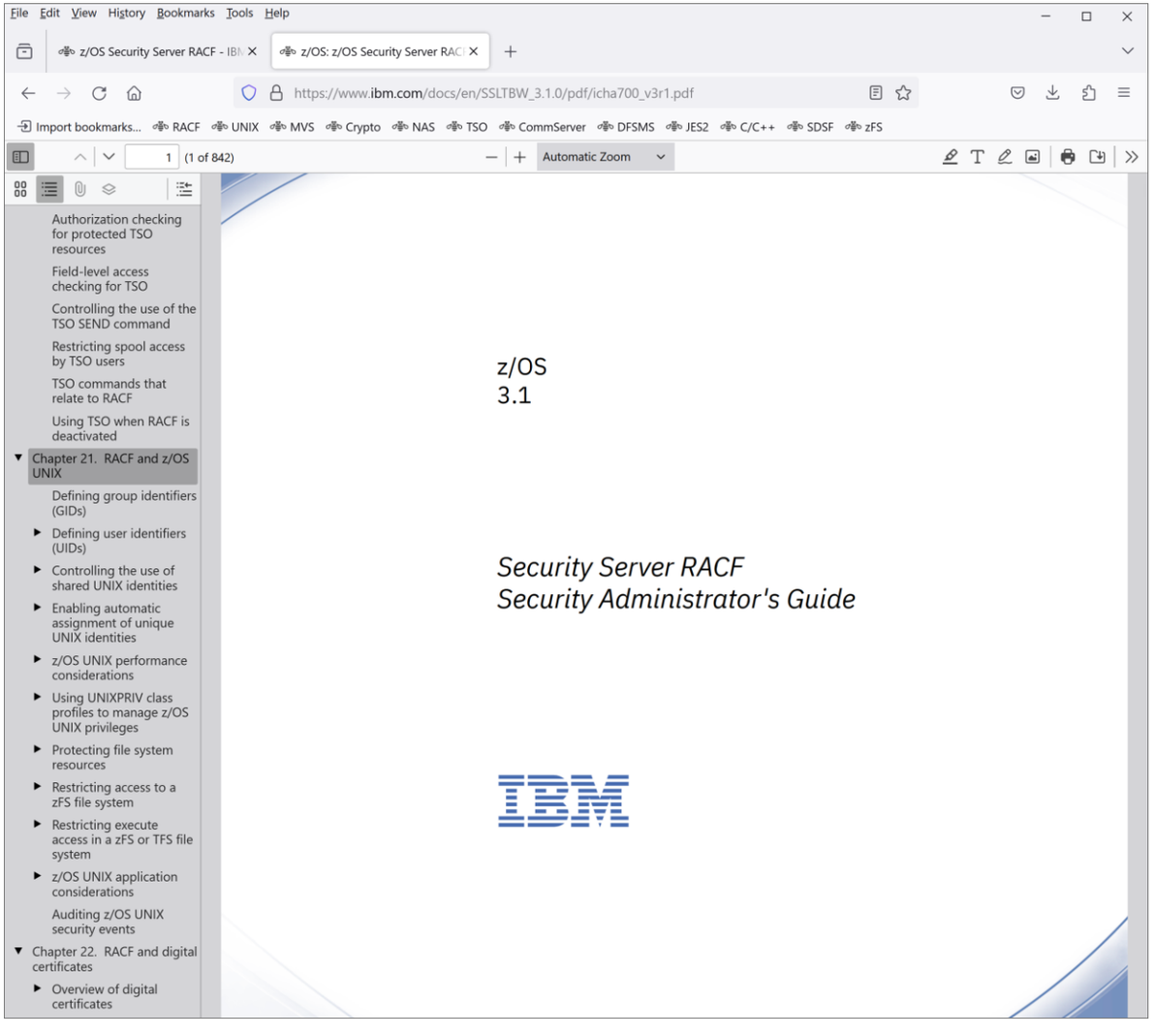
This book/chapter is the single most comprehensive source of information

USS Command Reference: security cmds



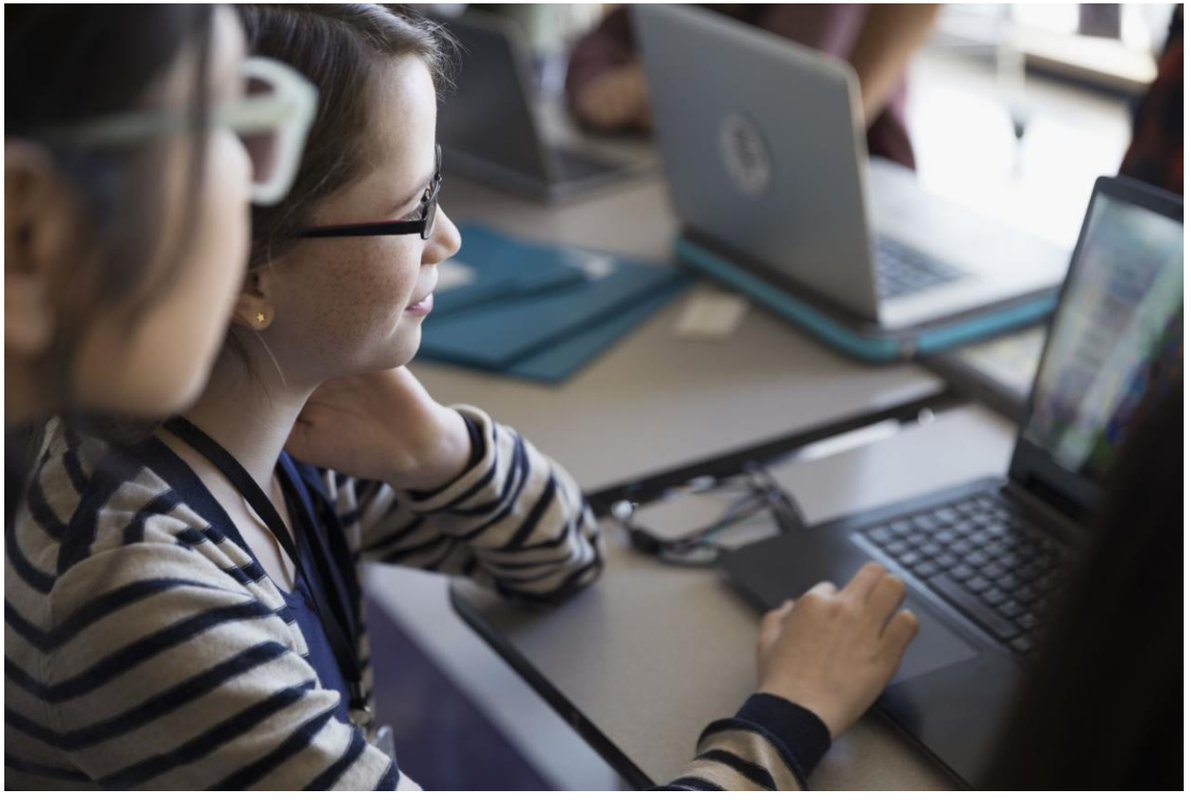
The shell commands you may need to issue, stating the authorization required to do so

RACF Security Administrator's Guide: UNIX chapter

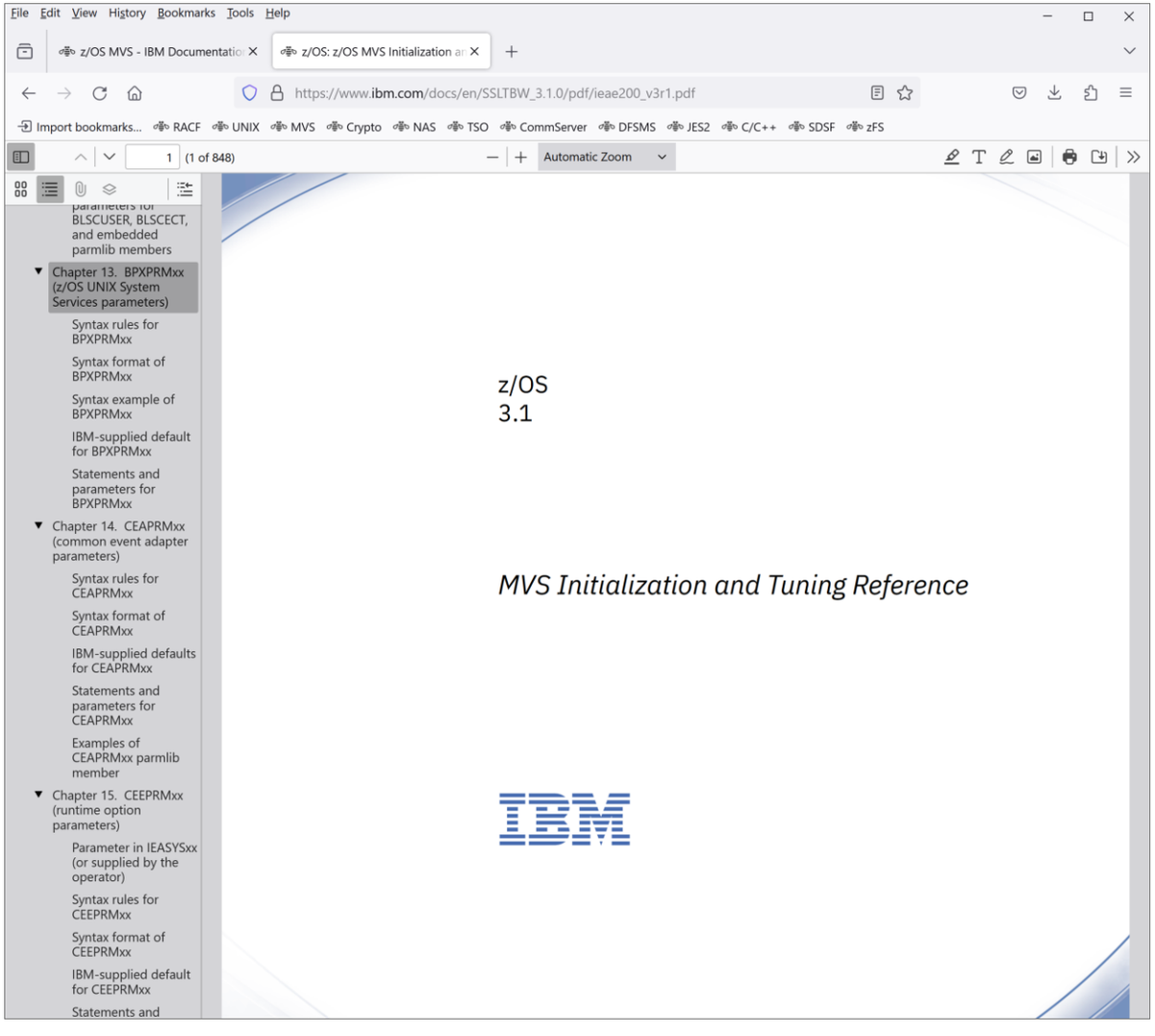


Serves as a good cross-check with UNIX Planning

For system programmers

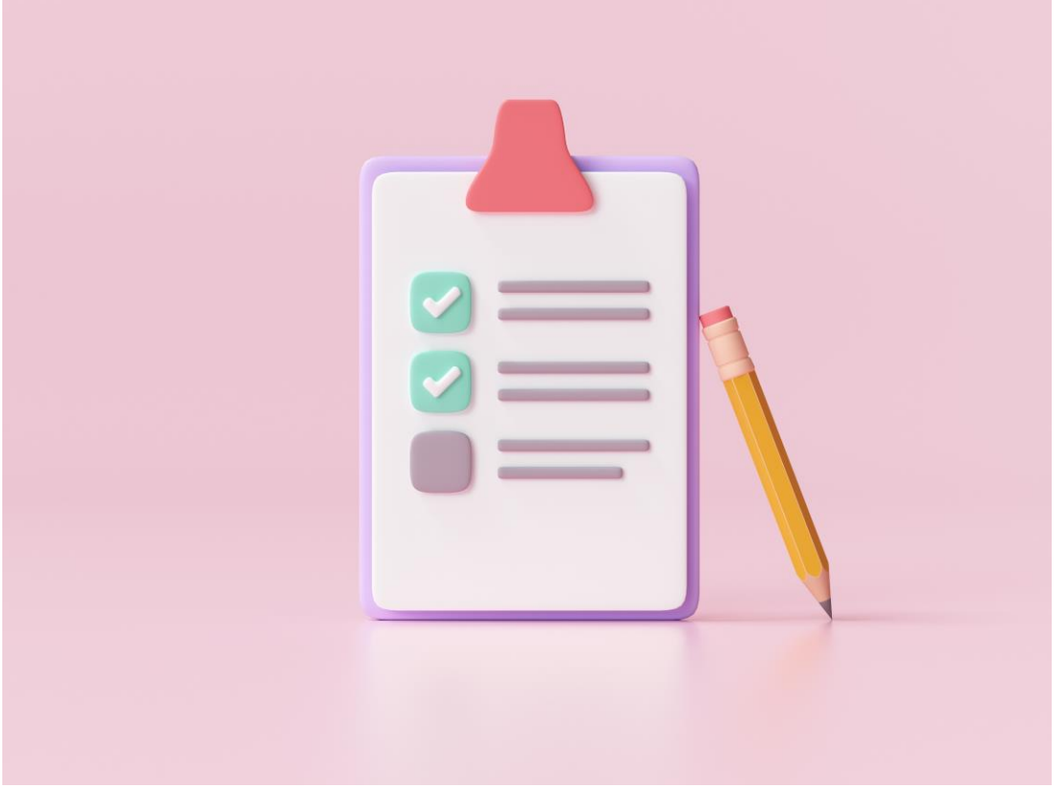


MVS Initialization and Tuning Ref: BPXPRMxx

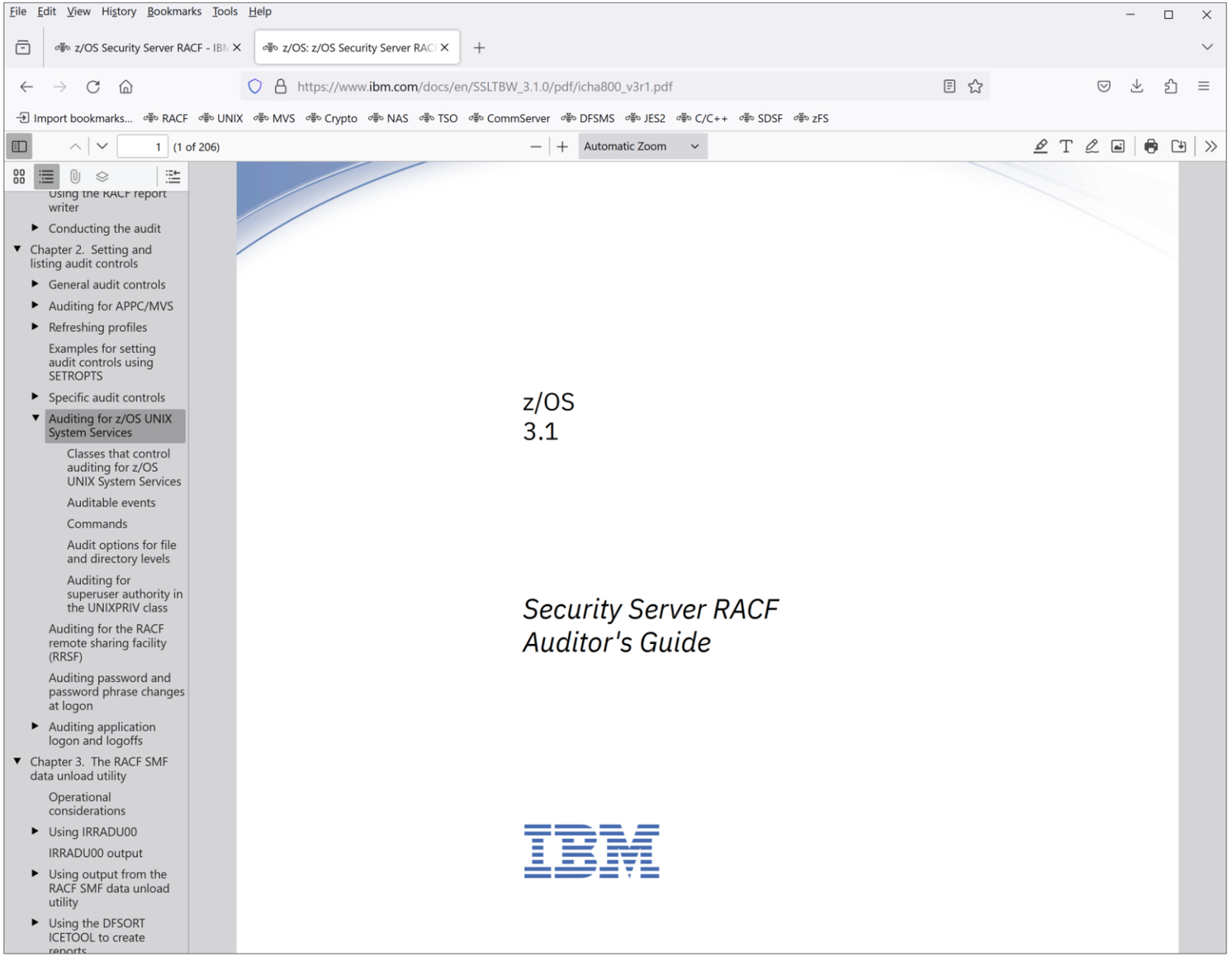


BPXPRMxx establishes file system structure at IPL, and contains many more options such as default system resource limits

For auditors

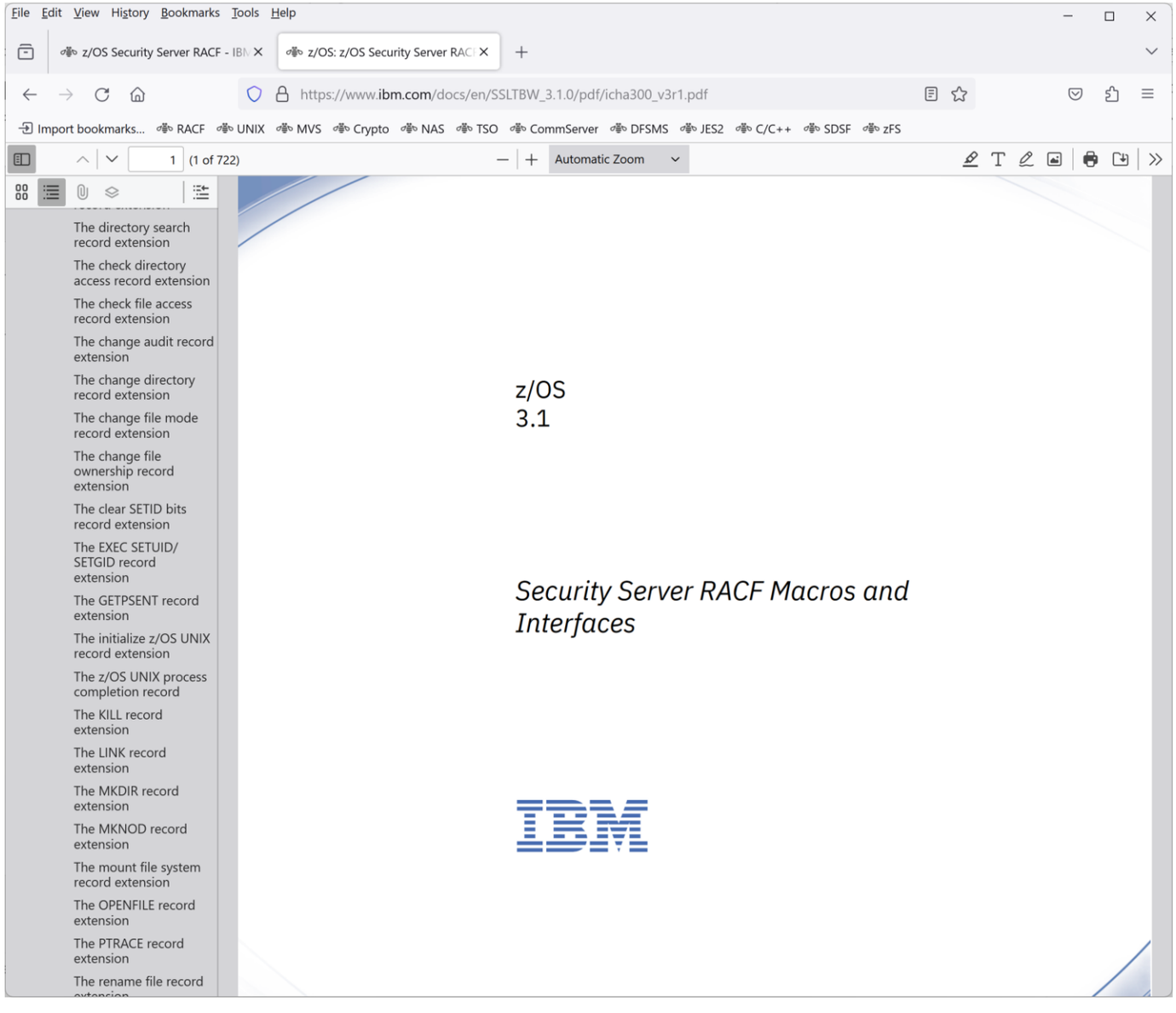


RACF Auditor's Guide: UNIX section



Describes the UNIX logging classes, which ones are controlled with SETROPTS AUDIT vs. LOGOPTIONS, and which UNIX functions are included in each

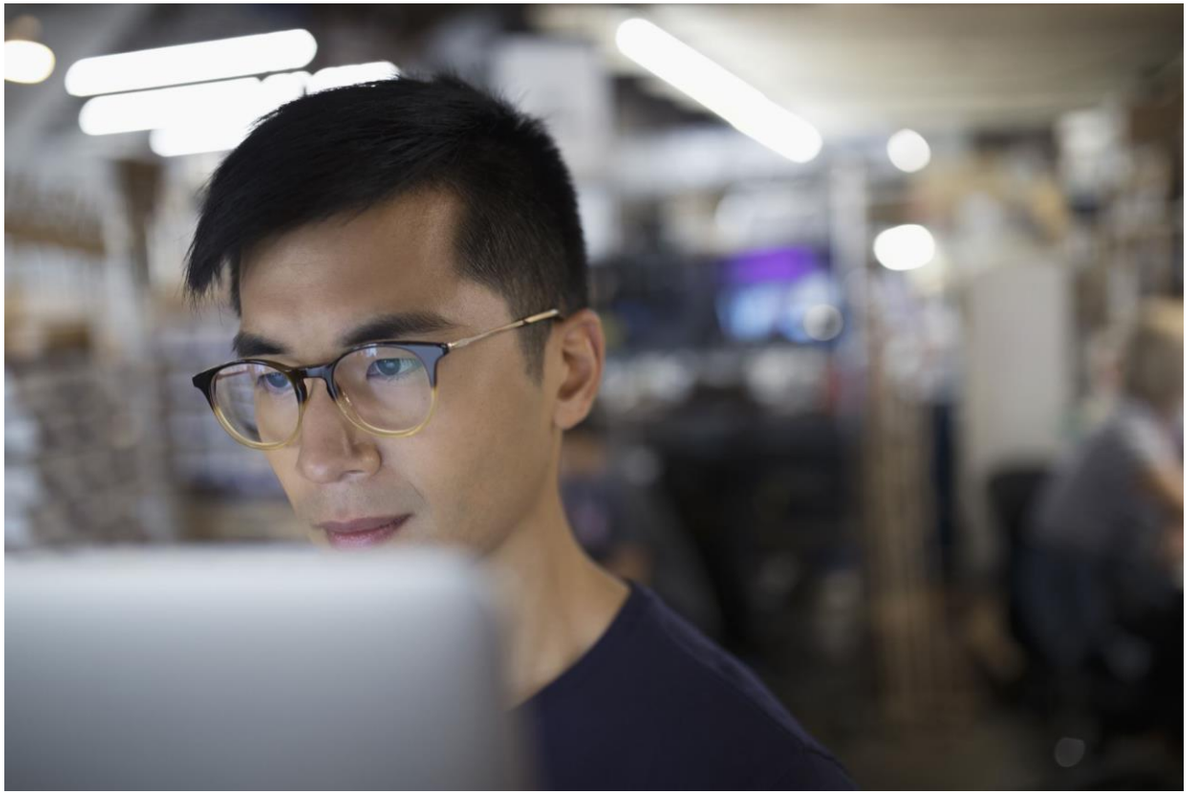
RACF Macros and Interfaces: SMF80/Unload



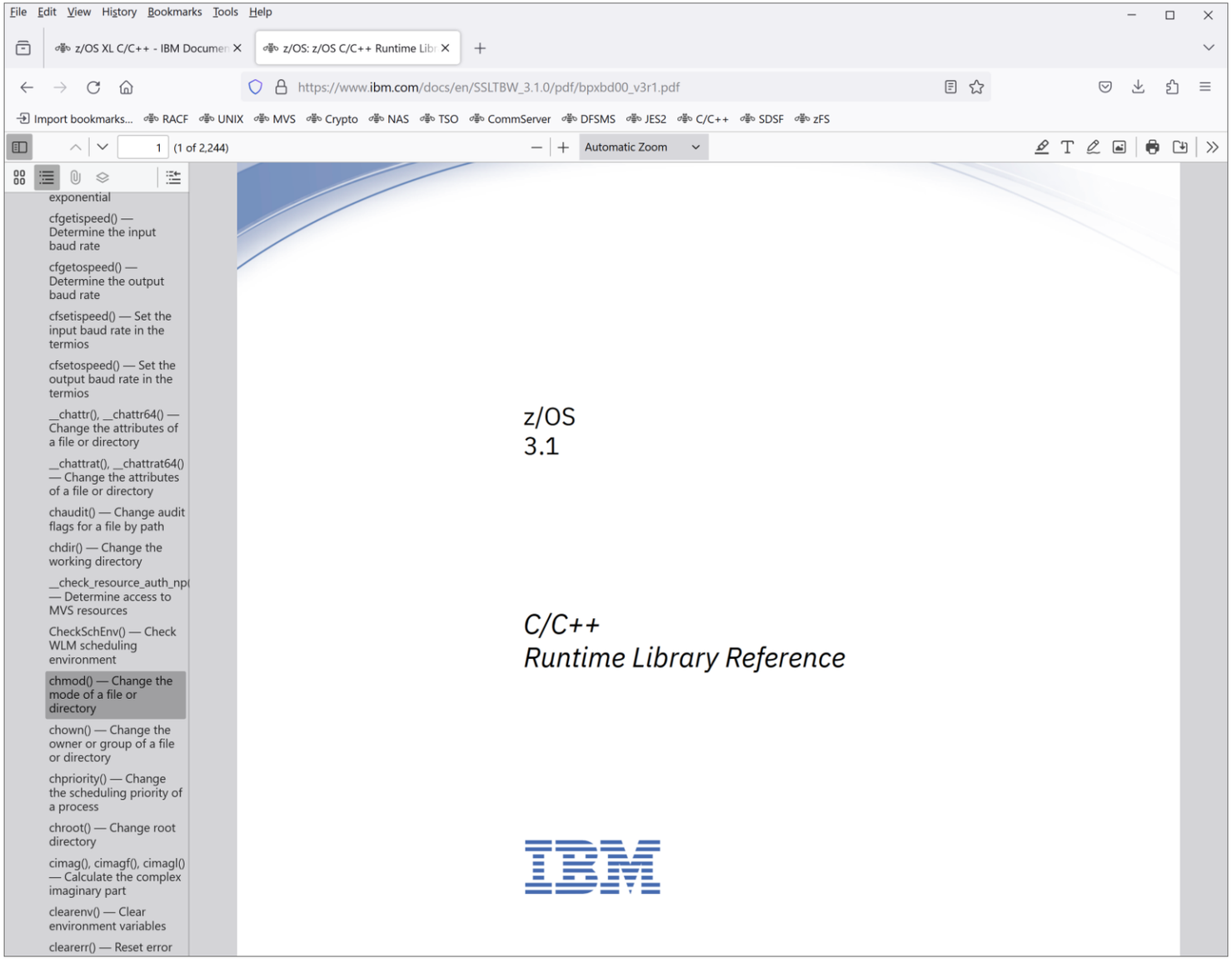
I count 41 UNIX-related SMF 80 Events Codes you can query in SMF Unload (but not Report Writer!) output. E.G. file access, security attribute changes, process creation, changes to process identity, etc.

Further down and not shown are the Database Unload record formats. You can see OMVS segment info, as well as profile and access list info for UNIX related profiles, such as UNIXPRIV.

For application developers

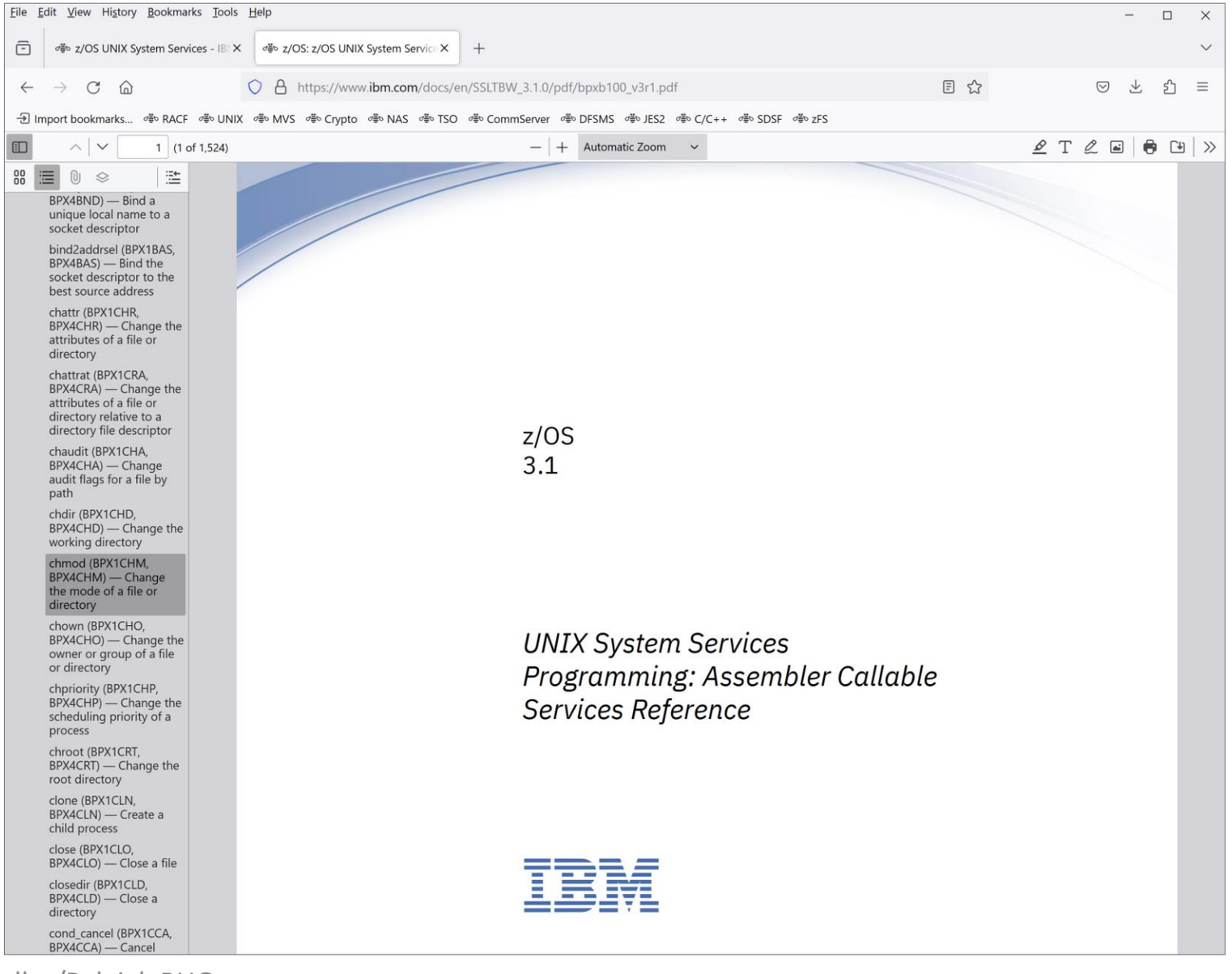


C/C++ Runtime Library Reference: individual APIs



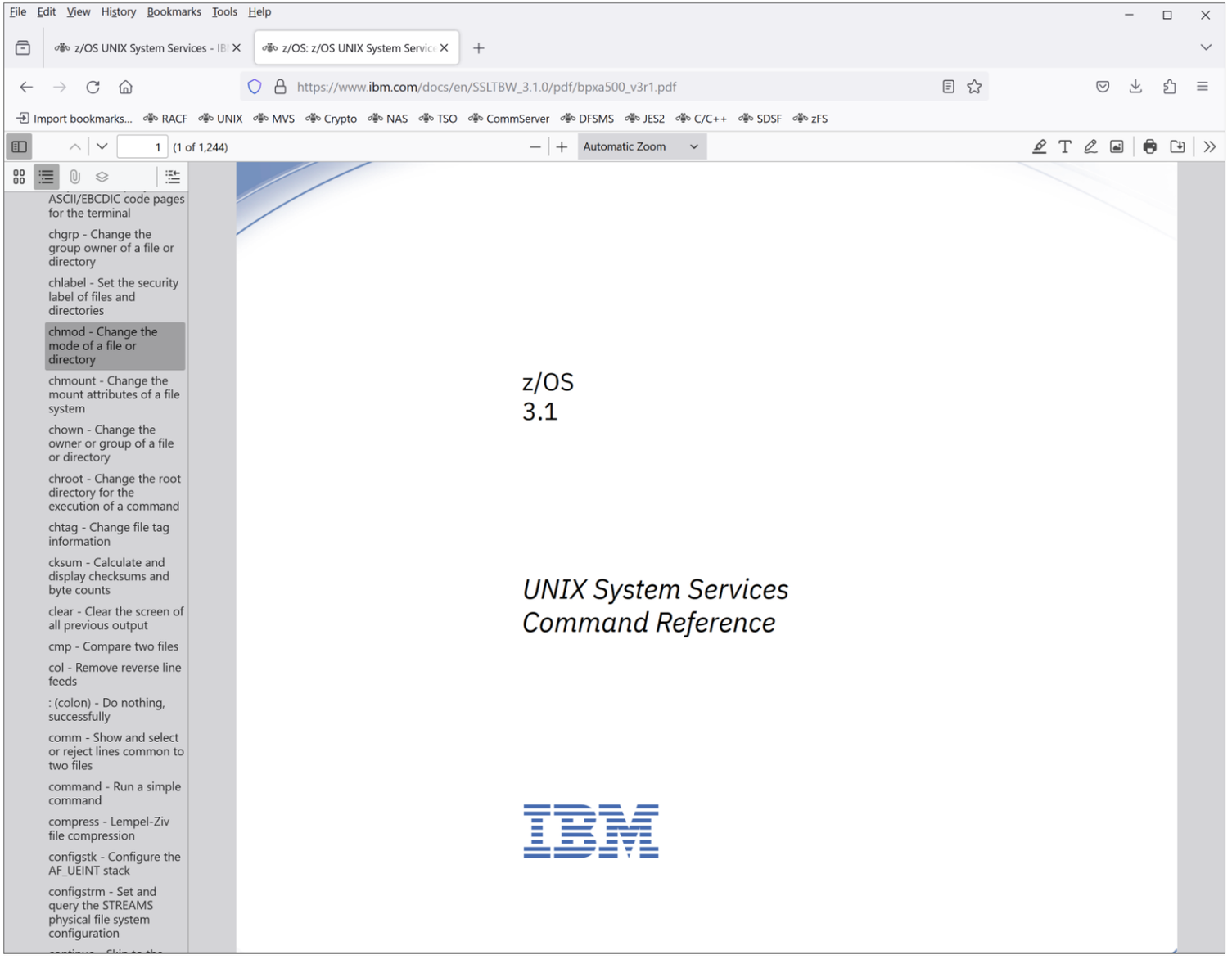
Describes the high level language UNIX APIs and the authority necessary to call them.

USS Programming: Assembler Callable Services Ref



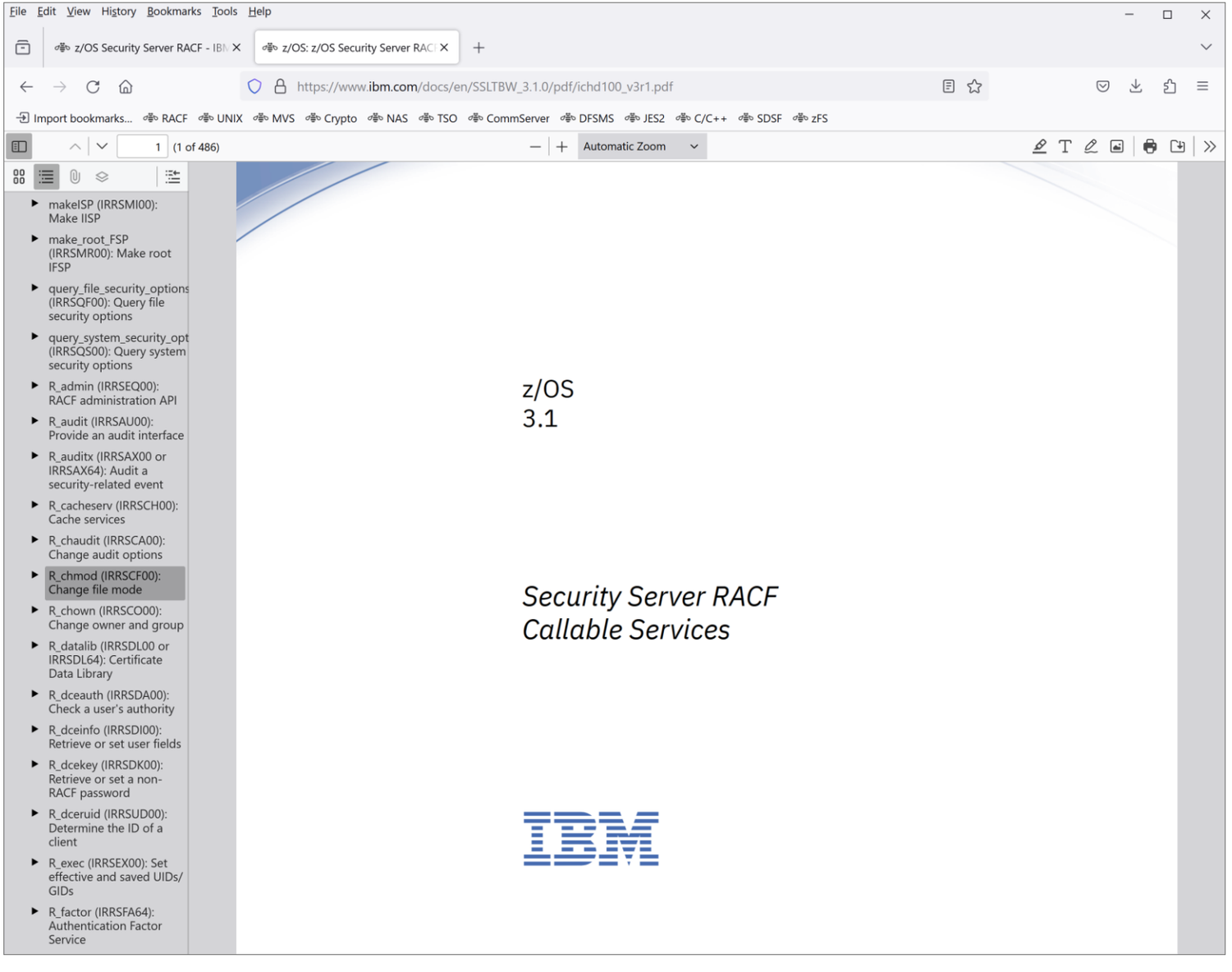
Describes the assembler APIs, many of which correspond to C/C++ APIs, providing a good cross-check on security requirements

USS Command Reference: corresponding command



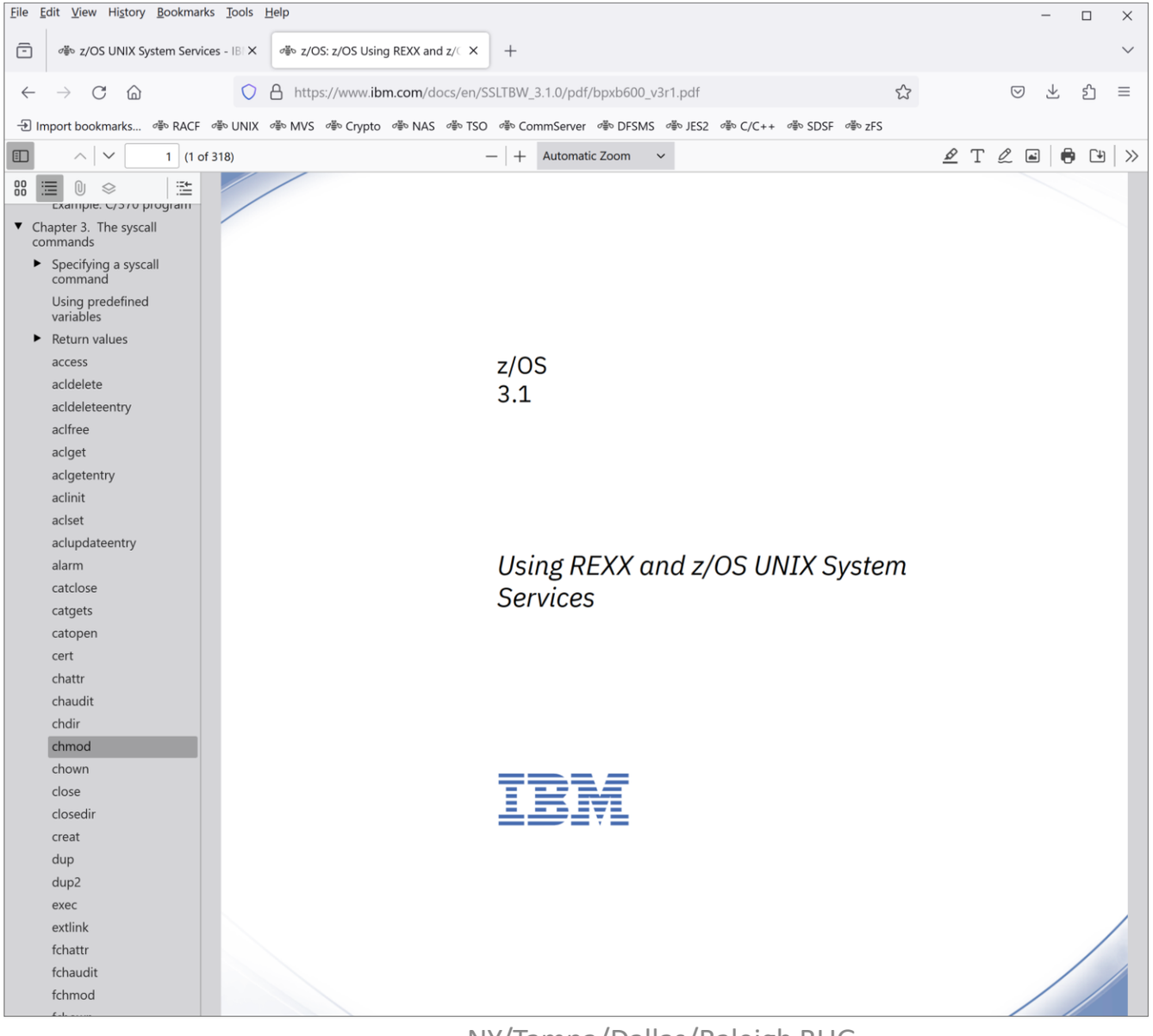
Since many UNIX APIs correspond directly to commands, another good cross check

RACF Callable Services



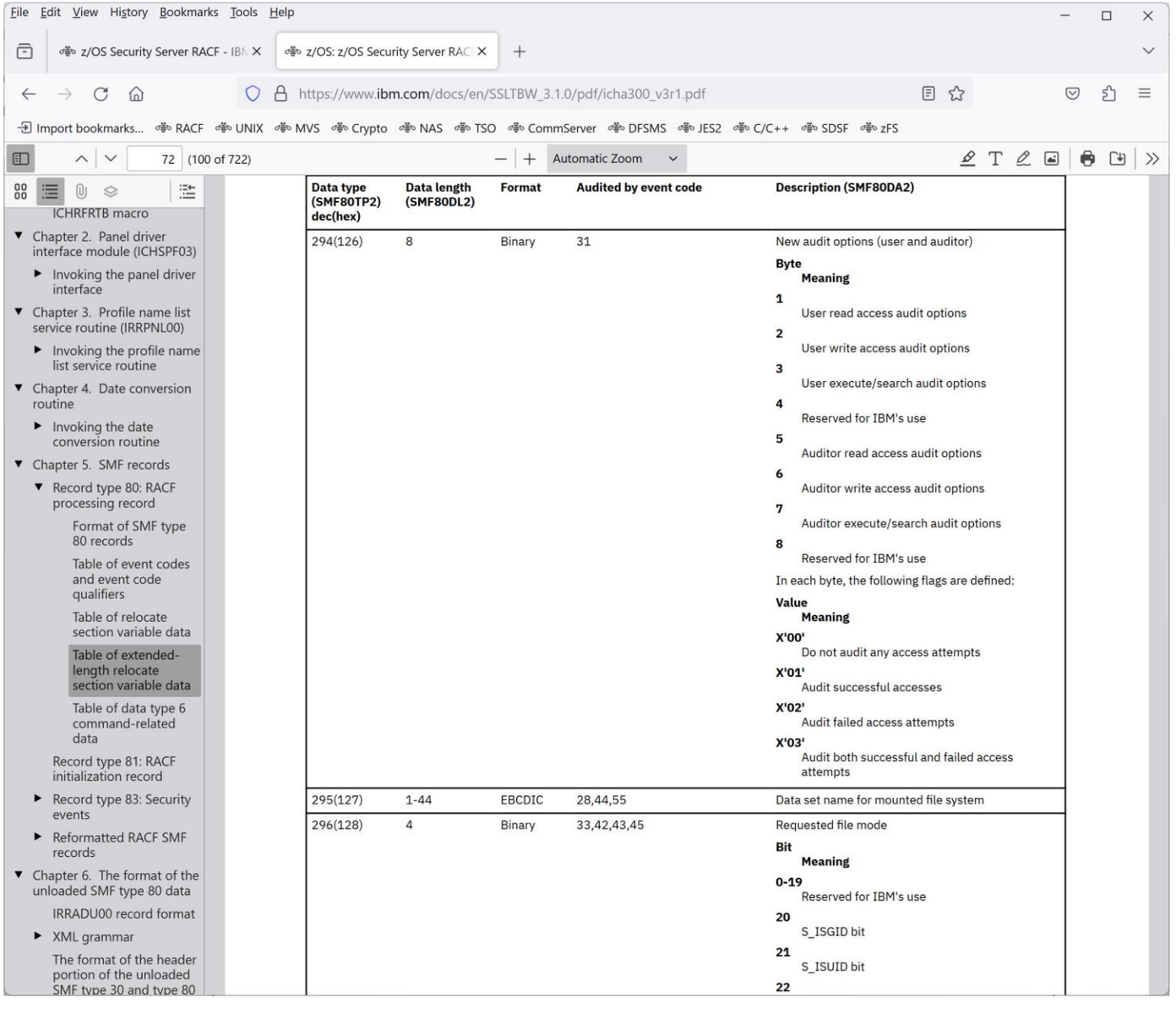
Though most of these are only for the kernel and file system, again, they correspond to the UNIX assembler APIs and provide a cross check

Using REXX and z/OS UNIX System Services



All kinds of good APIs for REXXophiles, many corresponding to the previously described APIs. The UNIX command download mentioned later uses some of these.

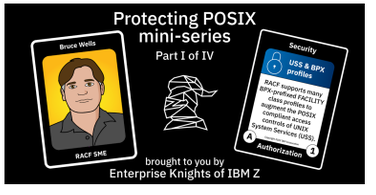
RACF Macros and Interfaces: SMF80/Unload



Applications can use SMF Unload output too! And if you like to crawl through the raw record, the formats are documented here also, in gruesome detail.

My UNIX brain-dump of record

1. **z/OS UNIX System Services File System Security**



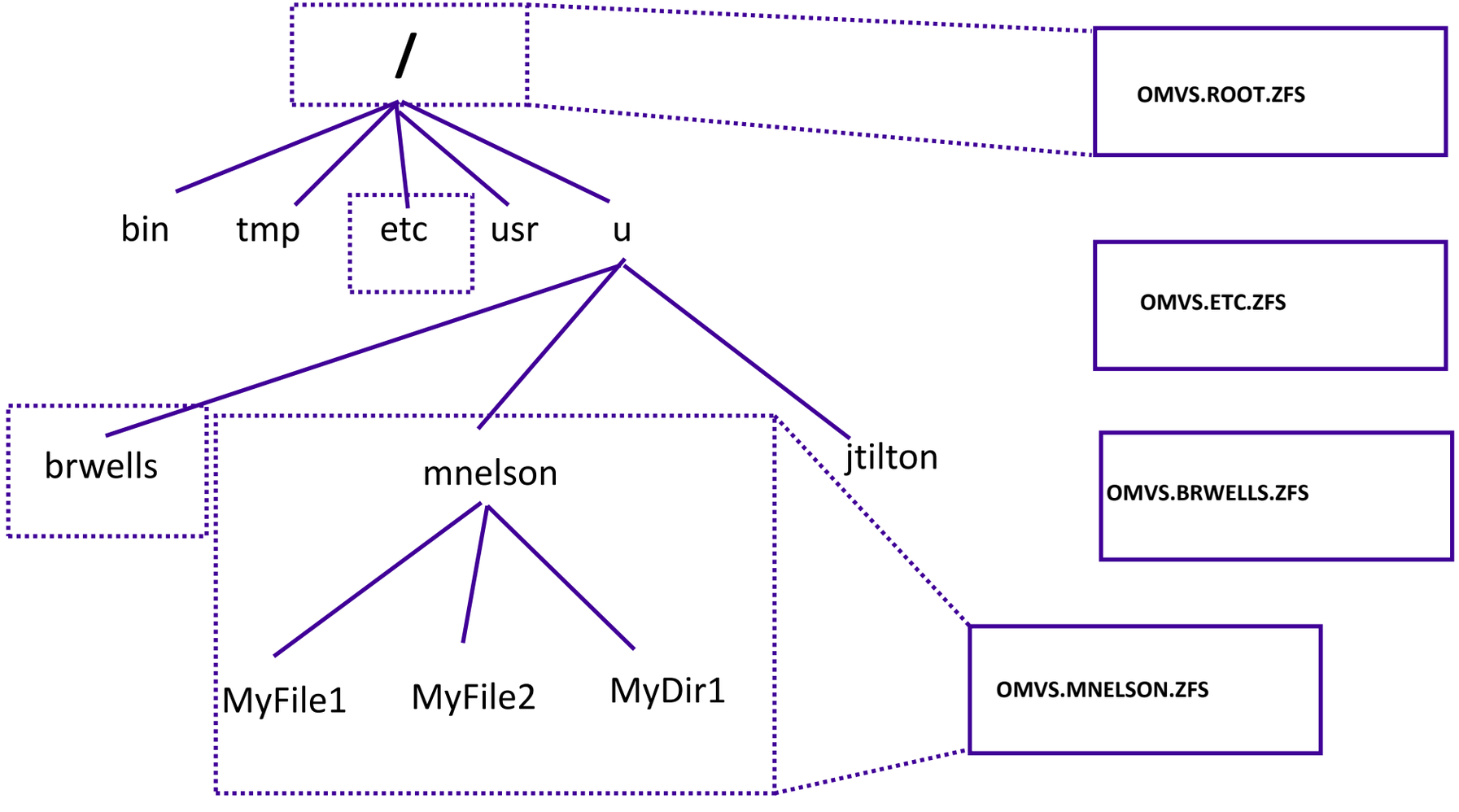
2. **z/OS UNIX System Services File Security**

3. **z/OS UNIX System Services Users and Groups**

4. **The UNIX superuser**

UNIX file *system* security

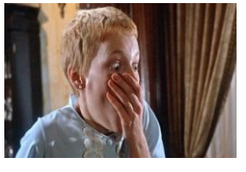
Data Sets (aggregates) are MOUNTed into a hierarchical structure



TSO MOUNT FILESYSTEM(OMVS.BRWELLS.ZFS) MOUNTPOINT('/u/brwells') MODE(RDWR) TYPE(ZFS)

Controls at the aggregate level

- Good old DATASET protection
  - SYS1.PARMLIB
  - zFS aggregates
    - Including user file systems, which should not use the user ID as the HLQ
- Ability to MOUNT and UNMOUNT
  - With specific modes like nosetuid, read-only, read/write,
  - SUPERUSER.FILESYS.MOUNT in the UNIXPRIV class
  - SUPERUSER.FILESYS.**USER**MOUNT in the UNIXPRIV class



nosecurity

- Ability to encrypt
  - 'zfsadmin encrypt' command can encrypt while file system is in use
- RACF FSEXEC-class profiles to prevent executables from running
  - Think /tmp, which is where attackers like to deposit a 'fingerprinting' script
- RACF FSACCESS-class profiles to prevent entry, even from UID(0)

Auditing the environment

- Looking at the RACF profiles (SEARCH, RLIST, LISTDSD, IRRDBU00)
- ‘df –v’ shell command displays detailed information on all the mounted file systems
  - Mount point
  - Mount mode
  - Aggregate name
  - File system type
  - Etc
- ‘find’ shell command – e.g. to discover your setuid/setgid files
- Zfs Unload utility on RACF downloads page
  - https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/ZFSUnload

UNIX *file* security

Security attributes are meta-data of the file

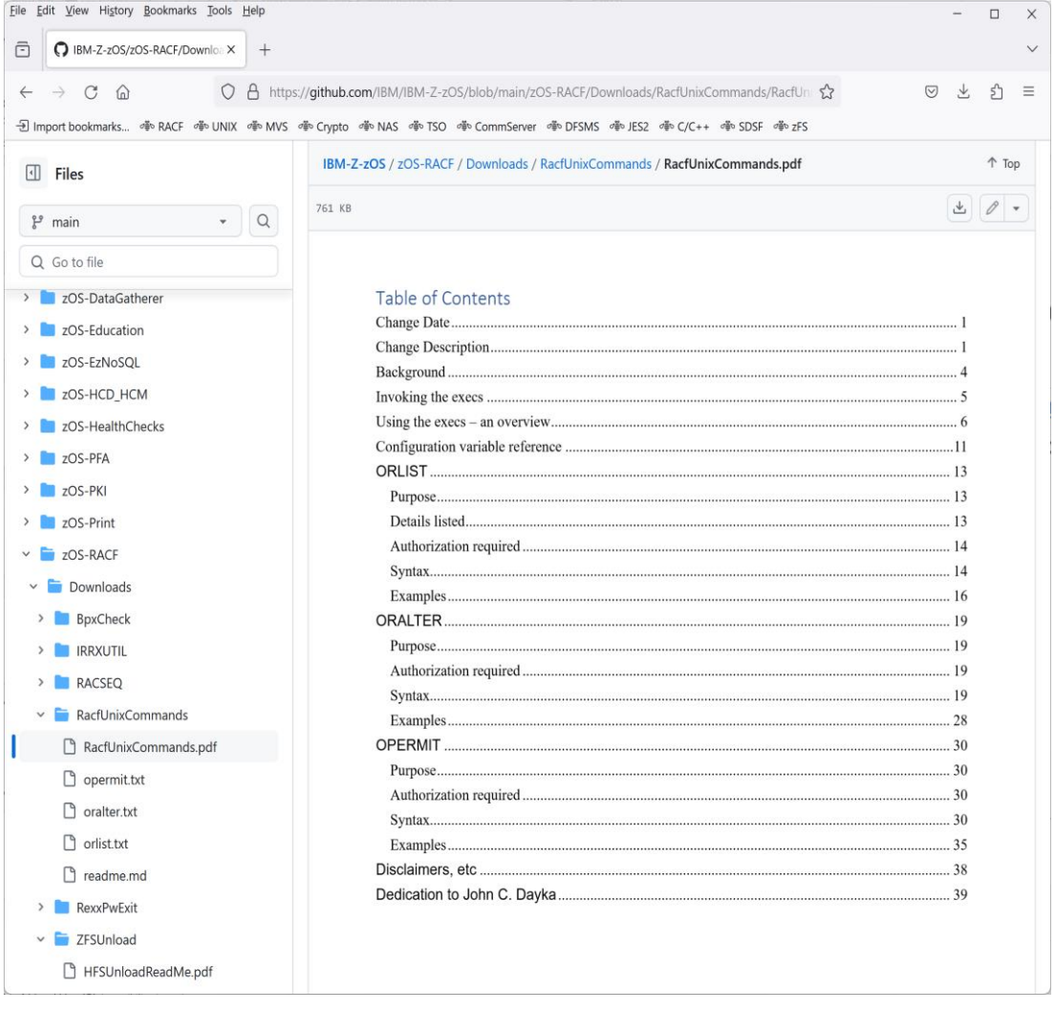
- Ownership: user and group
- Permission bits and access control lists (acls)
- Set-uid, set-gid, and sticky bits
- Logging specifications
- Extended attributes like apf and program-control
- Security label

File security attributes and how to manage them

initialized to ...	File security info			changed by ...
effective UID	User (UID) owner			chown command
parent dir's group	Group (GID) owner			chown or chgrp
varies by function (qualified by umask)	Permission bits			chmod command
	Owner rwx	Group rwx	Other rwx	
flags specified by open()	Flags			chmod command
	set-uid	set-gid	sticky	
read, write, and execute failures	Owner audit options			chaudit command
	read	write	execute	
no auditing	AUDITOR audit options			chaudit –a command
	read	write	execute	
SHAREAS bit on for executable files	Extended attributes			extattr command
contents of parent's default ACL	Access Control List			setfacl command
SECLABEL of covering dataset	Security label			chlabel command

Or use the RACF/TSO/UNIX command download

- https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/RacfUnixCommands
- REXX execs that act like RACF commands would if file security were protected with profiles
  - ORALTER, ORLIST, OPERMIT
  - Uses RACF keywords where possible
  - Uses RACF*ish* keywords where not
  - All create output files
  - All have a 'recursive' option
  - All have a 'path' option to operate on all components of a specified path
  - All have optional configuration variables (like 'noRun' to see what command *would* do)
  - Documented as if they were in the RACF Command Language Reference



OPERMIT syntax – RACF keywords used

OPERMIT (or whatever name you have chosen for it)
[absolute-path-name-1]
[ACCess(access-authority) | DELETE]
[ACL] [FMODEL] [DMODEL] | [ALL]
[CLASS(FSSEC)]
[DEBUG]
[FROM(absolute-path-name-2)]
[FTYPE(ACL | DMODEL | FMODEL)]
[ID(name ...)]
[OUTFILE(path-or-dataset-name)]
[PATH]
[RECursive[(CURRENT|FILESYS|ALL)]]
[RESET]
[VERBOSE]

Examples

- opermit /u/bruce/file1 id(mark) access(r-x)
- opermit /u/bruce/file1 from(/u/brwells/file2)
- opermit /u/bruce/file1 reset
- oralter /u/bruce/myfile perms(rwxr-x---)
- oralter /u/bruce/file1 owner(bruce) group(racfers)
- oralter /u/bruce/myfile noapf noprogram perms(o-w) recursive
- orlist /u/brwells
- orlist /u/brwells auth
- orlist /u/brwells/file1 auth path

ORLIST default and AUTH formats

```
CLASS      NAME
-----      ----
FSSEC      /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
```

```
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
```

Auditing the environment

- The shell ‘ls’ command with various options
  - ‘ls –l’ for most of the options (ownership, permission bits, more)
  - ‘ls –W’ for the logging options
  - ‘ls –E’ for the extended attributes
  - ‘ls –M’ for the security label
- ‘find’ shell command – find files with any attribute(s)/value
- ORLIST in the RACF/TSO/UNIX download
  - Displays all attributes in RLIST-style format
- zFS Unload utility on RACF downloads page
  - https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/ZFSUnload

UNIX users and groups

Provisioning UNIX

- Prevent UID reuse with SHARED.IDS profile in the UNIXPRIV class
- Assign OMVS segment with UID
  - Manually
  - Using the AUTOUID keyword
  - Using automatic OMVS segment assignment
  - Using an identity management provider that takes the rest of your enterprise into account
- The user's default group must have an OMVS segment with a GID
- Allocate a user file system data set
  - Perhaps using the UNIX automount facility

Least Privilege – preventing UNIX

- If a new user has no need for UNIX, don't grant it
  - Why worry about new attack vectors?
- If you have automatic assignment in place, give the user an 'empty' OMVS segment as part of provisioning to block it
  - ADDUSER JOE OMVS
  - ALTUSER JOE OMVS(NOUID)

De-provisioning UNIX

- Beware of residual access in the file system
  - File ownership
  - acl entries
- Have a process to
  - Deallocate their user file system
  - Search and destroy(/replace) references elsewhere in the file system
  - Don’t re-assign their UID until this has been verified
- Delete the user, or at least its OMVS segment
  - But if you haven’t done the above, remember its UID (in a custom field?) so you can associate file system references with the user ID
- And all that normal RACF stuff (IRRRID00, for example)

Auditing the environment

- Good old LISTUSER, LISTGROUP, and IRRDBU00
- ‘id’ shell command displays user’s identity as UNIX sees it

```
$ id bruce


uid=266(BRUCE) gid=115(COOLKIDS) groups=213(MYDEPT), 300(MYORG),
7356(RACFDEV),9004 (IZUUSER), 1151(PEVID), 1(POSIX), 1768(RACFALL),
1000044(ZOSDEV),1000046(ZOSTOOLS),1000043(ZRACFU)
```

- ‘find’ command again
  - Can find ownership and acl references in files, by user ID/group or UID/GID
- zFS Unload utility again
  - https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/ZFSUnload

UNIX superusers

A user with UID(0), or a TRUSTED or PRIVILEGED started task can

- Create, read, update, and delete any file
- Read and write to network sockets
- Change security attributes of a file
- Consume resources in excess of system limits
- Kill and inspect processes
- *Switch into the identity of any UNIX user without authentication*
  - And then maybe write into APF libraries? Manage RACF profiles?

- Totally pwn you
- Exasperate your auditors due to Separation of Duties violations

Fortunately, there are ways to limit capabilities

Scope superuser security management capabilities

- UNIXPRIV SUPERUSER.FILESYS.DIRSRCH
- UNIXPRIV SUPERUSER.FILESYS.CHOWN
- UNIXPRIV SUPERUSER.FILESYS.CHANGEPERMS
- FACILITY BPX.FILEATTR.APF
- FACILITY BPX.FILEATTR.PROGCTL

- In fact, a superuser cannot change extended attributes without this FACILITY authorization
- This underscores the fact that where we have extended the POSIX standard for z/OS-specific functions, we tend not to respect UID(0).

Scope superuser system programmer capabilities

- UNIXPRIV SUPERUSER.FILESYS.MOUNT
- UNIXPRIV SUPERUSER.PROCESS.KILL
- UNIXPRIV SUPERUSER.PROCESS.PTRACE

Scope superuser application identity capabilities

- UNIXPRIV SUPERUSER.FILESYS
- UNIXPRIV SUPERUSER.FILESYS.VREGISTER
- UNIXPRIV SUPERUSER.PROCESS.GETPSENT
- UNIXPRIV SUPERUSER.PROCESS.PTRACE
- UNIXPRIV SUPERUSER.SETPRIORITY
- UNIXPRIV SUPERUSER.SHMMCV.LIMIT
- FACILITY BPX.SERVER
- FACILITY BPX.DAEMON
- SURROGAT BPX.SRV.*userid*
- ‘Limit’ fields in the USER OMVS segment

Servers and Daemons

- Server: establishes a thread (subtask) for client after authentication (e.g. HTTP server)
- Daemon: establishes process (address space) for client after authentication (e.g. FTP daemon)
- Instead of requiring APF/supervisor state, access to a FACILITY profile and requirement for a clean address space is sufficient to establish identity
- BPX.DAEMON(READ) and UID(0) required
- BPX.SERVER or UID(0) required
  - READ: Server and client require authority to protected resources that may subsequently be accessed (unauthenticated client)
  - UPDATE: Only client requires access to resources accessed (authenticated client)

Thank you! Any Questions?