[illegible]

WAPD-T-2937

A SECURE FILE MANAGER FOR UNIX

BY

R. G. DeVRIES

BETTIS ATOMIC POWER LABORATORY
P. O. BOX 79, MAIL STOP 37U
WEST MIFFLIN, PA 15122
TELEPHONE: 476-5413

WESTINGHOUSE COMPUTER SYMPOSIUM
FOX CHAPEL YACHT CLUB
FOX CHAPEL, PENNSYLVANIA
OCTOBER 1990

WAPD-T-2937

A SECURE FILE MANAGER FOR UNIX

BY

R. G. DeVRIES

BETTIS ATOMIC POWER LABORATORY
P. O. BOX 79, MAIL STOP 37U
WEST MIFFLIN, PA 15122
TELEPHONE: 476-5413

ABSTRACT

The development of a secure file management system for a UNIX-based computer facility with supercomputers and workstations is described.

Specifically, UNIX in its usual form does not address:

1. operation which would satisfy rigorous security requirements.

2. online space management in an environment where total data demands would be many times the actual online capacity.

3. making the file management system part of a computer network in which users of any computer in the local network could retrieve data generated on any other computer in the network.

The characteristics of UNIX can be exploited to develop a portable, secure file manager which would operate on computer systems ranging from workstations to supercomputers. Implementation considerations making unusual use of UNIX features, rather than requiring extensive internal system changes, are described, and implementation using the Cray Research Inc. UNICOS operating system is outlined.

I. INTRODUCTION

It is increasingly apparent that UNIX is becoming a way of life for many computer installations. Practically all computer professionals entering the field today have at least some knowledge of UNIX, and many have absorbed the UNIX philosophy to the extent that creative work is based on what could be called the "UNIX way of doing things." UNIX has penetrated all corners of the computer industry, from personal computers to supercomputers. As a result, many installations, including government and commercial laboratories, are installing UNIX but still have to deal with the traditional problems of data security and space management. In addition, users demand file compatibility so that data generated on one system can be processed on another.

UNIX has as a basic assumption the free and open access of its users to data. In some government or commercial facilities, this openness must be limited on a need-to-know basis. As described in this paper, a file management system can be installed on top of UNIX to act as the only access method to an entire file system. This "secure" file system can be the basis for ensuring that only authorized users can access restricted data.

Also, online space is a limited resource. On a supercomputer, demands can greatly exceed the available disk space. Users expect online space to be managed automatically or by operations personnel. A file may be automatically removed from disk at any time with full confidence that a copy of the file is available offline or on an automated mass storage device.

Such file systems exist on various scientific computing systems, but not in UNIX. This paper describes how one file manager was developed for Cray UNICOS (developed by Cray Research, based on ATC UNIX System V) and how the code was ported to a workstation UNIX environment. The requirements addressed were: security, online space management, and portability.

II. ADAPTING A FILE MANAGEMENT SYSTEM FOR UNICOS

Simply stated, the File Manager (call it FM for short) stands between the user and the system; it is the only way for the user to access files within a certain well-defined set. FM can use file access mechanisms provided by the system vendor, but since it stands between the user and the file system, the user cannot access files without passing FM's validity checks. In addition to basic file access, FM provides other desirable features, such as:

- The use of access lists by which a user can specify a need-to-know group of users to be granted access to a particular file.

- The ability for the user to specify on which kind of medium or group of volumes a file will be stored.

File migration is important in an overall file management system. In a dynamic system, file turnaround is fast, and online space demands are large. Therefore, FM has an automatic migration strategy. The migration strategy is designed to incorporate future mass storage technological improvements. The migration function is designed to be transparent to the user: he or she can retrieve any file without concern for the actual residence of the file.

III. FM CHARACTERISTICS

In order to make such a file manager work in UNICOS, several drastic steps are necessary: or at least, the steps seem to depart from typical UNIX systems.

First, and probably most unusual, is that a user's home directory is empty at job initiation and is completely purged at termination. This feature is incorporated for a number of reasons:

- Online space limitations are so severe that it would be impractical to expect users to manage their own space to prevent data overflow and system gridlock. By removing all files from the home directory at logoff time, space is reclaimed immediately. Only those files specifically given to FM for storage will persist, and only within directories owned and managed by FM. In this way, FM can manage its own space by running migration and disk wiping utilities as needed.

- There is less chance of intentional or accidental compromise of data if it resides in a more secure place than a user's home directory. In normal UNIX systems, a user's directory is open to attack. With FM, the user's HOME directory is populated with files only while the user is logged on. That greatly limits the chance for access by an unauthorized person.

- The case can be made that this strategy enhances user convenience. In a large scientific environment, many temporary files are created during a job session. These files are a nuisance to clean up afterward, and they contribute to the waste of valuable online space. With FM, all files are removed except for those files which the user explicitly directs FM to save.

Second, certain commands which users of normal UNIX systems expect to be available are disabled or severely restricted. Such commands are few in number, and they constitute obvious security weaknesses or loopholes. The disabled commands are:

- chmod. It is not permitted to set group or world permission on any file. Users can set owner access on their own files, however, since they need the ability to set execute permission.

- su. Users can have more than one logon ID, with different classification levels. "su" makes it possible for a user to switch to his or her other ID (possibly at a different classification level) and pass data back and forth, effectively giving data an inappropriate classification. Thus it is necessary to disable the "su" command except for the "superuser" or authorized system administrator.

Third, the concept of "media sets" is introduced. A media set is a logical collection of volume numbers so that a user can specify that the files he or she stores are transferred to a volume within a specific media set. To the user, a media set is an amorphous entity used as a basis for file collection. In actual implementation, a media set can refer to one of several kinds of storage medium or even a combination of media. The real value of media sets is seen at file retrieval time. It is not unusual for a large scientific job to retrieve hundreds of files for processing. Media sets allow for the files to be spread among a relatively small set of volumes, statistically reducing the number of volume mounts necessary to retrieve a large number of files. Since FM maintains the vsn of residence for each file, the user does not need to specify the volume number at retrieval time.

Fourth, it is necessary that all files being fetched for a job must be available to a job before it gets into execution. The reason is that large production jobs tie up considerable resources during execution. An obvious example is scratch disk space. If, as on some systems, a file is not retrieved from secondary storage until it is opened, the system resources remain assigned to the job while the file is retrieved. Such retrievals could take minutes. Since the fetched files could number into the hundreds, the chosen approach is to fetch all the files before execution of the production code, while minimal system resources are assigned to the job.

Fifth, access to directories is severely restricted. In fact, a user does not have write access into his or her own home directory. Rather, a directory is created under the home directory and an FMHOME variable is created to point to this new directory. The new directory and other specially created directories are the only directories the user can write into. (The other special directories are created under file systems other than the one containing the FM system, such as the SSD on a Cray system. The directories give the user access to those other file systems.) The FMHOME directory and the other special directories are removed at job termination.

Sixth, all user jobs are initiated in the "C" shell. The reason is that the "C" shell provides for scripts that are executed at both login and logout. These scripts are used for the special functions such as creation of the job-duration directories at login, and a forced FM wrapup followed by removal of the job-duration directories at logout. A simple automatic mechanism is provided for the user if he or she desires to operate in the Bourne shell after the initial login is completed. A small side effect of the special login mechanism is that jobs are already two shells deep at the time the user gets control. Interactive users have to exit twice to leave the system.

IV. USER INTERFACE

To interact with the File Manager, the user calls a special program in UNICOS. (We can call it "fm" for convenience, staying with the UNIX convention of using lower-case letters.) The "fm" program expects a series of directives, which can be entered interactively or in batch. In batch, the set of directives can be given by redirected input or in a "here document", both of which are familiar concepts to UNIX users. The important point is that there is a series of directives concerning files needed by or generated by the user.

FM operates in two modes: directive processing and wrapup. An "fm" command with a single parameter specified on the command line calls for an FM "wrapup", which will be discussed later.

A basic set of directives is as follows:

fetch: retrieve a file from FM and make it available to the user in the current working directory or in a directory specified by the user.

store: flag a file for store. The file may or may not already exist, but must exist at the time an FM wrapup is performed.

merge: fetch a file and read its contents for additional FM directives.

erase: remove a file from FM.

The parameters are identified by key letters followed by equal signs and values and are separated by commas. The parameters are as follows:

FM Command Parameters (global identifiers)

- P: Prefix, or the identification of the owner of a file. The creator of the file is indicated separately at creation time so that files can change owners without losing information about their creation. The prefix for each user is assigned by the FM administrator. The default for the P parameter is the user ID for the job.

- I: File ID, or the computer independent global FM identifier for the file.

- T: Data type. This field is used to identify categories of files to make their management easier for users and programmers.

- V: Version: an integer field assigned by the user or application program and used to distinguish between different versions of the same file. The default is 1.

- G: Generation. Frequently, users want to run a sequence of jobs to study variations (generations) of a scientific model. The generation field provides a convenient way to assign a name to the set of output files representing one variation without having to change all of the FM directives in each job. This can be done by defining a generation name for the job in a UNICOS variable and by using the variable in the generation field of the store directives. If variable XYZ is set by the user to the string "GEN1", a G=$XYZ parameter causes GEN1 to be entered as the generation name.

FM Command Parameters (file attributes)

- A: Access codes. Values for the access codes are used to specify the granularity of access for both fetch and alter. A granularity of 0 gives all users permission. A granularity of 4 gives access only to the owner. Intermediate values specify an access group of successive levels in an organizational hierarchy. In addition, a wildcard group or individual outside the hierarchy chain can be specified. This discussion applies to an organizational chain of height 4. The access code range can be easily extended for chains with more vertical steps.

- M: Media, i.e., the name of a media set (as described above) into which the file is to be placed. The name of the media set may be contained in a UNICOS variable which can be referred to in an FM directive.

- R: Reel number (or vsn) for those directives which refer to a specific volume of tape.

- S: Synonym. The S parameter specifies the name the file will have for the duration of the job

- W: Working directory pathname. This provides for the file to be fetched into or stored from a pathname other than current directory.

The global identification of an FM file consists of the P, I, T, V, and G parameters. There are default values for P (the user ID for the job), V (version one), and G (null). If the G field is used on the store, the corresponding G parameter is required on the fetch.

All other parameters are FM file attributes that cannot be used to distinguish between files on an FM directive. In particular, the A, M, and R parameters provide information for the FM directory which might be used by a utility program, but is not part of the file identification.

V. DISCRETIONARY ACCESS CONTROL LISTS

Discretionary access control consists of two stages. The first stage is based on the hierarchical structure reflecting the organization of users at the installation. The code (explained in detail above under the A parameter) applies separately to read and to write/erase.

The second stage of access control takes the form of access permission lists which are examined whenever access is denied by the first stage. Utilities are available to allow users to specify exceptions to the first stage access controls. A file owner can specify lists of users who are to be given access to a file even though access is denied by the first stage. In both stages of access control, access can be controlled separately for read and write/erase.

Access list entries can be created by any user. They can be modified only by the list creator or system administrator. The access lists are maintained by FM in a separate file. The access lists convey access for certain files to one or more users for a limited period of time. This period is a fixed length (set by the installation), but can be easily renewed by the access list creator. A file whose FM directory entry refers to a certain access list is accessible to the users named in that list in addition to the users embraced by the hierarchical access code (the first stage discussed above).

VI. FILE AND DIRECTORY SECURITY

In the FM implementation for UNICOS, no permanent user files are permitted outside of FM. The key to security is that all permanent directories in the system (those that persist between jobs) are established with no write permission. This includes each user's actual home directory. The fm program is a 'setuid' program owned by the superuser and is the only program with permission to write into permanent directories.

All of the permanent user data files on the system are owned by FM and are inaccessible to all but the superuser. Access to the files through FM causes a copy or a link to be established for the user's job-duration directories. Actual residence of the FM files is under FM control. The files may be on disk, or they may be retrieved from another level of storage over the network or from tape. Since all of the files belong to FM, space can be made available on the disks for large overnight problems by migrating files to other storage or by discarding local copies of previously migrated files.

Files to be stored in FM must be created under the directory indicated in the corresponding store directive. Such files are copied to FM directories at FM wrapup; the user copies are destroyed at job termination.

VII. FM WRAPUP PROCESSING

The term "wrapup" in this context refers to an action taken by FM to dispose of files under FM cognizance (those files referred to by the user via an FM directive). Wrapup options are discussed below.

FM wrapup is done automatically at session termination or logoff to move any files marked for storage out of user directories and into FM directories and to delete all temporary user directories. There are times when a user may want to invoke a wrapup during a session; this can be done at any time by adding a parameter after the fm program name:

```
fm -p
```

The "fm" statement parameter -p can be used to denote the level of the wrapup. The value of "p" can be s, r, or w.

-s: Store all files designated for store that have been written by the user, but leave all files in place under the user's directories.

-r: Store all files designated for store that have been written by the user, and remove them from the user's directories, leaving fetched files in place.

-w: Wrapup, or store all files designated for store that have been written by the user, and remove all FM files from the job-duration directories (both fetched and stored).

Stores are accomplished by creating files under an FM subdirectory assigned to the owner-user. The simplest case is a file in the user's FMHOME directory. That directory is under the same UNICOS file system as the FM system, so a UNICOS move ('mv') can be used to remove the file from the FMHOME directory and make a new entry under FM. The FM name of the file is computed from a seed in the user's directory, which is updated at each store to ensure uniqueness of the names.

If the level of wrapup is -s, a link ('ln') is done instead of a move, to keep the file in the user's job-duration directory. Write access for the file is removed at this time.

If the file is in a different file system from the system where FM resides, a 'cp' UNICOS command is done to make a copy of the file in the FM file system. If the wrapup level is -r or -w, the copy is followed by a remove ('rm').

VIII. FM COMMUNICATION FILE PROCESSING

A file is needed for FM to keep track of each file directive processed by FM during a job. We can call the file FMCOM for convenience. The FMCOM file is also the communication link between FM and user programs; a program can interrogate FMCOM to determine what files are assigned to the job. The processing of each FM directive results in at least one record being added to FMCOM. At the end of the set of directives, FM makes a pass through the FMCOM file, resolving all unprocessed records.

The FMCOM file is scanned first for merges. Any merge directive is processed specially: the merge file is fetched, its directives are parsed, and the records are added to the FMCOM file. After a merge file has been processed, the pass through FMCOM starts again. The process continues until a complete pass is made without finding an unprocessed merge directive. Then FM knows the totality of files to be processed for the user, and fetches and stores can be processed as described below.

For fetches, the appropriate user directories are searched and access validated. (These directories are not UNICOS directories but UNICOS files belonging to FM.) If the access check passes, a UNICOS link is made into the user's FMHOME directory. If the user wants the file fetched to a different directory, the file is copied. Obviously, it is more efficient to do the link, but there may be other reasons (such as performance) to do the copy. The user selects an alternative directory via the W parameter on the "fm" directive or by being under a different current directory at the time of issuing the directive.

IX. EXAMPLES

The following are typical but simplified examples of user jobs using FM. The examples are in batch; similar calls to FM can be made interactively.

```
fm <<fmeof
merge,i=gener
fmeof
myprog
```

The above job causes FM to fetch a special file called gener and process further FM directives found therein. The gener file might contain directives like:

```
fetch,i=design file,t=101,v=1-100h,s=infile
store,i=design file,t=101,v=1-100h,s=outfile
fetch,i=myprog,t=62,s=myprog
```

The fetch is for the highest version in the range 1-100 of file "design file" with data type 101. The store directive is for the highest unoccupied version in the range 1-100. The program "myprog" is coded to read a file with the name "infile" and write a file named "outfile". Repeated runs of the job will read the latest version of "design file" and store the resulting file as the next greater version. When the job is finished, an automatic "fm -w" will cause the generated "outfile" to be stored under the identifiers given in the "store" directive.

The user may want a current listing of the files after each new one is generated by "myprog". The following slightly more complex job will accomplish this.

```
fm <<fmeof
merge,i=gener
fmeof
myprog
fm -w
fmlist
```

The user forces an FM wrapup rather than relying on the automatic wrapup, so that the subsequent "fmlist" command will list all of the user's FM files including those stored by this job.

X. IMPLEMENTATION CONSIDERATIONS

In a large scientific environment, online storage may be divided many ways. For discussion, let us consider the following:

(1) user space, which contains all the user home directories,

(2) temporary space, used for high-volume scratch storage, and

(3) SSD space, a high-performance but limited-capacity solid state storage device.

FM must provide controlled access to all these spaces.

One of the keys to the implementation of FM is specialized code in the /etc/cshrc script, familiar to any UNICOS programmer. This script is always executed at initiation time for every job for which the user is entered as a "C" shell user. In addition, in each user's home directory, there is a ".logout" script which is executed when any job exits the system. These two scripts respectively set up and clear FM directories used during the job.

The /etc/cshrc script sets shell variables SSD, TMP, and FMHOME, which contain the pathnames of the three directories (in the SSD, scratch, and user spaces respectively) to which the user has access for the current job. The pathnames are constructed from the job's process ID to assure uniqueness of the pathnames. The "fm" program is then executed, for two purposes: to create the actual directories referred to by the SSD, TMP, and FMHOME variables, and to fetch a "loginscript" file belonging to the user. The user's home directory permissions are set to give read and search permission to the user only. Write permission is withheld to this directory; he or she is expected to write into the FMHOME, SSD, or TMP directories. Next the current directory is set to FMHOME, and the newly fetched "loginscript" (residing under FMHOME) is executed. At this point user-supplied code is executed, typically to execute additional FM operations to populate the user's directories with files needed during job execution.

It is the "loginscript" which gives the user the appearance of operating in a more typical UNIX environment. If the user puts appropriate FM fetches into the "loginscript", required files will be available at completion of the login process. The last command in a "loginscript" could be, for example, a command to enter the Bourne shell.

The ".logout" script performs an "fm -w" operation which stores in FM all files so designated by the user by a "store" directive. FM then removes all FM files from the three directories. Then commands in the ".logout" script remove the three directories and their entire contents. Afterward the user's UNICOS HOME directory is intact but empty.

The FM online space is defined within the same file system as all the user "home" directories. This facilitates assigning of files to users at fetch time and storing of files into FM at wrapup time. A simple "link" assigns the file to the user after FM establishes access permission. A "mv" at wrapup time removes the file from the user directory and adds it to the appropriate subdirectory under FM. The FM directory is a special path name in UNICOS. Under the FM directory is a UNICOS directory for each user. Under these directories are the actual files. There is one special UNICOS file, UCDIR, that serves as an FM directory. Each UCDIR contains information on each FM file belonging to the user. The other files under the individual user directory are the actual data files. The UNICOS names of the files are constructed from the classification, user name, and a seed in the directory header which ensures uniqueness.

The FM directories (UCDIR's) consist of a header and a number of entries. From a systems standpoint, one of the more important fields in a directory entry is the residences field, which shows where the file can be found.

There are three other important files under the FM directory. One is the media file. It contains information on each media set. Another is a UNICOS directory called tape_manager. Under tape_manager are files with names matching the vsn's of the individual volumes known to FM. The files are indices of the actual contents of the corresponding tapes. Finally, the "access_lists" file contains all the access lists that users have established to control individual user access to files.

There are operator utilities which can be run as needed to migrate files to offline storage. There are two types:

mandatory: stored FM files which the user has designated to reside on specific volumes or media sets must be collected periodically.

discretionary: whenever necessary, the disks are wiped of files in order to create space. Files are selected based on age and size. If a file selected for removal is not already on an offline volume, it is copied offline before removal.

XI. FM APPLIED TO WORKSTATIONS

To make FM a complete system, users on workstations must be able to retrieve and/or store FM files on workstations. UNIX lends itself to file transport since it treats all files as bit strings. There are network facilities which transfer files between UNIX systems. It remains, then, to supply the FM security features to a workstation environment.

The configuration which best supports a secure file management application on workstations is a server-based network. One or more servers have several individual stations attached to them via a local network. An individual station can have its own private disk pack; the owner is responsible for its security. For public files, there is a need for access controls. To accomplish this, a file system can be maintained on the server. A feature of server-based systems is that stations can be downloaded from the server. If that feature is made mandatory, then only software maintained on the server is available to individual stations. An "fm" program similar to "fm" on UNICOS is invoked to process directives similar to those on UNICOS. Files are maintained on a file system on the server with permissions restricted as on UNICOS.

Network files can be pure character files which need no conversion, or files containing binary data, which must be converted to reflect differences in architecture between systems. Such binary files contain information which drives such a conversion.

Parameters on store directives inform FM whether a generated file is to be passed on to centralized file store. Thus, the user has the option of making the file available network-wide or keeping it on just the server. Likewise, on a fetch, if FM fails to find the file, the centralized file store can be searched, and if found, the file can be transferred to the server.

Standardized network services between UNIX based systems for transfer of files can be used by FM in its file management task. Design considerations see to it that an FM file is transferred only by an FM program. The key to secure operation is that users are prevented from transferring files themselves. The prevention can be done two ways: either disallowing all transfers except by an FM program, or (less restrictively) allowing general transfer of files but not to or from the FM file system.

As in UNICOS, file migration is a major part of FM. The FM file system on a server will become full, and it is the task of the system administrator to invoke utilities which transfer old files to centralized store. The exact nature of these utilities can be tailored to the installation or even to individual servers.

XII. CONCLUSION

This paper outlines a file management system that was implemented on Cray UNICOS and ported to a workstation environment. The system provides need-to-know access control as well as automatic file migration. The system is self-contained and can coexist with other file access methods on the same system.

END

DATE
FILMED

12/27/93