Extended Access Control Facility (EACF) USER MANUAL

This manual is intended for use by those who will set up EACF and use it to control file access on a system.

Introduction:

EACF introduces a number of new capabilities to VMS. This manual is designed to: * explain what these are and what their intended use is, and * give information on how to mark files to achieve added capabilities.

Overview, or, Why EACF:

Work any length of time in a computer facility and you hear the stories: someone reading another person's mail, some clerk telling a friend how much the boss makes, some systems programmer typing out a document that contains long range company plans and was for Vice Presidents only. People are often curious and occasionally malicious. Regardless, the information loss can create serious problems, especially if it falls into the wrong hands, or lands you in court because of some breach of privacy.

Technical means to control this are often lacking. If you can only allow or disallow read/write access to a file by someone, as is the VMS norm, then (for example) the payroll clerk can edit the payroll database. Doing this with the normal payroll application, at his terminal during the day, might be perfectly OK. However, should he be permitted to access it with, say, COPY? Someone developing new software might need access to system areas to install his latest versions. Should he, however, have access also to the technical plans?

As these questions make clear, it is desirable for the system to limit access to sensitive files. By doing so, it gives useful warning when someone's curiousity gets out of bounds, prevents loss of information, and reduces organizational exposure to anyone giving in to temptation to sell company secrets, by making those secrets inaccessible outside of authorized limits.

Extended Access Control Facility is designed to prevent loss, damage or unauthorized disclosure of data by controlling disk file access. Most security breaches ultimately involve tampering with disk storage. Others such as login breakins are already dealt with fairly well by VMS if set up properly. However, VMS doesn't make it possible to regulate what can be done with highly sensitive data at all closely.

The difficulty of allowing access with certain programs to certain files, for certain people and times, has led to many incidents of VMS users browsing areas where they are not authorized by policy, but in which they have access because their privileges work in a coarse grained way. Often these are innocent curiousity. Sometimes they are industrial espionage. With most (some estimates are as high as 98%) of security breaches coming from "insiders" exceeding their authority, this exposure is a problem in any organization keeping data online which competitors shouldn't get at.

A number of products claim they prevent unauthorized file accesses in VMS. Unfortunately, all of them currently on the market, with the exception of this one, work by accessing data from the VMS AUDIT facility. The problem with this is that anyone even moderately familiar with VMS knows that the Audit facility keeps track of security violations, and there are numerous ways to shut it off tracelessly, including code from public domain sources. This renders the strategy of these packages ineffective.

What is just as bad, these products find out that an unauthorized file access has happened from Audit only after the fact. At that point, all they can do is (attempt to) stop the image or its process, since the file access has already happened, and often already succeeded. (Where the access is a delete, the file is already gone.) As a result, they require that a policy be in place to punish offenders after the fact, since they may have already done some damage. It's hard to be relaxed about a breach in such a case.

EACF, on the other hand, lives in the VMS file system and prevents unauthorized file accesses from happening in the first place. It is not circumvented by privileges, works whether the Audit facility is up or not, and can ensure that a protected file is NOT accessed where this is not authorized. Thus, the innocent browser cannot have done any harm, and may be treated with less rigor than might be needed if damage or data loss may have occurred. Also, while other packages have to use obtrusive measures like process deletion to stop accesses, EACF can silently hide files if that is desired. Where a real violation IS happening, this can be much more effective in finding out what is being attempted than stopping a process. EACF of course does generate logs of any detected violations so that notice of them is preserved.

EACF is uniquely valuable in controlling access by privileged users. Many sites have people who have elevated privileges, but who should not be able to access payroll, grade files, company plans, etc., and who are not prevented from this by VMS. EACF can protect against exactly such access (and even discriminate AGAINST people with various privileges if you like) and gives extensive controls over privileges and their uses above those VMS has.

Finally, EACF is simple to use. You mark files for various accesses with a menu oriented utility which selects how a file (or set of files) should be marked, or which will display what markings exist on a file. All security relevant information is present at one time in an easily understood format; you do NOT need to hunt through all disk file ACLs to find it, if it exists at all. No normal VMS facilities are lost, either; VMS security controls continue to work normally, though EACF may in various ways alter the outcome of their tests in understandable ways. (An example of this might be that if you attach an identifier TWIT to a script, that script may fail to open some files which have ACLs prohibiting access to identifier TWIT.)

If your business depends on any data on your VMS systems and your operation is more than a VERY few people, you NEED EACF.

Capabilities:

EACF offers the following abilities:

* You can specify a password for file access. This password must be entered before access is attempted, using an EACF utility for the purpose. Then if the file password has been entered,

and other EACF checks satisfied, the file may be opened (or deleted). Otherwise, access is denied. The password can be anything at all, and the tokens left in user space to indicate the entered password are also process dependent, so others cannot use them. The password can be changed at any time using EACF, but allows control via a separate piece of knowledge if this is preferred to control by time, username, etc. This facility has long existed in the mainframe world. Now it is available for VMS. While it is less secure than encrypting the file, this method also cannot cause a file to be lost forever if the one who knows the key dies or forgets it. Also, changing the file password is quick and easy, unlike changing its encryption, so that temporary access can be given and revoked at need.

* You can control access by time of day (in one hour increments). You can use this facility to prevent file access during "after hours" times where there is no reason to allow them to be accessed, or "open up" files during periods where normal controls are deemed adequate. The access modes for each hour can be:

> Y - Access permitted provided all other security checks pass.
> N - No access (except to users on backup list)
> U - No CHECKS, let everything by EACF checks on this
> file during this hour. (DEC security remains in effect.)
> P - Check privileges against max privs only
> R - Readonly access only, by anyone; r/w opens are
> rejected. Other checks may deny access totally as well.
> X - Allow r/o opens, test others normally.
> D - Disallow access by ANYone (even the backup account)
>
> (Note that provision for a backup account exists, and it can be forced to permit only read-only file access, but this facility allows further restrictions, operating per file.)
>
> Notice that this means that writing to a file can be disabled whenever one likes even if reading is left permitted. If there is a suspicion that some databases may be tampered with off hours, this gives a simple way to protect them. The privilege check mode is also of particular value, since it can disallow use of elevated privileges which might otherwise be used to bypass normal system protections by industrial spies or other evil doers.

* You can control WHO can access a file (with both permitted and forbidden usernames), WHERE they can access the file (with permitted and forbidden terminal names), and WITH WHAT they can access the file (with both permitted and forbidden image names). This is the major advantage of EACF. You have, for usernames, terminal names, and image names, two lists each. You can specify username strings that are acceptable or that must be rejected,or can use * to indicate all, or & to indicate to skip a test. Likewise you can specify permitted or forbidden terminal names (access port names are tried also if they exist), or image names that are permitted or forbidden. These are handled with the normal VMS wildcard conventions (except that in addition a string with & in it means to skip the test). Thus, to say that a given file cannot

be accessed with the COPY image one could add "*COPY.*" to the forbidden image list. (Note: "Access" as used for EACF means either open OR delete access. Renaming a file does not have any effect on the controls.

The beauty of this is that all the EACF parameters can be controlled in one spot, without huge ACLs and unmanageable machinery to fiddle with system logicals or identifiers on the hour. This permits really fine grained access to your files. If you have an accounting person who works on the payroll file, you can authorize access to PAYROLL.DATA during working hours, from that person's normal terminals, but forbid access after hours, or from terminals like RT* or NT*, and you can ensure that the file cannot be accessed with, for example, COPY or BACKUP, or perhaps can only be accessed from *PAYROLL*.EXE* if only a given suite of programs with such names should be used for access. The access lists are all variable length, comma-separated lists, so considerable flexibility is available. Finally, you can prevent access using privilege (see below) to this data, so that people who have privileged access to your system, yet have no physical access to the computer, cannot simply turn on privileges and read the data to their heart's content.

* You can prevent files from being accessed with more than a given set of privileges. You do this by attaching a maximum privilege mask to a file and if the process trying to open or delete the file has any privileges not in this mask, access is forbidden. The default mask forbids access using BYPASS privilege. This is highly useful against curious privileged users (and any intruders who gain privilege) since it means that files can be protected even where local conditions force operation with many privileged users, or where there is concern that intruders may gain privileged access to the machine. The tags that cause files to be checked in this way can be kept in kernel space as well, so that they are safe from being seen even by privileged users.

> NOTE: This and other checks are applied in addition to the normal VMS checking. Hence even a file owner is affected by this checking.

Many sites have people who must do development and need to be able to use INSTALL or other privileged functions, and whose privileges cannot be revoked without impairing their ability to work. However, most of these people have no need to be authorized to see, e.g., the payroll file. Conventional VMS offers NO way to prevent such access, and other file "security" packages work by watching the output of the AUDIT utility, which can be silently and untraceably shut off by publically available programs whatever someone is running to listen to AUDIT. As a result, there has been, until now, NO way to block access by these people to sensitive data left on normal disk files, whatever file protection is attempted. EACF, due to its residence within the file system, works whether AUDIT is turned on or not, and prevents these folks from using the file system to access such data. EACF offers the ability to either block access altogether, or to block write access only, and cannot be confused by command obfuscation, nor does it care whether the access is by program or by console command. Someone who has used privilege to change his username can be blocked by finding that the privilege he needs to continue to access his system tables denies file access. Thus even common tricks one might play to fool an access system into permitting access can be thwarted readily.

* You can hide files. That is, if access is denied, you can arrange that instead of giving the user a "privilege violation" message, EACF can arrange that another file (on ANY disk) can be opened instead wth no indication to a user exceeding his authority that anything unusual has happened. EACF will of course generate logs of the denied access, as it does in all cases of such, but the intruder will not know that his access attempt has failed. Therefore, you can arrange that he will open FAKE_SYUSUAF.DAT or FAKE_PAYROLL.DAT instead of the real SYSUAF.DAT or PAYROLL.DAT. This allows you to safely allow him to leave evidence of his intentions (e.g. let him try to enter system accounts, or see whose pay records he wants to see or tamper with) without exposing your real business data to tampering. In effect, the unauthorized user CANNOT SEE the real files, but "sees" other files which pose as them. In some cases this can be FAR safer than an error message, which will alert a person exceeding his authority or an intruder that he has tripped some protection barrier. Rather, it seems to pose no resistance to tampering, but gives complete resistance, while permitting evidence of what is attempted to be accumulated.

* You can attach a cryptographic checksum to the file. At any attempt to open the file, this checksum is compared with that of the actual data, and the file will not open if they do not match. This is used for detection of file tampering and means that if someone should tamper with a file, it cannot be used in such a state. This tamper-proofing works for any file type, and is not limited to executables. However, it does require that the file be read in its entirety to compute the checksum before an open can be done, so it is most effective for relatively short files. The most common examples of such are DCL scripts used by privileged users, which are frequent targets of attacks on system integrity. Certain file types (with very long maximum record sizes) cannot be fully checked in this way, but these are quite rare; the system provides for checks of files with records far larger than the longest permissible DCL line. If only a partial check is possible, EACF will check what it can. This facility can be used to guard many critical files against tampering which might otherwise be a first step in breaking into a system. Note too that the file size is used in the checksum, and this is tested first, so that anything that has changed the file size will fail the check very quickly. Where a file is over 100 blocks long, the script asks for confirmation before marking it for this check.

* You can attach a replacement privilege mask to a file which will replace all process privilege masks when the file is executed. This can be attached to any file, no matter how executed (so that interpreted scripts can be tagged too). This is normally used to ensure that a script or image never runs with elevated privilege. If you attach a privilege mask to, say, a DCL script or image you got from somewhere, and the mask contains no privileges except perhaps TMPMBX (and maybe NETMBX), then whenever it runs, the process' current and authorized privilege masks will be replaced with masks having only these privileges. (The privileges are restored on exit or when the file is otherwise closed.) Thus if any Trojan code would normally be triggered by the script finding that it had some elevated privileges, it would never find such, even if run from privileged accounts. Also, since the process lacks privilege while the script or image is running, it cannot abuse privileges that might otherwise be available to it. (The authorized mask is reset to keep Trojan horse applications from using $setprv to negate the privilege lowering.)

You should note that the UIC is not altered, so that implied system access via a low UIC would not be affected by this. Also, if you use this facility to raise privileges, be sure to allow E (execute) access to the file only and check the file carefully. (See below for a fuller explanation.) This facility can be used to attach privileges to DCL command files, but can open security holes if this is not done very carefully. Remember: if the file is opened, the opening process' privileges are affected. When used to reduce privileges, the facility provides a pro-active measure to inhibit viruses and other covert code from running.

Usage Note:

Privilege, base priority, and identifier modification poses few if any risks where these facilities are used to LOWER access privileges. In this case, someone opening a file which has these side effects only reduces his access. (This might even be handy for privileged people wanting temporarily to assume a nonprivileged identity without using another account.) However, any RAISING of privileges or access rights should be done this way only with great care. Anyone permitted to open a file marked in this way can in principle gain these added privileges/rights. In this case, other facilities of EACF should be used to prevent access to the file in undesired ways. For instance, by allowing E (execute) access but not read access, and enforcing this with a max-privileges mask that prohibits it from being overridden, and perhaps further limiting username and image allowed to do the opens (prohibit *DCL*, for example). In this way security needs can be served while privileges can be more completely controlled than vanilla VMS allows.

* You can attach identifiers to a file. These identifiers are granted on file open, revoked on file close. This can be used for any convenient purpose, though again they tend to make sense most for files which are executed. If used for data files, it is wise to use other EACF facilities to narrow the exposure these have. If for example you have an identifier BUS_PLAN that grants access to certain files, you can "attach" that identifier to an executable or script and allow it to access them. (DEC has something similar, but it won't work on interpreted code such as DCL scripts or interpreted languages of any kind. With EACF, you can apply protections regardless of language choice.)

* You can set the base priority of a process so that when a file is open, the base priority is replaced by the one attached to the file. (The priority returns to its old state at close.) This can be used to ensure that known "CPU hog" programs run with lowered base priority even if run interactively, or that some programs run with higher priority where this is desired. The first case is likely to be more useful. In some shops, elaborate systems to ensure that long compilations (e.g., large Ada programs) run in batch have been set up. This facility means that the CPU intensive programs do not gain extra priority from running interactively if they are set up with a base priority lower than normal. Likewise, some editors or screen intensive applications may

benefit from higher priority, and will not cause problems so long as they don't go into a compute loop.

NOTE

The three above modifications are flagged with cryptographic authenticators so that the tags on the files cannot be altered or moved to other files. A key that is used on loading EACF for each disk is also used in the computation, so such keys cannot be entered successfully unless one also knows this value. It is essential to system security that these key values be kept secret, and is desirable that the tagging program be available only to systems personnel. These modifications do not require any user-mode code to run, so impose negligible overhead.

* You can set EACF to have a set of "exempt" images which are permitted full system access. Access to these images can however be controlled by EACF. This permits EACF to avoid causing problems for disk defragmenter programs and the like, which must have a global view of the disk, without imposing large user burdens. Because access to these exempt images is still controllable, they can be prevented from causing security holes.

It is also possible, for example, to have a copy of BACKUP which is allowed to have global file access, BUT it can be specified that this BACKUP image can be accessed only by certain users, at certain terminals, at certain times, with certain privileges etc., and perhaps accessed only with certain identifiers which might be granted by attaching them to a backup script which grants the identifiers. The full panoply of EACF controls is available to screen who can use this copy of BACKUP. Meanwhile, another copy of BACKUP.EXE might be the one most users can access. The specially privileged copy would typically be accessed using a different verb, and could be used for making system backups.

This facility is essential to permit programs like disk defragmentgers to have global system access without interference or to allow system wide applications like backup to be run without interference when done by authorized people at appropriate places or times. Up to 32 image pathnames can be exempted.

In addition, an image is provided which can be used by a process to exempt itself from EACF checking or reenable it. This image must be treated with extreme care, but can be used for complex jobs where system maintenance is needed. When it is used, it logs its use so that such use can be audited.

* There is provision for a "privileged user" list (empty by default) which can be used to give certain users broader file access, limited only by certain time codes and privilege mask tests

(and DEC access controls), should such be desired. A supplied privileged utility can also be used to exempt a process from EACF controls where this is wanted for whatever reason. This is referred to in the menus as the Backup Account. It can also be prevented from write access to files while allowing read access.

* You can arrange for EACF to execute an arbitrary set of commands whenever access to a file is denied. The username, PID, terminal, and file ID are available to the commands (plus whatever else you care to make available). This can be used for mail or pager notice to systems people, for suspending the process, or for starting a keystroke monitor like SPY to record the process' activities, or for ANY other desired purpose. This takes place BEFORE the file can be accessed. (There is an additional capability, controlled by the command line language as described in *jtcommand.doc* which allows you to force execution of a site command script whenever an image tagged for this function is in use and opens a file. This is provided where it is needed to have additional controls over files which may be opened by a program whose security properties may be especially untrustworthy. The script runs whenever the untrusted program tries to open a file and it can veto the open. An application of this could be to alert a user of opens of files not in a permitted list. In this mode, or if in "check all opens" mode (see below) EACF will check the file's parent directory permissions if the file has no EACF security record.)

DISCUSSION

EACF is useful in relatively friendly environments, where it allows the limits of authority to be defined in the machine, so a privileged person who may be inclined to browse in public directories will be prevented from thoughtlessly printing private files with a message that he has exceeded his privileges, or in much more hostile ones where tampering or industrial espionage (which often uses the services of insiders who have been paid off to make (or have been tricked into making) a quiet copy of the payroll, or business plan, files) might be a risk. In either case, it is rather easy to limit access to sensitive files (or to directory files holding sensitive files) to certain people, and to remove the opportunity, and temptation, to disclose other data from people who might otherwise do so for whatever reasons. If, for example, your system manager should be curious about a file named "NEW_PRODUCT_PLANS.DOC" and tries to look at it, isn't it better that he be unable to do so (even though he may have BYPASS privilege) if company policy means he really shouldn't have access to it? Isn't HE safer with a system that will just block access rather than an alternative system that can't block access, but will get him in trouble after he DOES access it? Conventional VMS lacks the ability to do a detailed job of limiting access to such areas, so inadvertently this kind of thing happens a lot. With EACF installed, it is easy to have the machine understand, and enforce, the limitations of access company policy and safety demand. Thus, inadvertent as well as intentional access is controlled, and data access can be managed properly for the first time.

> EACF protections are in addition to normal VMS ones, and a person must be able to open a file to mark it for security. Henca all normal VMS group access provisions can be used to control which files a given group can mark.

WHAT IS HAPPENING TO FILES MARKED

You mark files in EACF using a menu-oriented file marking script, though a command line interface exists if it essential to use one. This marking results in an application ACE (the file mark) being attached to the file and in the EACF database being updated to reflect desired actions on the file. In addition, EACF startup can be set so that the server is passed a list of file IDs which will be inspected. This list may contain any number of IDs per disk and can be created from simple

```
$ DIR /FILE_ID
```

commands. The list is a sequential file each line of which contains a file ID in the form (n,s,r) where n is file number, s is sequence number, and r is relative volume number. For example the line

```
LOGIN.COM;85    (9493,12,0)
```

will do fine.

When a file is marked like this, one uses a /FILENUM:sequentialfilespec.typ switch to tell the server where the list of IDs is, and the server will then check extended access even if the file mark has been damaged or removed from the file. A file mark must have been created in the first place, so that the EACF database will have been filled in, but if a mark has been deleted, this will prevent such deletion from interfering with EACF checks.

There is an option in the menu system which checks that all files in the EACF database correspond to files with correct file marks and will regenerate these marks if any are missing. It also warns the operator of any files whose marks have been deleted, since this may represent someone having attempted to get around the EACF system's protection. Any unexplained cases of marks disappearing should be investigated as security breaches. If this happens, also, it indicates which files should be in the kernel-based lists, so that they will remain protected even if their marks are deleted entirely. There is no mechanism for removing files from the kernel based lists except a reboot and restart of EACF with a different, or no, /FILENUM switch on the command to start the server process.

The EACF setup script asks, for each disk selected, a security key (used as part of the authentication for all ACEs that change identifiers, base priority, or privileges) and a file containing these file IDs. Either, both, or neither may be selected. Note too that directories are files too, and can be protected, with the usual caveats. However, if the file [000000]000000.dir is protected, EACF protection criteria can prevent a volume from being mounted.

HOW TO SET EACF UP

Pre-Installation:

Before installing EACF, you should prepare for the installation by making sure that there is enough nonpaged pool available for the dynamic needs of the system. These vary somewhat but can be adequately approximated by increasing the SYSGEN parameter NPAGEDYN by the quantity (10000 * (number of EACF daemons you will set up + 1)). Insert a line something like the following

ADD_NPAGEDYN=30000

in your MODPARAMS.DAT and run AUTOGEN. You may of course have adequate nonpaged pool already (many systems will) in which case no modifications are needed. SHOW MEM/POOL will show this.

Your system disk must have about 5000 blocks free to install EACF, though if the EACF components are placed on a separate disk, system disk space required for the final resident components is under 500 blocks.

Installation:

EACF is installed by VMSINSTAL. From the system account, use a command

`$ @SYS$UPDATE:VMSINSTAL EACF010` distribution_medium:

to do the installation. The script will ask where EACF programs and scripts should be stored, and most EACF components will be placed there. This can be anywhere on the system, provided that the disk is mounted in your startup script before EACF is started. These components will occupy around 2000 blocks on a VAX, double that on an AXP, so ensure the disk chosen has room enough. When running the setup script after installation, the area chosen for the EACF components should be specified as the area for EACF executable images.

Post Installation:

EACF comes with a setup script which is to be run after the VMSINSTAL script has installed the package. This script (`JT_SETUP.COM`) generates a startup script which should be run from `SYSTARTUP_VMS.COM` or `SYSTARTUP_V5.COM` at boot time. The script generated will start one access server which will be attached to all disks in the system, and enough intercept driver units to control all disks selected. It is perfectly legal and permissible to have many EACF servers, each handling one or more disks, however. This allows faster processing of security markings in some cases. The script offers an option to select the starting intercept driver unit number, so that it can be run more than once, and will arrange to load a server for all disks in each run. Normally it starts with intercept unit 0, but if you connect three disks, you can run the setup script again and start at intercept unit 3 (since the prior run will have used 0, 1, and 2). Then with this run, another server will be started. It is up to you to ensure that disks allocated don't overlap.

JT_SETUP is fairly easy to understand. Its initial menu looks like this:

```
                   EACF SETUP 18:11:55

   --> *Set area to hold EACF database files
        Set start intercept driver unit number (now 0)
       *Set area for EACF executable images
        Done this menu, process disk selection
        Remove a disk from an existing EACF configuration
        Set images which are exempt from EACF (e.g. defraggers)
        Quit, do nothing

 ( Asterisk means that a value already exists for the item.)
```

It allows you to specify where EACF database files are kept (for the series of EACF intercept driver units you configure). Each run of JT_SETUP will let you configure one or more disk units, each of which uses a JT intercept driver, with its necessary data areas. Normally this starts with JTA0: and goes up in units, but if you want multiple servers you can run JT_SETUP once for each server and set the "start intercept driver unit number" item to the start unit number, which should be one higher than the maximum number of disk units configured so far. One JT unit can control only one disk unit. Asterisk implies that values already exist (perhaps from kitinstal) for these selections.

The "Set images which are exempt from EACF (e.g. defraggers)" item allows image device:[directory]filename.type;version strings to be specified for images which are exempt from EACF controls. Note that any such images can bypass all EACF actions, BUT you can and should use EACF to control access to these images.

When you select the "Done..." item, you enter a menu to select disk units to configure for this EACF server. It looks like this:

```
        EACF Configuration     Disk Selection
Use arrows to move to selection. Use RETURN to select.

  End disk selection
 _ARISIA$DKA700: VMSSYS
 _ARISIA$DKB0: USRBAK
 _ARISIA$DKB400: DWU
 _ARISIA$DCA0: XRP07
 _ARISIA$DCA2: XRP06
 _ARISIA$DCA3: CDI
 _ARISIA$DCA4: XTRA
 _ARISIA$DCA5: CDTPL
 _ARISIA$VDB0: TEST1
 _ARISIA$VDB1: TEST2
 _ARISIA$DKB200: <unmounted>
 _ARISIA$DKB300: <unmounted>
 _ARISIA$DCA1: <unmounted>
 _ARISIA$DCA6: <unmounted>
 _ARISIA$DCA7: <unmounted>
 _ARISIA$FQA0: <unmounted>
 _ARISIA$FQA1: <unmounted>
 _ARISIA$FQA2: <unmounted>
 Type H for help.      Currently on item 1 of 151
```

where mounted disks appear first on the list, followed by all other disk-class devices on the system. The display will scroll if more disks are there than will fit on a terminal.

Once you select a disk, you must enter a number of parameters for that disk. This is done in another sub-menu which looks like this:

```
EACF DISK SETUP 18:19:45

   --> Enter security key for disk
       Enter file of file IDs to load for kernel marking
       Examine only kernel-marked files for access tests
       Set to automatically kernel-mark all files on load
       (This requires generating the file list at boot)
       Generate filelist database and set to use it
       Act as if ALL files were kernel-marked (use if most are)
       Quit, do nothing.
       Done this menu
```

The security key for the disk is the secret string (of any reasonable length) which is used to validate EACF actions that alter privilege, identifiers, or base priority. It must match the marking on a file or these will not be altered (which prevents random people from creating such markings). Therefore this key should be well chosen and the EACF startup file carefully protected from browsers. Note that when a marking on a file is created which resets privilege, grants an identifier, or resets base priority while that file is open, a file security key is asked for. That key must match the volume key entered here or the changes to process priority, identifiers, or privilege are not made. This prevents unauthorized people from gaining privilege even if they are permitted to mark their own files with EACF.

The other options are pretty self-explanatory. If you can afford the time to create a file of IDs to load at EACF start, it makes the system MUCH harder to avoid and allows the option of only examining marked files, which reduces overhead on nonmarked files and is therefore a good idea too. Whether to use item 4 or item 5 depends on how often your files that need to be examined change. Item 4 (automatically kernel mark all files on load) generates a complete list of all files that are marked at every boot. Item 5 generates a list when you select it and uses that, rather than generating the list at every boot. This can suffice for fairly static lists of files to mark.

Once you finish with the menu, it will create EACF_STARTUP.COM and EACF_LOGIN.COM which are used to start EACF and to define some symbols needed for convenient use. EACF_STARTUP.COM should be run from systartup_v5.com or systartup_vms.com (depending on version of VMS) at boot time.

WHAT TO MARK

You will want to mark files (and perhaps directories, though remember that files can be opened by file ID without going through directories) which are important to your business with EACF. Transient files should ordinarily not be marked.

Critical system files should be marked. These include SYSUAF.DAT, SYS$SYSTEM:NET*.DAT, and the startup scripts (so people can't easily find out what nonstandard things you're running), as well as any other .COM files usually run by SYSTEM. Be sure, however, to allow access BY the SYSTEM account to these.

You should also use EACF to regulate the access to the files in the area you choose for your EACF data files. The files in JTD$DB and GCY$CM areas hold your security database and must be protected against access by unauthorized people. It is also often a good idea to access protect the file marking utility also (filename JTAUTHMAINT.EXE) so that only authorized people may mark files. Limiting access to the *.GDB files to the image JTAUTHMAINT.EXE (and including the whole path) is a good idea. Also, the scripts used to start EACF up, which will contain the volume access keys, should be protected against access too, so that users cannot enter file markings that alter identifiers, etc. If you use the SETUP script, the file to protect is

```
SYS$MANAGER:EACF_STARTUP.COM
```

and it should be treated and protected at least as well as SYSUAF.DAT. Note that the EACF setup script will, by default, protect the EACF databases against access from images other than EACF's images and BACKUP. It is recommended additional tailoring be done however.

HOW TO MARK FILES

You mark files in EACF by using the mark_files script JTMARK_FILES.COM. The MARKFILE command does this. (This presumes EACF has been installed and generally presumes a privileged account to assure the ability to tag files appropriately.)

A brief note first: permitted and prohibited usernames, imagenames, and terminal names are matched with string matching, so that to allow access by BACKUP.EXE, for example, you would use a string like *]BACKUP.EXE;*, since otherwise it won't match the full path name which is available for matching. Therefore most of the time your selection strings will contain the * character, and the file marking script will issue a warning message if this is not present. If your selection, for example, IS the complete pathname for an image, or the complete terminal name (or access port name) for an image or terminal respectively this is not a problem. In any case, your selection is used. If however you want to (dis)allow access by, say, COPY, you can do so by excluding *COPY* more compactly. There is a limit in total size available for this information, so conserving space is often a good idea.

Now to actual use of the script. In fact there are two scripts. One is named JT_SEL_MARK.COM and uses a fullscreen frontend as a selection front end. With this script, you set your default directory to the tree where you want to select files and run the script, using the arrow keys and spacebar to select/deselect files. When you are satisfied with the selections, type G to mark the selected set of files. The selection menu is the same for both scripts. The MARKEACF command runs the fullscreen - selection markup script.

The other script (jtmark_files.com) asks for a file specification explicitly, which may contain wildcards.

The start of that script looks like this:

`$ MARKFILE` File(s) to mark (? for dir):

At this point you select a file (or a wildcard file specification, in which case you will be marking each in succession). The selection could for example be:

```
$ MARKFILE
 File(s) to mark (? for dir): VDB0:[EVERHART]ND.TXT
```

[Alternatively one can just use a command like

```
 $ Markfile vdb0:[everhart]nd.txt
```

to accomplish exactly the same thing.]

At this point, if the file exists, you are presented with a menu which looks like this:

```
     Extended Access Control Facility – File Marking
        File VDB0:[EVERHART]ND.TXT;7
```

```
--> Edit max-privs-to-open mask (current: DFFFFFFFFFFFFFFF)
    Set base priority override (current: None)
     Edit permitted user list (*)
     Edit forbidden user list (None)
     Edit permitted term. list (*)
     Edit forbidden term. list (None)
     Edit permitted image list (*)
     Edit forbidden image list (None)
     Edit backup account list (None)
     Edit time-use profile (YYYYYYYYYYYYYYYYYYYYYYYY)
     Edit replacement priv mask (when file open) None
     Set backup user access [Readonly/Read-Write] R/W
     Set file tamper check
     Set file password Set AccFail-action cmd @GCY$CM:JTACTN
     Enter security key
     Set conditional softlink (None)
     Grant identifier on open/revoke on close ( )
     Quit, make no changes
     Done, update file controls now
     Perform database maintenance commands
     Apply selected mods to all wild files
```

In the above case, the settings are the defaults. The max-privs-to-open mask excludes access with BYPASS privilege but allows all others.

If we select the first item (by pressing <cr> (the RETURN key)) the system shows another menu which looks like this (showing all but the BYPASS privilege selected:

```
 Privileges Permitted For Opening File Privilege Set Up

 --> Done setup   * MOUNT     * U05        * U25
     Set all priv * OPER      * SECURITY  * U26
     Clr all priv * EXQUOTA   * U07        * U27
   * CMKRNL       * NETMBX    * U08        * U28
   * CMEXEC       * VOLPRO    * ACNT       * U29
   * SYSNAM       * PHY_IO    * U10        * U30
   * GRPNAM       * BUGCHK    * U11        * U31
   * ALLSPOOL     * PRMGBL    * U12        Set NO mask
   * DETACH       * SYSGBL    * ALTPRI    Make no changes
   * DIAGNOSE     * PFNMAP    * U14
   * LOG_IO       * SHMEM     * U15
   * GROUP        * SYSPRV    * U16
   * NOACNT         BYPASS    * U17
   * PRMCEB       * SYSLCK    * U18
   * PRMMBX       * SHARE     * U19
   * PSWAPM       * UPGRADE   * U20
   * SETPRI       * DOWNGRADE * U21
   * SETPRV       * GRPPRV    * U22
   * TMPMBX       * READALL   * U23
   * WORLD        * U04       * U24
```

At this point we can toggle any privilege setting by selecting it and when done select the "Done setup" item.

The "Set base priority override" item, when selected asks the question:

Enter base priority for process with this file open:

which can be answered by a priority between 0 and 31. If outside this range, or if nothing is entered, the priority setting is left unset, which means that EACF does not alter it.

The next 7 items:

```
        Edit permitted user list (*)
        Edit forbidden user list (None)
        Edit permitted term. list (*)
        Edit forbidden term. list (None)
        Edit permitted image list (*)
        Edit forbidden image list (None)
        Edit backup account list (None)
```

all lead to similar menus. If we select the "Edit forbidden user list" item, for example, we get the following:

```
     Edit Forbidden User List
     Current user list: None

     * is wild card, & means omit this test

 --> Add users to list
     Remove users from list
     Replace list
     Quit, leave user list alone
     Done with this menu
```

If the string when you are done contains a '&' character, the test corresponding to that menu is skipped. * on the other hand is a wildcard character. In several lists, a message is issued if no * is seen, since image paths or terminal lists will often contain them if they are to perform as expected.

NOTE: ***The permitted lists have priority.*** Thus if the permitted user list is *, everyone is permitted access to the file regardless of the forbidden list. If the permitted user list is & (or just something you won't get like "=") the forbidden user list controls who is denied access. If a few names are on the permitted list and the forbidden list contains *, then only the few named on the permitted list can access the file. Similar rules hold for image names and terminal (or accport) names.

Now if I select "Replace list" and add the value GUEST, the question and response look like:

Enter user list: GUEST

When we select the "Done..." item, we return to the top level which now looks like this:

```
Extended Access Control Facility - File Marking
    File VDB0:[EVERHART]ND.TXT;7
    Edit max-privs-to-open mask (current: DFFFFFFFFFFFFFFF)
    Set base priority override (current: None)
    Edit permitted user list (*)
--> Edit forbidden user list (GUEST)
    Edit permitted term. list (*)
    Edit forbidden term. list (None)
    Edit permitted image list (*)
    Edit forbidden image list (None)
    Edit backup account list (None)
    Edit time-use profile (YYYYYYYYYYYYYYYYYYYYYYYY)
    Edit replacement priv mask (when file open) None
    Set backup user access [Readonly/Read-Write] R/W
    Set file tamper check
    Set file password Set AccFail-action cmd @GCY$CM:JTACTN
    Enter security key
    Set conditional softlink (None)
    Grant identifier on open/revoke on close ( )
    Quit, make no changes
    Done, update file controls now
    Perform database maintenance commands
    Apply selected mods to all wild files
```

If we now want to control when this file may be accessed, we select the "Edit time-use profile" item and get the screen following:

```
 Time Usage Permission Editing (by hour)
.  Current Profile YYYYYYYYYYYYYYYYYYYYYYYY
                             11111111112222
                   012345678901234567890123
 Key:
Y - checked acc. N - No access except bkp D - no access at all
U - Unchecked acc P - Priv checks onlt R - Readonly access only
        X - R/W chks only

    --> 0-1AM   12N-1PM       Quit this menu, no changes
        1-2AM   1-2PM         Exit, apply changes
        2-3AM   2-3PM
        3-4AM   3-4PM
        4-5AM   4-5PM
        5-6AM   5-6PM
        6-7AM   6-7PM
        7-8AM   7-8PM
        8-9AM   8-9PM
        9-10AM  9-10PM
        10-11AM 10-11PM
        11-12AM 11-12M
```

To explain the key: Y means normal checked access, checking all security criteria. N means No access, except that the "backup" account may access the file. D means no access at all, to anyone. U means Unchecked access. The extra EACF tests are skipped. P means only the test that allows open if the process has only privileges also in the "Max-privs-to-open" list. R means Read-only access. Opens for r/w access are denied. X means no tests for readonly opens, extra EACF checks only for R/W opens to a file.

Each letter selects an access code for one hour of every day.

If I want to disallow use (except backup account use) of the file from 2-3AM, I select that hour and enter N. The selection question is as follows:

Give code for this hour access:: N

If I know that between 9AM and 3PM the file should be open to all there with no added checks for reading, but I want to prevent writing except for users allowed by the rest of EACF, I can use the X code for those hours. When I have done so the screen looks like:

```
 Time Usage Permission Editing (by hour).
  Current Profile YYNYYYYYYXXXXXXYYYYYYYYY
                            11111111112222
                  012345678901234567890123
 Key:
 Y - checked acc. N - No access except bkp D - no access at all
U - Unchecked acc P - Priv checks onlt R - Readonly access only
        X - R/W chks only

        0-1AM   12N-1PM         Quit this menu, no changes
        1-2AM   1-2PM       --> Exit, apply changes
        2-3AM   2-3PM
        3-4AM   3-4PM
        4-5AM   4-5PM
        5-6AM   5-6PM
        6-7AM   6-7PM
        7-8AM   7-8PM
        8-9AM   8-9PM
        9-10AM  9-10PM
        10-11AM 10-11PM
        11-12AM 11-12M
```

Notice the Current Profile now shows X access codes between 9AM and 3PM (and the cursor has been moved by hand to the exit item.) Now we can exit this menu and see that the time access profile is set.

Where the file is not executable, a replacement privilege mask makes no sense for it, but if we wanted to set one the screen would look like this:

```
 Privilege Mask to Replace all priv masks
 Privilege Set Up

    --> Done setup   MOUNT      U05      U25
        Set all priv OPER       SECURITY U26
        Clr all priv EXQUOTA    U07      U27
        CMKRNL       NETMBX     U08      U28
        CMEXEC       VOLPRO     ACNT     U29
        SYSNAM       PHY_IO     U10      U30
        GRPNAM       BUGCHK     U11      U31
        ALLSPOOL     PRMGBL     U12      Set NO mask
        DETACH       SYSGBL     ALTPRI   Make no changes
        DIAGNOSE     PFNMAP     U14
        LOG_IO       SHMEM      U15
        GROUP        SYSPRV     U16
        NOACNT       BYPASS     U17
        PRMCEB       SYSLCK     U18
        PRMMBX       SHARE      U19
        PSWAPM       UPGRADE    U20
        SETPRI       DOWNGRADE  U21
        SETPRV       GRPPRV     U22
        TMPMBX       READALL    U23
        WORLD        U04        U24
```

The privileges selected will replace all a process' privileges which has the file open, until the file closes. Note the "Set NO mask" item means that you don't want privileges to be modified when the file is open. . Note that both current and authorized privilege masks are replaced. This prevents anything from setting privs with the $setprv service, since the authorized privilege mask is replaced also.

If we want to set a file password, we select the menu item and get a prompt thus:

```
Enter file password: Thisisaprettylongpassword
```

The password will not echo, and you are asked for it twice to ensure you had no typing mistakes.

We can enter a security key; this will allow privilege, identifier, or base priority changes to work if it matches the one used for the disk in the setup script. This prevents random people from marking their files and altering privileges. The question asked looks like this:

```
Enter security key:
```

Again, the security key does not echo and is asked for twice to ensure it is read properly.

Hiding Files

Also we can arrange that another file be opened if the access is refused. suppose we want EACF to force an open of VDB2:[EVERHART]LOGIN.COM in the case access is denied. We select the "`Set conditional softlink`" item and get a prompt. With our reply it looks like this:

```
Enter device:file to open if access denied: VDB2:[EVERHART]LOGIN.COM
```

Action Commands

You will notice the menu item "Set AccFail-action cmd" here. This is a command which can be executed whenever someone tries to open the file and is rejected. This command runs in a spawned procedure and can be any command at all. If the command is terminated with a "\" character, it is executed with no arguments. Otherwise it is passed 5 arguments, namely the username of the rejected accessor, the terminal name, the file ID (as two hexadecimal longwords), and the PID (Process ID) of the process whose access was rejected. An example routine named JTACTION.COM is furnished which just sends email to SYSTEM when such an access is attempted. Any desired commands are permitted, though you should be aware that access is delayed till the command finishes. Hence any long actions may be best placed into SUBMIT commands. This mechanism can be used to provide for prompt notice of attempted file access by whatever means are convenient, to suspend offending processes, or whatever else is desired. Note that the commands can be different for each file, permitting tailored responses. The default is no such commands are executed unless you enter one. The current selection, if any, is displayed on the menu. It is suggested that options such as turning on session recording via a SPY type program could be one useful action to enter here.

Automatic Identifiers

If we want an identifier on the file while it is open we can grant one with the menu item. Suppose we enter the identifier FIELD. The prompt and response look like:

```
Enter identifier to associate with file: FIELD
```

When done with all changes so far (including setting a priv mask on open that has TMPMBX and NETMBX in it and nothing else) the menu looks like:

```
     Extended Access Control Facility - File Marking
File VDB0:[EVERHART]ND.TXT;7
Edit max-privs-to-open mask (current: DFFFFFFFFFFFFFFF)
Set base priority override (current: None)
Edit permitted user list (*)
Edit forbidden user list (GUEST)
Edit permitted term. list (*)
Edit forbidden term. list (None)
Edit permitted image list (*)
Edit forbidden image list (None)
Edit backup account list (None)
Edit time-use profile (YYNYYYYYYXXXXXXYYYYYYYYY)
Edit replacement priv mask (when file open) 0010800000000000
Set backup user access [Readonly/Read-Write] R/W
*Set file tamper check
Set file password Set AccFail-action cmd @GCY$CM:JTACTN
*Enter security key
Set conditional softlink (VDB2:[EVERHART]LOGIN.COM)
  --> Grant identifier on open/revoke on close (FIELD)
Quit, make no changes
Done, update file controls now
Perform database maintenance commands
Apply selected mods to all wild files
```

If we select Quit (to abandon the marking) or Done (to perform it) we eventually get another prompt:

```
File(s) to mark (? for dir):
```

and can either enter a filename or a return to exit the script.

Note that if you have not entered a security key, the selection of base priority, replacement priv mask, or identifier will not be effective. The security key must be entered, and must match that entered for the volume at EACF startup to be effective. This protects against unautorized file markings causing security problems. If any of these are entered and no security key is entered, you will be warned and given the chance to go back.

The "Apply selected mods to all wild files" is used only where the file specification given is a wildcard one, and means that the markings just set up for the first file will be applied to all files in the wildcard specification. If this is a huge number of files it may take some time. If you have marked files in the fullscreen front end, this option will apply the selected controls to all files you selected. Thus you select files, then set up access controls, and then apply the access controls to the whole list for simplest use.

The "Perform database maintenance commands" option selects various EACF maintenance operations. The menu it brings up is the following:

```
Extended Access Control Facility
MAINTENANCE COMMANDS

Delete this entry
Remove all database entries for now-deleted files
Create listing of all files marked
Repair deleted ACEs (ANY such are a security issue!)
Change or delete a string in all lists
Done with this menu
```

"Delete this entry" means delete the currently selected file's ACE and database entry, removing it from EACF protection.

"Remove all database entries for now-deleted files" means to go over the whole database and delete any entries referring to files which have been deleted. This should be done periodically to ensure that the database file is not cluttered with old information. It is handled in this way so it will work even if files are deleted with EACF disabled. (If the system logical GCY$DELMNT is defined to be "OK", then EACF will delete database entries when files are deleted to reduce the need for this maintenance.)

"Create listing of all files marked" will create a sequential list file of all marked files in the database in the format

Filename (fileid)

suitable for the /FILENUM command's input. This can be used for any desired purpose and is created as JTD$CM:OP<device>.GDB_LIST. The DCL symbol JTA_LISTFILE also is set to the filename value. The intent is that this file is used to set up the map of all marked files used in the /FILENUM switch (see above).

"Repair deleted ACEs" goes over the database and checks that the correct EACF ACE (file mark) is in place on each file. Any files whose ACEs have been deleted are possible evidence someone was trying to access a file EACF denied access to. Since the ACEs are marked hidden and privileged, this generally means a privileged user. Warnings are output to the OPCOM log and the user terminal if this is seen. The ACEs are then reconstituted. Running this item is a good idea also; it should be done periodically to ensure that protected files STAY protected. Files attacked in this way are protected anyway if they are in the list supplied to the /FILENUM:file-list-name.typ command. If they were not, they may have been compromised and should be checked carefully. If they were, EACF protection was not lost, but it is likely someone tried to break the EACF protection.

****

These kinds of events represent security related attacks by privileged users. They must be taken VERY seriously. EACF output to the operator log will give more information about any denied accesses if the /FILENUM feature was used so that prot, ection was not actually lost. Also this will show potential earlier attempts to access the file from before the ACE was deleted.

"Change or delete a string in all lists" is used, for example, to remove a username from all lists in the system requiring only one command per disk. It globally replaces one string in the lists with another. It should be recalled that since this is literal, if you are, for example, renaming a terminal server, you must replace the selection strings actually in the database, which are likely to be of form ",*name*" rather than plain strings. The menu front end is such that the "‘" character (accent grave) cannot appear in either string. The underlying utility can use any delimiter. However, this character is illegal in all the selections likely to be there. The string changed cannot be null. However, it is possible to add a user to a group by replacing, for example "USER1" by "USER1,USER2". This can be useful.

COMMAND - LINE USAGE

EACF defines several commands discussed more fully in the file JTCOMMAND.DOC, which permit use of and control of EACF from DCL command lines. Among them are the following commands, mentioned briefly here.

* MARKFILE - Set EACF security markings on files as discussed above. This uses the command file JTMARK_FILES.COM and allows marking one or a collection of files, or editing their security profiles.

* MARKSDH - Enter or edit security markings on files. This differs from MARKFILE in that a fullscreen file display is presented first to allow easy graphical selection of files to operate on, with the assumption that the files are in the current default directory tree.

*FILEPASSWORD - This allows one to enter passwords for files protected by password controls. It should be used by a process before attempting to open such files. If the password entered for a file is correct, this will permit access to the file.

* JTEXEMPT - This command has two forms:
  JTEXEMPT JTAn: and
  JTEXEMPT/DEAS JTAn:

and serves to exempt the current process from filtering by EACF, DPS, and HSM in the first form, or to re-enable such filtering by the second form of the command. The disk associated with the intercept driver JTAn: is the one to which access is affected. Whenever this command is used (and it should be used very sparingly, and protected against unauthorized use) it will log the fact of its use and who used it so that it cannot be readily used silently. Note that if the JTEXEMPT image is access controlled by EACF, additional controls on its use can be enforced.

The system also defines the verb JXSET which can exempt a process from controls. It should be made available only to trusted individuals, however. Its use is logged.

PERFORMANCE ISSUES

Highest performance at runtime is obtained by using the maintenance commands to create a list of all files in the database and loading this into the EACF database in kernel mode, selecting the option that tells EACF only to inspect files marked in kernel. In this mode, essentially the only files which incur more than a few extra instructions overhead at runtime are those known to have markings on them; the rest are quickly passed over. If however the number of files so marked exceeds around 10,000, it is probably best just to let EACF inspect all files for markings. As long as there is any header cache at all in the system in that case, the extra header access done to read the ACL will only use the cache, and not cause another disk access. Also, the disk containing the EACF databases should ideally be as fast as practicable (and remember: EACF should also be used to protect those database files from tampering). Finally, the EACF servers can profitably be given higher than normal priority, and should be. They are I/O intensive when used, and system responsiveness will be improved if the EACF servers are run at software priority greater than the normal 4.

Even if EACF is checking all files, system access time will not be noticeably affected save for those files which have extended checks, and in those cases the checking may add time. The longest operation is the file integrity check. This check requires, if the file length has not changed, that the file be read completely before opening by the user, to ensure that it has not been tampered with. This is reasonable for short scripts and the like. It should be done with caution for 30000 block images. (If file length changed, this is noticed at once.)

Other operations take negligible time.

EACF can be set up with as many servers as desired, from one for all disks in the system to one per disk. Basically, the more servers there are, the more opens with EACF checks can run in parallel. The servers are however activated only where it is known there is a file marked which needs server access. ACEs are read from a kernel thread, of which there can be many active at a time. Accordingly one EACF server is likely to suffice for most systems.

USAGE NOTES

To show some of the uses of these capabilities, let's take an example situation.

Let's say we have several files whose contents are confidential (say, a business' general ledger) which need to be manipulated by a few programs which we'll call GL, AP, AR, IN, and PR for short. What we want to avoid is that anyone browse these files.

One way to do this is to create an identifier (call it GLG) and set up ACLs on all the files which grant access to the files to holders of identifier GLG, but deny access to all others. Then we arrange that each of the 5 images has the GLG identifier granted when it is run (the programs are set for E (execute) access only) and now the programs in question can get to the files.

IF nothing else in the system ever gets the identifier GLG, and IF nobody uses privilege, we're done.

Note however that there are two disadvantages:

1. The control is scattered around, so you need to do considerable poking to find that nobody else might have the GLG identifier normally, and you must set the ACLs on all files that the

GL, AP, AR, IN, and PR access.
2. Anyone with privileges can override the controls.

Note that you can fix objection #2 by prohibiting access to any process holding READALL or BYPASS privileges. Ensuring that nobody has the GLG identifier requires separate examination, however.

If it is now decided that access at night to these records should be prohibited, EACF can simply be set to disallow access during those hours to the files, or possibly to the GL, AP, etc. programs that access them. Notice that with pure ACLs this is quite awkward, and blocking privileges is impossible. With EACF, it's easy.

However, there is a neater way to accomplish this.

On each file to be accessed by the GL, AP, AR, IN, or PR programs, have a permitted image list containing these image names, and a forbidden image list of * (everything) so that only those images can access the files in question, regardless of privilege.

Note that now no identifier needs to be created, and the files used by the programs are clearly marked so that it is obvious by looking at them using the marking script that only the selected images have access. Moreover, this cannot be overridden with privileges.

Notice too that preventing access at night to this data is done in the same records, and in these cases, you protect the confidential data, not necessarily the programs (which might be run at night on test data). All the security relevant controls are visible in the file marking display.

It may occur to some to ask how the data gets backed up now. One can of course set certain times of day for uncontrolled access to the files in question. A simpler way is to have a special copy of BACKUP that is permitted unlimited system access.

The CLD for such a copy, assuming it is called "SYSBACKUP" would be as follows (using the VERB utility to extract the original BACKUP command language definitions):

```
define type ENCRYP_VAL
  keyword NAME  value (type=$outlog)
  keyword ALGORITHM  value (default="DESCBC",type=$outlog)
  keyword VALUE  value (list)
 define syntax BACKUP_LIST
  parameter P1 , label=INPUTS , prompt="From"  value (list,type=$infile)
  parameter P2 , label=OUTPUTS , prompt="To"  value (list,type=$outfile)
 define type MEDIA_FORMAT_TYPE  keyword COMPACTION  negatable
 define verb SYSBACKUP  image SYSBACKUP  parameter P1 , label=INPUTS , prompt="From"  value (required,list,type=$infile)
  parameter P2 , label=OUTPUTS , prompt="To"  value (required,list,type=$outfile)
 qualifier ANALYZE  nonnegatable
 qualifier ASSIST  default
 qualifier BACKUP
 qualifier BEFORE  value (required)
 qualifier BLOCK_SIZE  value (required)
  qualifier BRIEF  nonnegatable
  qualifier BUFFER_COUNT  value (required)
  qualifier BY_OWNER  value (type=$uic)  placement=local
  qualifier COMMENT  value (required)
  qualifier COMPARE
```

```
qualifier CONFIRM
qualifier CRC  default
qualifier CREATED
qualifier DELETE
qualifier DENSITY  value (required)
qualifier ENCRYPT  value (list,type=ENCRYP_VAL)
qualifier EXCLUDE  value (required,list,type=$infile)
qualifier EXPIRED
qualifier FAST
qualifier FILEMERGE  nonnegatable
qualifier FULL
qualifier GROUP_SIZE  value (required)
qualifier IGNORE  value (required,list,type=$infile)
qualifier IMAGE  qualifier INCREMENTAL
qualifier INITIALIZE
qualifier INTERCHANGE
qualifier JOURNAL  value (type=$outfile)
qualifier LABEL  value (required,list)
qualifier LIST , syntax=BACKUP_LIST  nonnegatable  value
qualifier LOG
qualifier MEDIA_FORMAT  value (required,type=MEDIA_FORMAT_TYPE)
qualifier MODIFIED
qualifier NEW_VERSION
qualifier OVERLAY
qualifier OWNER_UIC  value (type=$uic)  placement=local
qualifier PHYSICAL
qualifier PROTECTION  value (list)  placement=positional
qualifier RECORD
qualifier RELEASE_TAPE  nonnegatable
qualifier REPLACE  qualifier REWIND
qualifier SAVE_SET  placement=local
qualifier SELECT  value (required,list,type=$infile)
qualifier SINCE  value (default="TODAY")
qualifier TAPE_EXPIRATION  value (required,type=$datetime)  placement=local
qualifier TRUNCATE
qualifier VERIFY
qualifier VOLUME  value (required)
disallow (ENCRYPT.NAME and ENCRYPT.VALUE)
disallow (DELETE and RECORD)
disallow (DENSITY and MEDIA_FORMAT)
```

Now one adds the path to this copy of SYSBACKUP.EXE (a copy of BACKUP.EXE) to the list of exempt images in EACF. (The correct full path is shown in SHOW PROCESS/CONTINUOUS when the image is active for the process being shown.)

In so doing, now, the SYSBACKUP verb can be added to DCLTABLES and used for making system wide backups. Note too that EACF can be used to control access to the SYSBACKUP.EXE image, permitting access to known people at known places and times. It is possible, too, to deny access to anyone with privileges to the SYSBACKUP.EXE image and have EACF grant the privileges itself (so that the "maximum privileges to open" mask would have only a few privileges, and the "replacement privilege" mask might have all privileges set). [One can also just use a VMS INSTALL to install the image with privileges.] In this way it can be made exceedingly difficult for anyone to use this system backup utility for unauthorized access. It also makes it fairly simple to allow an otherwise unprivileged operator account to perform these backup duties without granting the account privilege. By limiting the ability to use the program to a NONprivileged user, you make it much harder for anyone to gain "spoofed" access.

The security model supported here is much richer than the default VMS model, and while it can be enhanced, it covers a great deal of ground in controlling who has access to what files.

Hopefully the above discussion is helpful in describing some of the ways in which EACF facilities can be used.

Appendix - Extra Odds and Ends

(The intercept driver is also able to be set to prevent file opens where a volume is allocated or mounted privately. Normally such volumes' files are private but the privacy can be overridden in normal VMS with the SHARE privilege. This option allows the volumes to remain private even so. Note however that this can mean a print spooler will not have file access, and may inhibit other actions, so that this is desirable only some of the time. Access to disks mounted /system or /cluster is not affected. This operation mode is not currently supported by the setup scripts however, as it is somewhat esoteric. One sets the 2048 bit of the function mask to enable this. One also can enable fragmentation reduction by setting the 2 bit. By setting the 262144 bit, one tells the driver to only examine opens of files that have had their file numbers loaded into memory. This may sometimes be wanted for performance reasons. These and some other more esoteric functions are built into the driver and may be used as needed. The file *jtcommand.doc* gives information on the command language available in EACF and on some other capabilities not shown via the menu interface in V1.)