

# Techniques for handling mobile code and other OS access control issues

*Source:* <http://www.derkeiler.com/Mailing-Lists/securityfocus/secprog/2001-07/0000.html>

---

**From:** Everhart, Glenn (FUSA) ([GlennEverhart@FirstUSA.com](mailto:GlennEverhart@FirstUSA.com))

**Date:** 07/26/01

Date: Thu, 26 Jul 2001 10:25:52 -0400

From: "Everhart, Glenn (FUSA)" <[GlennEverhart@FirstUSA.com](mailto:GlennEverhart@FirstUSA.com)>

Subject: Techniques for handling mobile code and other OS access control issues

To: "'[secprog@securityfocus.com](mailto:secprog@securityfocus.com)'" <[secprog@securityfocus.com](mailto:secprog@securityfocus.com)>

Message-id: <376986A7203BD511B51100508BB88C338FB9E7@swilnts802.wil.fusa.com>

Dear secprog list:

Note in the following: the description at the bottom originally was intended to describe a product, but the product is now free and available in source. The ideas discussed hope to elicit comment and suggestions for how one should handle access control issues in unix and other OSs; in my (not so copious) spare time I am working on a followon...

Glenn Everhart

(personal replies preferred to [everhart@gce.com](mailto:everhart@gce.com))

Thanks

Security OS Design Issues and Thoughts

Glenn C. Everhart

July 2001

Smyrna, Delaware 19977

([Everhart@GCE.Com](mailto:Everhart@GCE.Com))

In spare-time work I have been designing what I think of as an access control add-in for unix, to be based on the notion of having system objects (mainly files) tagged with extended access permission flags which will control usage and are meant to allow efficient detection and prevention of misuse, and a sensible way to react to mobile or covert code based on noticing when the code ventures to touch system objects tagged as sensitive.

This stems from earlier design work for a VMS version, which was implemented some years ago and forms a starting point for later design. I want to step back, though, and discuss design principles under consideration, and offer them for consideration and reaction by others. Since there is some related work now going on, I hope that some light can be shed into the best strategy for dealing with mobile code and access control issues generally. As background I have been able to use multiple pattern searches, in series, to automatically

locate sensitive data in shared storage. This has used pattern density as a discriminator and works well enough that I believe that at least some global, automated classification of sensitive information is possible. Since the classification is highly business specific I do not include it in the design plans, save by leaving places to use its results.

I have observed with interest also the publication of descriptions of a "SubOS" facility in unix as a way of dealing with the security of mobile code. I have been dealing with related issues for some time, having implemented a system on VMS in the mid 1990s which allowed me to tag files with security tags which could effectively change the ownership, privilege levels, and running priority of processes which accessed files.

(Aside for those not familiar with VMS)

It must be realized here that processes in VMS are far more persistent than they are in Unix, on the whole, that granting an identifier to a VMS process is a close analogue of the setuid operation in Unix (it can be akin to a setgid also), and that VMS has a large number of privileges which govern where potentially dangerous things are permitted to happen, instead of an all-powerful root account. These privileges control net access, many aspects of device access, what functions can be sent to any device, whether the process may enter any elevated hardware privilege modes (most commonly exec or kernel), whether file protections can be bypassed, whether diagnostic functions are permitted, whether temporary or permanent mailboxes can be created, and many more. They are not completely orthogonal; someone who has cmkrnl privilege, for example, may run a program which enters kernel mode, and from there the program may set other privileges. However, absent such activity, a program needing kernel mode for some special device control need not have any elevated privileges with respect to filesystem access. This has saved many a VMS programmer from accidentally clobbering filestructures in privileged code.

(end of aside)

While the Safety description below describes what I had in 1995, there are a number of further developments that I want to incorporate into followon designs. Even though I have published the entire package now freely (including sources) on DECUS distributions and given up on selling it, I am working on a Unix analogue and feel that design goals for doing OS security effectively are worth discussing. Rather than re-describe the old work I have let my old description stand.

One will also note that the bundle of functions in the Safety package is rather thick. This is deliberate. I believe that where useful functionality can be inserted naturally, the final system will be more stable with it done in one place than it might be with the functions separated. In thinking as a systems, not just a security, designer

it is then possible to cover borderline cases which separated designs cannot cover. Finally, user acceptance of a package requires that there be something "in it for the user". In the case of Safety, the existence of a user undelete, and to some extent the storage hierarchy functions, make it useful to install even if the security specific functions are to be turned on slowly. Therefore my hope was that Safety would be a function enabler, and seen as such, rather than simply another external control on people. From the strict "security" point of view it doesn't matter. From the more practical point of view of actually getting the package approved for installation it matters a lot.

The world has changed since the early 90s though, and my sparetime unix version work will need to incorporate some additional functions and considerations. In the hope of stimulating reaction to the security protection model here, let me list some of them.

- \* First, and most obvious, the information gathered within a single machine here must be extended to all of a network, so that protection based on time of day or on which code is running is meaningful even across machines.

- \* Second, the lists of individual accounts for file access need instead to be lists of roles. The added indirection is a practical necessity for administration.

- \* Third, the protections in Safety form a single "tuple" defining a set of accesses. An object's protection in full generality needs to be a number of tuples. It may be one only, but must be able to be more than one.

- \* Fourth, creation of an object needs to inherit its protections from its ancestor directory or from its creating process, whichever is more stringent. I have determined that automated searches for important data can be successfully carried out via automaation, so errors in this classification can be overcome. A piece of covert code which creates new programs, however, must find that the new programs also enforce the same limitations. I have had code to implement this in hand since 1996 (though it is a tad inefficient; it uses the daemon process for the filesystem to do the setup from user mode) but did not productize it because of the low interest in VMS security apps from a homebrew shop.

I now view this inheritance as fairly important. It is simple with "paranoid mode" to in effect mark a program as suspicious, or to arrange for the paranoid flag to be set when sensitive data are accessed. Mobile code has proven inventive, though, at creating new files and new programs and using facilities for running such programs which are not directly connected with the creating process code. Therefore it is essential that where suspicious activity or origins are noted, that the taint of such origins be propagated to anything written by a process with that taint. Yes, this is

the good old star property, but without NECESSARILY having the star property mean absolute prohibition of access. It may only limit rates of access (see below). There should be a marking on any file that specifies whether this inheritance should take place, so that for example known innocuous areas might not trigger the added checking. (Note that in VMS, the checks are per device, and different rules per device may be adopted. You may want to simply disallow writing to the system disk to programs resident elsewhere, for example. In unix, I would allow checks to be sensitive to device also.)

\* Fifth, access rates must be controllable. Higher rates than authorized should slow access down or possibly block it; slowing down is preferred since that is the normal response to covert channels and prevents a process from crashing which may be only a little out of spec. The rate of increase of delay is TBD, or maybe to be set by the site. Add a little constant delay and you just slow things down. Double the delay with every new read and you limit total access very effectively.

\* Sixth, all checks should have "Warning mode" function so that it is possible to set the entire system up and have only warnings when something would be altered, not actual prohibition. This sort of functional control is for the most part already present in Safety, though it is considered an "internals" feature and I have only sketchily published it apart from the source code by this time. (Publication took place on DECUS SIG tapes, available from volunteer chain or made available by the DECUS library when each volume came out. The DECUS SIG tapes come out twice a year, and have done so since the Spring 1979 issue. The Spring 2001 issue came out in May, 2001 and is available on the Internet at a few sites.

\* Seventh, my existing code to do full softlinks in VMS should be productized and extended so that various types of conditional softlinks should be offered. At present one can get Safety to open one file if access is allowed, another if not. This should be extended so that certain identity flags can cause softlinks to be done if they are seen, linking to anywhere else visible to the system. The identity flags will be potentially inheritable like other attributes. Various different system check success or failure will deterministically be able to set identity flag number bits so that the access check results will determine which of several possible files may be accessed. There will also be ways to set such flags in a process (process tree in unix) so that whole directory trees could appear or disappear depending on what was running or who was running or what the behavior was of what was running.

\* Eighth, and further out, the facilities here should be extended so that they get integrated with directory lookup, so that directories

on disk contain not only filename and file location, but content and security information. Schematically this would use a DBMS to hold the information, use syntactic sugar in directory paths to specify SQL type queries a piece at a time, and potentially replace the normal directory operations altogether. A VMS disk would have only the [000000] directory and many files in the index file, but no directory files at all, and a unix disk might have only / and a lot of files with inodes but no directory entries. This will be important someday in having the computer help you automatically find things. One would use site indexing systems to fill in the DBMS and by default would select files that were in the current directory (I would keep directory trees) and had not yet been categorized.

Thus one might have a file whose path looked like

```
/foo/bar/$${CONTENT="payroll"}/$${CREALVL=19}/mumble/pays.txt
```

where the stuff inside \$\$ {...} meant queries that would be given in addition to the query inside the mumble directory to find a file "pays.txt". Directory listings would do these queries too, which could mean the file contains the string "payroll" and was created by someone of pay grade 19 or above. The ability to interpret the sugar and pass the rest on as normal directory operations would be in something just ahead of the filesystem and would allow existing programs that understood file paths to be used unaltered. Because a newly created file would not in general have all fields set, it would be left as "not yet tagged" and included in directory listings by default.

Safety keeps its extra info in its own files, and I would continue that. It is not necessary and not desirable to have to bend the on disk format of filestructures to hold the security info. My measurements indicated a 1% degradation in open speed with Safety (doing nothing but opens!) even where every file was being examined for markings without benefit of bitmap. This level of overhead can be bettered but is not bad. Note too that Safety access failures are a high signal to noise level indication of intrusion or misbehavior. I would use this fact to feed data to such engines rather than attempt to parse audit logs in real time. By flagging mobile code as in effect becoming someone else, a someone less trusted than the person at the machine, and making these markings be inherited by anything untrusted processes create, we ensure that the default behavior of a program that is not locally known need not be trusted with much of the system.

(Paranoid mode was initially designed so that you could watch what areas were normally accessed by a program and block other accesses. The inheritance of other protections is however a better scheme. Tagging any program that will access the net and requiring files to be separately declared not hostile before they may access anything but minimal scratch areas of

the system gives a system that is much less in need of constant tweaking.)

\* Ninth, I would add direct control over non filesystem objects to the monitor. While control of privilege mask does this indirectly, a more direct control would allow access to set mask bits. In VMS this would mean trapping \$assign/\$dassign. In unix it means trapping the opens to sockets. There would be less finegrain control here, but at least the open/close type operations would be there to set up additional access conditions as though the sockets (or channels in VMS) were files. Disallowing access, faking errors, or allowing redirection to other channels would be possible extensions.

\* Tenth, I would improve the logging of errors in variety and direction, allowing encrypted logs directly (and not just using encrypted pseudo-WORM disks; unix might not have these) and allowing alerts to go to possibly several places rather than one only. (Only one set of routines needs to be altered to expand this...it is well encapsulated now.)

All the foregoing are valuable, though the filestructure extension is the most "blue sky" of the lot. Notice that almost nothing in Safety cares about file structure detail and nothing in Safety requires anything of the underlying filesystem beyond the ability to store its own files somewhere. Because the security information is vital, there needs to be thought to having redundant copies of it, and the existing facilities for fixing up inconsistencies must be extended if these proposals get implemented.

A unix implementation of this will be structured as a LKM to get control of some system calls and to provide a communications path that will not require a full driver, and a wrapper filesystem which will communicate with the LKM and at least one daemon process. The reason for this is to allow complex decisions to be made without need of doing it all in kernel. These components will allow a call to formulate a message for the daemon, send it via interaction with the LKM, place the process in a wait state, and allow it to come out of the wait when the daemon, via the LKM, terminates the wait. Note that the daemon's accesses must be untouched, and selected other processes must be exemptable from control. This is pretty much the control flow of the VMS version (less the kernel AST thread for open). It works well and makes few demands on the particular unix flavor.

The "Safety V1.5" product description gives a thumbnail sketch of a security package announced publically around 1995 (the date at which its implementation was complete).

Software  
Product

Description

Safety V1.5

Comprehensive Data Safety for your VMS systems.

from General Cybernetic Engineering

Executive Summary:

There are many perils your data faces, and loss of data can cost time, money, and jobs. Intruders, disgruntled insiders, or hidden flaws in installed software can destroy records. What is more, mistaken losses occur constantly.

Safety protects your system and your critical data in three ways:

1. A comprehensive security system adds extra checks for access to VMS files so that access by intruders or by people in non-job-required ways can be regulated or prevented. This allows your business – critical data to finally be protected against misuse, tampering, or abuse. Access from programs doing background dirty work (viruses, Trojans, worms, and the like, or even programs with security holes which can be exploited remotely (like Java browsers)) can also be blocked without damaging normal use. This active protection works three ways: by checking integrity of your files against tampering, by preventing of untrusted images from gaining privilege, and by regulating what other parts of the system an image may access.
2. A deletion protection system provides a way to undelete files which were deleted by mistake and to optionally copy deleted files to backup facilities before removal. Unlike all other VMS "undelete" programs on the market, this facility does not rely on finding the disk storage that contained the file and reclaiming it before it is overwritten. Rather, it changes the semantics of the file system delete to use a "wastebasket" system and captures the file intact. Thus, this system works reliably. No others do. This facility is also useful where you have a requirement to keep all files of a certain set of types, since the backup function can be used to capture such files while permitting otherwise normal system function. The shelving or linking functions are also available for moving copies offline if this is desired. The Safety protection features are fully integrated with the DPS subsystem, so that deletion protection does not involve destroying file security.
3. When space runs out, hasty decisions about what to keep online often must be made, and the risk of accidentally losing something important is high. Safety protects you from running out of space. Space can be monitored and older items in the

## SecurityFocus SecProg: Techniques for handling mobile code and

wastebasket deleted if it is becoming low, without manual intervention. In addition, Safety is able to "shelve" files so that they are stored anywhere else desired on your system, and they are brought back automatically when accessed. Thus no manual arrangements need be made for reloading them. Safety can also keep the files on secondary storage, keeping a "soft link" to the files at their original site so they will be accessed on the secondary storage instead. Also, Safety can store files compressed, or can store them on secondary storage so that read access is done on the secondary storage, but write access causes the file to be copied back to its original site. Standard VMS utilities are used for all file movement, and moved files are also directly accessible in their swapped sites with standard VMS utilities. The VMS file system remains completely valid at all times.

Safety gives you a full complement of tools for dealing with space issues automatically according to your site policy. These facilities are safe and easily understood. A comprehensive utility is provided by which you set your site policy to select which files are and are not eligible for automatic shelving. Also you are provided with screen oriented utilities for selecting files to shelve at any time. Access to the shelved files of course causes unshelving if the normal shelving-by-copy mode is used. Also, a simple set of rules permit locating shelved or softlink target files at any time, even without Safety running. Safety at no time invalidates your file structures for normal VMS access...not even for an instant.

In addition Safety contains functions to speed file access and inhibit disk fragmentation.

The major subsystems of Safety will now be described.

The Security Function System:

Summary:

Managing access to data critical to your business using ACL facilities in native VMS can be cumbersome and still is vulnerable to intruders or people acting in excess of their authority.

Want to be sure your critical records can't be accessed save at authorized places, times, and with the programs that are supposed to access them (instead of, say, COPY.EXE)?

Want to have protection against privileged users bypassing access controls?

Want to be able to password protect individual files?

## SecurityFocus SecProg: Techniques for handling mobile code and

Want to be able to invisibly hide selected files from unauthorized intruders?

Have you read that attacks on machines can happen because a Java browser points at a web site that damages the system (as has been reported in the press)? Want to be able to protect your systems?

The Safety security subsystem builds in facilities permitting all of these, and is not vulnerable to intruders who disable the AUDIT facility as all other commercial packages which purport to monitor access are.

Description: When your business depends on critical files, or when you are obliged by law or contract to maintain confidentiality of data on your system, in most cases the options provided by VMS for securing this data can be cumbersome and far too coarse-grained.

The problem is that certain kinds of access to data are often needed by people in a shop, but other access should be prevented and audited. Moreover, the wide system access that can come as a result of having system privileges often does not mean that it should be used to browse or disclose data stored on the system. A system manager will in general not, for example, have any valid reason to browse the customer contact file, the payroll database, or a contract negotiation file, save in a few cases where these files need to be repaired or reloaded from backups. Likewise, a payroll clerk may need read and write access to the payroll file, but not in general with the COPY utility, nor from a modem, nor in most cases at 4AM. Finally, a person who must have privileges to design a driver and test it should ordinarily not have the run of the file system as well.

Given examples like these, it is easy to see that simple authorization of user access to files is inadequate. While it is possible to build systems that grant identifiers to attempt some extra control, these can be circumvented by privilege, and create very long ACLs which become impossible to administer over a long period as users come and go.

What is needed is a mechanism that is secure, cannot be circumvented by turning on privileges, and which provides a simple to administer and fine grained control that lets you specify who can get at your critical files, with what images, when, from where, and with what privileges. It is also desirable to be able to control what privileges the images ever see, and to be able to check critical command files or images for tampering before use, so that they cannot be used as back doors to your system. It should be possible to demand extra authentication for particular files as well, and to prevent a

## SecurityFocus SecProg: Techniques for handling mobile code and

malicious user from even seeing a particularly critical file unless he can be permitted access.

The Safety security subsystem is a VMS add-in security package which provides abilities to control security problems due to intruders, to damage or loss by system "insiders" (users exceeding their authority), and to covert code (worms and viruses). It provides a much easier management interface to handle security permissions than bare VMS and provides facilities permitting control over even privileged file accesses, for cases where there are privileged users whose access should be limited. Unlike systems which only intercept the AUDIT output, EACF can and does protect against ANY file accesses, and can protect files against deletion by unauthorized people or programs in real time as well as against access.

The Safety security subsystem offers the following capabilities:

- \* Files can be password protected individually. If a file open or delete is attempted for such a file and no password has been entered, the open or delete fails.
- \* Access can be controlled by time of day. Added protections can be in place only some of the time, access can be denied some times of day, write accesses can be denied at certain times, or various other modalities of access can be allowed.
- \* You can control who may access a file, where they may be (or may not be), with what images they may or may not access the file, and with what privileges the file may be accessed. Thus, for instance, it is trivial to allow a clerk access to the payroll file with the payroll programs, but not with COPY or BACKUP, not on dialup lines, and not if they have unexpected privileges. The privilege checks can be helpful where there are consultants working on a system who should be denied access to sensitive corporate information but who need privileges to develop programs, or in similar circumstances. You specify what privileges are permitted for opening the file, and a process with excess privileges is prevented from access. Vital business data access should not always be implied by someone having privilege. With this system you can be sure your proprietary plans or data stay in house, and are available only to those with business reasons to need them, not to everyone needing system privileges for unrelated reasons. Unlike packages using the VMS Audit facility's output (which can be silently turned off by public domain code) Safety cannot be circumvented by well known means. Its controls are designed to leave evidence of what was done with them as well.
- \* You can specify that images able to run portable code (applet viewing programs or programs with powerful scripting languages)

trigger a "paranoid mode" system. When this is triggered (normally when the "loading" image is active), all file opens by the process running the triggering image are filtered by a script. This script can be different for different programs, and is site customized. The furnished sample script will broadcast the identity of user and of files being opened. It is trivial to arrange to limit this to unusual files or filesystem areas. The script can also veto the open. Thus the recommended way to treat web browsers is to limit their file access so they may read system areas and a scratch area, and may write only a scratch area. (The script is informed whether the open is for read or for write.) This "low-integrity-image" mode in which all file opens are checked with a site script which can report or veto access. This can be used to track or regulate what a Java applet can do, in case someone happens to browse a web site which exploits a Java hole to browse your system or damage it.

\* You can hide files from unauthorized access. If someone not authorized to access a file tries to open it, they can be set to open instead some other file anywhere on the system. Meanwhile, Safety generates alarms and can execute site specific commands to react to the illegal access before it can happen. This can be helpful in gathering evidence of what a saboteur is up to without exposing real sensitive files to danger. Normal access goes through transparently.

\* You can arrange that opening a file grants identifiers to the process that opens it and that closing it revokes these identifiers. Set an interpretive file to do this and set it to be openable only by the interpreter and you have a protected subsystem capability that works for 4GLs which are interpretive. (Safety identifier granting, privilege modification, and base priority alteration is protected by a cryptographic authenticator preventing forging or duplication.) This can be done whenever you want the opening of a file (including a directory)

to cause the process to assume another identity. This is governed entirely by the program's behavior, so that if your sensitive directories, for example, are tagged to grant an identifier "twit" to the process, any files whose ACLs deny or restrict access to "twit" will prevent access. These identifiers can be any legal VMS identifiers and are not restricted in form. Of course, Safety can directly control where an image may go if targets are controlled by it, but those who prefer to restrict access by ACL, or who may want to control access to network devices or the like, can use this facility. The process does not wholly "morph" into another identity, but its security relevant attributes can and do change.

## SecurityFocus SecProg: Techniques for handling mobile code and

\* You can actively prevent covert code (viruses and worms) from running in two ways. First, Safety can attach a cryptographic checksum to a file such that the file will not open if it has been tampered with. Second, Safety can attach a privilege mask to a file which will replace all privilege masks for the process that opens it. By setting such a mask to minimal privileges, you can ensure that an untrusted image will never see a very privileged environment, and thus will be unable to perform privilege-based intrusions into your system even if run from a privileged user's account. This facility is useful to ensure that processes whose security attributes are being edited due to their behavior cannot undo the changes. The reason that privileges are replaced, rather than being only reduced is that it is possible to use Safety to nominate some image with carefully controlled access characteristics to do some privileged activities. This facility will allow such functions to be done even by interpretive processes provided the system security officer has set Safety up to permit it. It is strongly recommended that this facility be used only to lower privilege levels in normal use. It is far from simple to write code that will correctly use privilege in an otherwise less trusted environment.

\* You can control base priority by image. Thus, a particularly CPU intensive image can be made to run at lower than normal base priority even if it is run interactively.

\* You can run a site-chosen script to further refine selection criteria. (Some facilities for doing additional checking while an image runs exist also.)

Safety allows you to exempt certain images (e.g., disk defragmenters) from access checks, and it is possible to put a process into a temporary override mode also (leaving a record this was done) where this is needed. Safety facilities are controllable per disk, and impose generally negligible overhead. Safety will work with any VMS file structure using the normal driver interfaces. Also, Safety marking information resides sufficiently in kernel space that it cannot be removed from lower access modes, yet it uses a limited amount of memory regardless of volume size.

Best of all, the Safety protection is provided within the file system and does not depend on the audit facility. Thus it prevents file access or loss before it happens, and does not have to react to it afterwards. Safety allows all of its security provisions to be managed together in a simple screen-oriented display in which files, or groups of files, can be tagged with the desired security profiles or edited as desired. Safety protections are in addition to normal VMS file protections, which are left completely intact. Therefore, no

## SecurityFocus SecProg: Techniques for handling mobile code and

existing security is broken or even altered. Safety simply adds additional checking which finally provides a usable machine encoding of "need to know" for the files where it matters.

The Safety Deletion Protection Subsystem.

Description: The Safety Deletion Protection System is designed to provide protection against accidental deletion of file types chosen by the site, and to allow files to be routed by the system to backup media before they are finally removed from the system. This is accomplished by an add-in to the VMS file system so that security holes are not introduced by the system's action.

The user interface is an UNDELETE command which permits one or more files to be restored to their original locations provided it is issued within the site-chosen time window after the undesired deletion took place. In addition, an EXPUNGE command is provided which allows files to be deleted at once, irretrievably, where space for such is required. Provision for automatic safe-storing of files prior to final deletion is present also in Safety DPS.

Safety DPS is implemented as a VMS file system add-in which functions by intercepting the DELETE operation and allowing the file to be deleted to be copied or renamed to a "wastebasket" holding area pending final action, and to be disposed of by a disposal agent. The supplied agent will allow a site script to save the files if this is desired, and then finally deletes any files which have been deleted more than some number N seconds ago. If the UNDELETE command is given, the file(s) undeleted are replaced in their original sites. The supplied system can also be configured to rename files to a wastebasket area or to copy them directly, for undeletion by systems people only. (These options are faster than the site command file option.)

Safety DPS can be configured to omit certain file types from deletion protection (for example, \*.LIS\* or \*.MAP\* could be omitted), to include only certain files in the protected sets, or both. This can reduce the overhead of saving files which are likely to be easily recreated, or tailor the system for such actions as saving all mail files (by selecting \*.MAI for inclusion).

In addition, Safety DPS monitors free space on disks, and when a file create or extend would cause space exhaustion, Safety DPS runs a site script. By setting this script to perform final deletions, Safety DPS can be run in a purely automatic mode in which deleted files are saved as long as possible, but never less than some minimum period (e.g., 5 or 10 minutes).

## SecurityFocus SecProg: Techniques for handling mobile code and

Safety DPS files can be stored in any location accessible to VMS. If they are renamed, they must reside on the same disk they came from. Otherwise they can be stored in any desired place.

Safety DPS is installed and configured using a screen oriented configuration utility to set it up, and basically runs unattended once installed.

### The Safety Storage Migration Subsystem

#### Description:

Safety has the ability to move files to secondary storage and automatically retrieve them when they are accessed. This backing can be similar to what HSM systems call "shelving", though it can be done in multiple levels, or it can be done in a way which permits files moved to secondary storage to be accessed there as though the files remained online. This resembles what are called "soft links" in Unix systems, in that file opens are transparently redirected to a file stored somewhere else reachable on the system, and the channel reset to the original device on close. A "readonly link" mode acts like a soft link for readonly access, and like an unshelve operation where a file is opened read/write, should this be desired. Full control over this shelving and unshelving is provided.

This provides a great deal of flexibility in reclaiming space when the Safety space monitoring function detects that space is needed. Not only can previously deleted files be finally moved to backup destinations and deleted, but the system can migrate seldom accessed files to nearline storage transparently. The site policy can drive this, or utilities provided can be used instead.

Where it is chosen to run Safety in a lights-out fashion (with Safety reacting to low disk situations by emptying older deleted files from the wastebasket and/or file migration to backing store), the policy chosen for controlling such setting is handled by a full-screen, easily used, tool which sets the policy. Should still greater flexibility be needed, the scripts used for a number of operations are supplied together with a full description of the command line interface of the underlying software. This facilitates linking Safety file management functions with other packages should such be desired.

Safety can be run in a mode where there is essentially no overhead at all imposed (just a few instructions added along some paths and no disk access) for any files except those which need softlinks or possible unshelving. There is no limit to how many files may be so marked on a disk. A fullscreen setup script allows one to select the Safety run modes. Even if Safety is

## SecurityFocus SecProg: Techniques for handling mobile code and

forced to examine all files for its markings, the overhead  
imposes no added disk access and costs only a tiny