A Multilevel File System for High Assurance

Cynthia E. Irvine
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

Abstract

The designs of applications for multilevel systems cannot merely duplicate those of the untrusted world. When applications are built on a high assurance base, they will be constrained by the underlying policy enforcement mechanism. Consideration must be given to the creation and management of multilevel data structures by untrusted subjects. Applications should be designed to rely upon the TCB's security policy enforcement services rather than build new access control services beyond the TCB perimeter.

The results of an analysis of the design of a general purpose file system developed to execute as an untrusted application on a high assurance TCB are presented. The design illustrates a number of solutions to problems resulting from a high assurance environment.

1 Introduction

As a result of the Trusted Computer System Evaluation Criteria (TCSEC) [2] system architecture requirement for minimization, trusted systems at the highest levels of assurance can present primitive interfaces lacking the rich variety of functions typically offered by general purpose operating systems. The primitive nature of high assurance trusted computing base (TCB) interfaces has lead some to argue that high assurance systems are unusable. Others assert that, as a result of its enforcement of both mandatory and discretionary access control policies, the underlying TCB places constraints on untrusted subjects that render the development of all but the most trivial of applications impossible.

This paper presents an analysis of a file system developed as part of the Gemini Application Resource and Network Support (GARNETS) [5], an operating system intended to execute as an untrusted application on a Class A1 TCB.[1]

The results presented here will demonstrate that an application as broadly stated as a general purpose file system can be designed and implemented on a high assurance base, thus providing a usable applications support environment. The design uses TCB mechanisms to provide an interface which is both "friendly" and flexible, and relies entirely upon the high assurance TCB for enforcement of access control policy.

[1] All system properties described in this paper are the result of analysis of materials which have been cleared for public release and are available to the government on a no-cost basis.

2 Design Objectives

Several design objectives motivated choices made in the development of the GARNETS system:

1. The file system has a general purpose interface usable by a variety of applications and provides sufficient functionality to permit standardized applications libraries to be ported to its interface. Applications and libraries are constrained by the underlying TCB and by GARNETS design choices reflecting the high assurance base.

2. Both mandatory and discretionary access controls are maintained and mediated exclusively by the GEMSOS Distributed TCB. The file system is designed to permit the enforcement of a hierarchical discretionary access control policy [3]. TCB mechanisms are used to propagate DAC to newly created objects.

3. The file system is multilevel but is managed by single level subjects.

4. At the GARNETS interface, all file system operations are atomic.

5. No read locks are used in the file system and application subjects with discretionary access whose level dominates that of a file system object will be able to read the object.

6. Application subjects have access only to the GARNETS file system. GARNETS has been designed so that even the application stack is built using data structures which are maintained within the file system. Support is provided so that at each access class application stacks and data segments are managed using databases located within the file system.

7. GARNETS is designed to satisfy rigorous software engineering requirements, i.e. those corresponding to Class B2 of the TCSEC:

> *... shall be internally structured into well-defined independent modules. ...*

The principle motivation for choosing a rigorous approach to software design and implementation derives from Dijkstra's [9] successful implementation of an operating system using a small engineering team.

A second motivation for achieving at least Class C2 architectural requirements results from the potential for using GARNETS as part of an Class A1 composition under the Trusted Network Interpretation (TNI) of the TCSEC [4]. Instead of implementing GARNETS as an untrusted application on a monolithically evaluated TCB, one might elect to use it as part of a Class A1 TNI composition. In this case mandatory access control policy would be enforced by the underlying Gemini Trusted Network Processor (GTNP) [20], a recently evaluated Class A1 TNI "M-component". Here the GEMSOS Discretionary TCB and GARNETS would be executed on a virtual machine created by the underlying GTNP and would be untrusted with respect to mandatory access control policy. If a database management system enforcing a highly granular access control policy were to be ported onto the GARNETS interface, then GARNETS would become part of a TNI "D component" [11].

Figure 1 illustrates the general architecture of the GEMSOS/GARNETS system. The GEMSOS mandatory TCB provides a ring mechanism which is used to create the protection domains available to each process. The classic definition of *process-domain pair* is applicable to each GEMSOS subject. Thus there is a TCB subject enforcing discretionary policy, a GARNETS subject, and an applications subject.

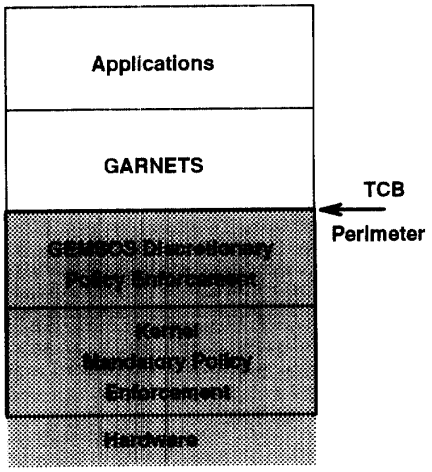

Figure 1. Standard GARNETS Architecture

Figure 2 depicts a systems architecture for a data base management system based on a TNI composition. In this case, the TCB perimeter is at the DBMS interface. The same subject applies to both GARNETS and the DBMS.

3 File System Building Blocks

The GARNETS documentation revealed that the GARNETS file system is constructed using named objects exported at the interface of the GEMSOS distributed TCB. There are two types of named objects: discretionary access control lists (DACLs) and multisegments. Access control lists (ACLs) are associated with every object of both types and are used by the TCB to mediate discretionary access to the objects. Objects at the TCB interface may be composed of one or more storage objects. Associated with each storage object is a label which is used by the TCB to enforce a non-discretionary access control policy.

3.1 Segments

The fundamental storage objects and the loci of mandatory access control are segments. GEMSOS segments are similar to those described in the Multics system [19]. Associated with each segment is an immutable access class which reflects the sensitivity of the information stored in the segment [18]. A process, as the surrogate for the user, provides the vehicle by which the user can reference and access segments. Segments may be referenced once they are added to the virtual memory of the process, in GEMSOS, an action called "making known" a segment. When part of the virtual memory of the process, segments may be read or written, according to the mode of access obtained, as a piece of memory. Finally, segments are simultaneously and independently shared by processes, where the actions of each process on a given segment are controlled by the access rights with which the segment is made known to the process. At the GEMSOS TCB interface, segments are accessible only as elements of multisegments, which will be discussed below.

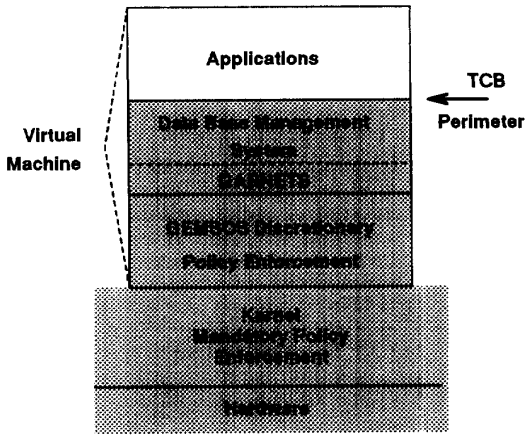

Figure 2. GARNETS TNI Architecture

3.2 Discretionary Access Control Lists

Discretionary access control lists (DACLs) are interpretively accessed named objects exported at the

GEMSOS TCB interface used for mediating discretionary access. Each DACL contains a limited number of access control lists corresponding to the number of entries that can be created off of the DACL segment in the GEMSOS segment naming system [18]. The ACLs contained in DACLs are used to mediate discretionary access to DACLs and msegs which are associated with that particular DACL by virtue of being "entries" off of the DACL. Each DACL contains two modifiable templates which are used by the TCB to initialize the access control lists for DACLs and multisegments respectively, thus ensuring a default access control list for each object. DACLs are used by GARNETS as be building blocks for directories.

3.3 Multisegments

Multisegments, or msegs, are named TCB objects. Each multisegment consists of a collection of zero or more segments. Each segment is a TCB storage object having a label attribute so that access to individual segments is mediated by the underlying mandatory TCB. Within a given multisegment, all segments are hierarchically related to a single base segment with which an ACL is associated. Once current access to the base segment of the mseg is obtained, discretionary access to other members of that mseg's hierarchy is granted. An mseg may be multilevel and access to each segment within an mseg is mediated by the TCB for binding to the base segment and for the non-TCB subject's access with respect to the mandatory access control policy. The rules for compatibility and inverse compatibility [7, 8, 18] govern the mandatory relationship between a segment and its entries. The size of each segment within an mseg is bounded only by an upper limit to segment size imposed by the TCB. Msegs are used to contain data. GARNETS' internal data structures, such as directory databases, and interpretively accessed files are built from msegs. Msegs are also exported at the GARNETS interface.

4 File System Objects

In this section, analysis of the rationale for the choice of objects visible at the GARNETS interface is reviewed. A guiding principle evident throughout the GARNETS design documentation was the reliance upon strong TCB mechanisms to provide the enforcement of access control policy rather than building mechanisms within the operating system which would have provided a lower level of or no assurance that the DAC policy was enforced correctly [10]. Thus despite the fact that GARNETS is not part of the high assurance TCB, it exhibits "strong DAC". This means that, although GARNETS might be flawed or contain malicious software, user data is protected by TCB mechanisms. The types of objects supported by GARNETS were based upon the objectives of relying on the high assurance TCB to provide access control policy enforcement and constructing a file system which could be managed without resorting to trusted subjects.

The GEMSOS TCB provides a ring mechanism [18] which GARNETS uses to protect its internal data structures and provide interpretive access to objects by less privileged applications.

4.1 Directories

Several access control objectives provided motivation for the design of GARNETS' directories.

GARNETS' designers chose to permit DAC access to files independent of discretionary access to directories along the paths to files. (This choice results in a divergence from UNIX [6] which requires access to all directories along a path in order to access a file. The result of this design choice in an environment based on the high assurance GEMSOS DAC interface was a rather complex directory structure.

In addition, GARNETS' designers sought to create a file system in which the default access to files and named msegs (see section 4.2) could be managed independently of the default access to directories.

4.1.1 Internal Directory Structure

The file system is built from three parallel trees each with a similar structure. The first is a directory tree in which access to a particular DACL in that tree determines whether or not the subject can manipulate directories, e.g. create and delete entries. Associated with each directory component in the directory tree is an mseg used to contain GARNETS-internal directory management databases.

Files occupy a separate, but parallel, tree consisting of two components per directory: a DACL from which the tree is extended and a DACL containing the access control lists applied to the files and named msegs which are its entries.

Finally a third tree is constructed from segment entries within an mseg the internal structure of which mirrors the directory tree. This huge mseg contains a dynamic "road map" to the file system and is used to walk the directory and file trees in search of target objects. It contains, for example, the names and aliases for directory entries. Because access to this giant mseg must be set at the root of the tree, its discretionary access control list must be such that it is accessible to all GARNETS subjects. Upon first inspection one might assume that this mseg, which is writable by all GARNETS subjects (constrained by the mandatory labels associated with each component segment), renders the entire file system vulnerable to attack. There are three reasons why this is not so:

1. Data containers, viz. files and named msegs, are protected by ACLs associated with the file tree. So even if the information in the global mseg were to become corrupted, the data to be protected would still be subject to access mediation by the TCB, however difficult it might be to locate.

2. The mseg is accessible only by GARNETS subjects, thus a GARNETS Trojan Horse rather than a malicious application would be needed to intentionally disrupt this portion of the directory data.

3. For each function requiring explicit access to a directory, GARNETS requires a check of discretionary access to the directory components in canonical order. Thus GARNETS relies on DAC

checks within the directory to protect the mseg from unauthorized modification.

Figure 3 illustrates the components that are used to build GARNETS directories

GM is the multisegment in which provides all subjects with a road map to the directory tree

DTD is the Directory Tree DACL. the GEMSOS DACL used to control access to the tree from which directories are built and contains the access control lists associated with directory entries to the directory

DM is the Directory Multisegment, the GEMSOS multisegment used to contain dynamic data associated with directory entries

FTD is the File Tree DACL, the GEMSOS DACL used to extend the tree

FD the File DACL, the GEMSOS DACL which contains the access control lists associated with file entries to the directory

These three trees permit the DAC, including the default access control lists, for files and directories to be managed independently. In addition, access to files does not require explicit access to intervening directories.

4.1.2 Single Level Directories

GARNETS supports directories which contain information all at one access class.

4.1.3 Multilevel Directories *(rejected)*

From the GARNETS perspective, multilevel directories were considered to be directories which contained information at different access classes. GARNETS designers chose not to support multilevel directories. The need for trusted subjects to manage multilevel directories was the principle reason for their rejection. In addition, GARNETS' virtual multilevel directories (see section 7.3) provide an agreeable alternative to physical multilevel directories.

4.1.4 Upgraded Directories

GARNETS permits the creation of upgraded directories. The GEMSOS TCB requires that the compatibility property be preserved, thus the access class of an upgraded directory must dominate that of its parent [18]. GARNETS limits directory information contained in the parent directory to that which should be visible at parent directory's access class. For example, the names and creation dates of upgraded subdirectories are visible to parent-level subjects. All dynamic directory information is contained in the upgraded directory itself. Thus, attributes such as the time of last modification and contents of of the upgraded directory are visible only at the upgraded access class. In order to provide uniform directory semantics and implementation, both normal and upgraded directories are created and initialized using the same functions. In the case of upgraded directories, the initialization is exported to the GARNETS interface since it must be performed by a subject at the upgraded access class rather than one at the level of the parent directory.

Deletion of upgraded directories will require the use of a trusted subject, which in this context does not necessarily imply the use of a subject with the entire range of access classes possible on the system, but one whose range encompasses the access class of the upgraded directory and its parent. Because of this need for a trusted subject, GARNETS controls the creation of upgraded directories. and requires users to have a special authorization to create upgraded directories. The fact that special measures are required to delete upgraded directories might indicate that such objects should be prohibited. Subsequent discussion will illustrate how they can be avoided, however possible requirements for a multilevel file system on a multilevel volume appears to have lead to their inclusion in the file system.

4.2 Named Multisegments

At its interface, GARNETS presents multisegments that are nameable directory entries. In contrast to files, which are interpretatively accessed GARNETS objects, once a multisegment is included in the process' address space, segments within a named mseg may be accessed directly via the available hardware primitives. The GEMSOS segment aliasing virtualizes segment names in order to prevent covert channels that would result from a "flat" system-wide segment naming scheme [18]. Within each process, subjects in more privileged domains must protect the segments used for subject-internal databases from corruption by less privileged subjects. GARNETS accomplishes this by virtualizing its per-process segment naming scheme and utilizing the GEMSOS ring mechanism to insure the integrity of its own segments.

There are three major reasons to justify the exportation of named msegs at the GARNETS interface. First named msegs permit processes to **avoid unnecessary buffering**. For example, if executable code is stored in files, then internal to the file the executable will be broken up into one or more segments each with a length equal to the standard file block size. When code is to be executed, it must be read from the file and placed into one or more executable segments the size of which corresponds to that required by the code itself. By using named msegs to contain executables, the code can be stored in segments within the file system which are the correct size and directly executable. Since code should not be modifiable, these segments can be shared by multiple processes (only data segments need to be created on a per-process basis). Thus named msegs **promote sharing** of executables the benefits of which include efficient use of real memory resources and potential increased performance resulting from reduced swapping. Because named msegs are directly accessible

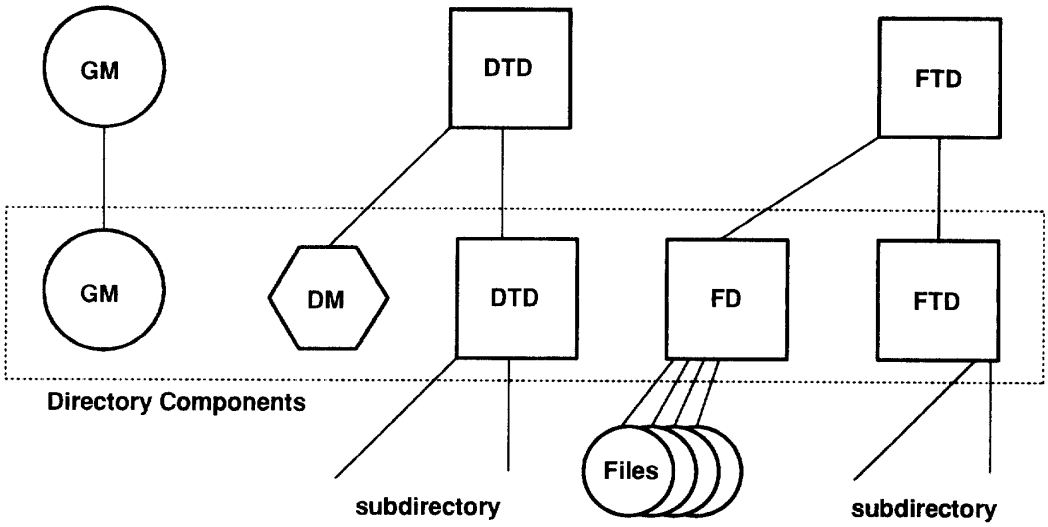

Figure 3. GARNETS Directory Structure

through available hardware primitives, their use for application-level databases **reduces context switching**.

GARNETS multisegments may be used by applications for the following purposes:

- Storage managers and databases.
- Executables for either general purpose libraries or specific applications.
- Interprocess communication objects.
- Synchronization. GARNETS exports the TCB's synchronization functions, which are based on eventcounts and sequencers [17, 18]. Segments are used to name these abstract data objects with named msegs providing the connection between file system objects and these more primitive TCB abstractions. This provides a highly efficient mechanism for synchronization and concurrency control.

4.2.1 Extended Code Sharing

The availability of msegs as containers for code permits the use of the TCB ring mechanism to create code segments which may be accessed by subjects in multiple domains within the same process. This was utilized to advantage by GARNETS in a storage manager which is required by GARNETS itself and is available for use by low level libraries in the application domain as well. The two domains maintained their own initialization code and data segments for the storage manager, but shared the bulk of the executable code in common. Thus GARNETS provides executables which, if labeled at the system low access class, are executable by all GARNETS and application subjects: code sharing is both inter- and intra-process. Since the shared code made TCB function calls, applications executing the code are spared the additional context switches through GARNETS. This could provide a performance advantage.

4.2.2 Single Level Named Multisegments

Subjects which have modify or append access to a directory are able to create single level named multisegments. The ACL associated with a particular named multisegment determines the mode of discretionary access permitted for a given subject.

4.2.3 Multilevel Named Multisegments

Multilevel named multisegments are exported at the GARNETS interface and are available for applications designers to create multilevel data structures. Obviously, electing to utilize these objects requires care on the part of applications designers since the untrusted subjects of the GARNETS environment cannot delete upgraded objects. To prevent users from wantonly creating objects which require administrative intervention, viz., trusted subjects, to delete, GARNETS requires users to have a special authorization to create multilevel named msegs.

4.3 Files

In this section the analysis reviews the choices regarding files made by the GARNETS designers. The focus here is on decisions related to the high assurance multilevel perspective.

4.3.1 Single Level Files

Files are interpretively access objects provided for use by applications at the GARNETS interface. One ACL managed by the TCB is associated with each file. File attributes maintained by GARNETS include file size, time of last modification, and a write version (see section 9.2). Time of last access, viz. *read*, to files is updated only when the access is made by subjects at the level of the file.

4.3.2 Multilevel Files *(rejected)*

GARNETS designers chose not to support multilevel files. Apparently, experience had already indicated that sufficient care in the design of applications could eliminate perceived requirements for multilevel files [14, 12]. Where multilevel objects are required, it is possible for untrusted applications to create views that give the user the illusion of multilevel objects. The semantics of multilevel files would have created an incoherent interface. A small sample of the problems associated with multilevel files clarifies the reasons for their rejection: Where would file attributes be stored, at the lowest access class of the file or the highest? How could standardized support libraries at the application level, e.g. the ANSI C language libraries, be used in the context of multilevel files? How would complex trusted subjects be avoided?

As a result of electing to not implement multilevel files, all attributes for all files are stored within a directory in a single directory-local object which has the same mandatory access class as the files and their parent directory. Obviously this object is not protected by individual file ACLs, but the ACL associated with the directory. Thus the directory is the unit of access control for attributes. For most purposes, this would be adequate, however, if high granularity on DAC for attributes is needed, then additional directories will be required.

5 Solving the Gizillion Problem

High assurance multilevel trusted systems must handle the problem of a potentially very large number of access classes. (Systems are in use which have 1.5×10^6 access classes [1].) Known as the *gizillion problem* [11], it had to be addressed when building flexible untrusted applications on GEMSOS. The underlying TCB provides access classes consisting of two sets of 16 hierarchical levels and 96 non-hierarchical categories. Because TCB complexity must be minimized at high assurance, an objective of any design technique to accommodate these access classes will be be the avoidance of the construction of elaborate data structures by the TCB in support of applications.

At each access class, the GEMSOS system provides only one object which can be used as an access class base by non-TCB applications. As a consequence, an untrusted application such as a file system cannot depend upon trusted mechanisms to build its multilevel data structures; whatever file system constructs are required at a particular access class must be constructed by the untrusted operating system.

When a previously unencountered access class is selected, a GARNETS subject must be able to create the data structures and data bases required to support an application subject. At a minimum, a stack for the application subject executing on GARNETS will be needed.

Several solutions were possible. Users could request that the GARNETS administrator create the data structures at the new access class. At system low, the administrator would create an upgraded directory below the file system root for each new access class. Then at each new access class the administrator would have to create the necessary file system data structures. A disadvantage of this approach is that the GARNETS administrator would have access to the system at the full range of access classes. In addition, use of the system at a new access class would depend upon the administrator's timely response to requests.

An alternative would be for the administrator to create data structures at all possible access classes *a priori*. The advantage to this approach is that the data structures would be available whenever a user wished to use a "new" access class. We note that this choice is untenable because of the time required to create a gizillion data structures. On the other hand, if a trusted subject were employed to automate this process, the GARNETS designers would have failed in their objective to have the system managed by untrusted subjects. In addition, a trusted subject able to create portions of the file system would be far too complex to satisfy high assurance system architecture requirements, which must be met by the GEMSOS TCB. In either case, the creation of a gizillion data structures would certainly consume all of the system's available disk space.

GARNETS utilized another strategy: to dynamically create all required data structures at each newly encountered access class. On the first occurrence of a previously unencountered access class, i.e., the instantiation of a GARNETS subject at that access class, the GEMSOS TCB builds a DACL segment that may be used by non-TCB subjects as a base for creating data structures. A function is provided at the TCB interface to locate the DACL. With this base, GARNETS dynamically builds the segment substructure needed for its own execution and portions of the file system sufficient to support initial execution of non-GARNETS applications. Once all of the essential data structures have been created for a particular access class, GARNETS is able to support additional non-TCB subjects at that level.

A portion of the file system is created at each new access class. The per access class file system base is called a per access class (PAC) directory. With the PAC directory, application subjects at the new access class have a location for "home" and "temporary" di-

rectories.

It should be noted that when GARNETS is first installed, a GARNETS administrator must login at the system low access class and execute bootstrapping code which has been previously installed using a tool provided with the Gemini system. Thus GARNETS code is not located in the file system, but in separate data structures at pre-defined locations. Later, when other subjects at higher access classes are instantiated, these GARNETS subjects will dominate the access class of the code and will be be able to create the file system structures using the per access class base as a starting point.

6 File System Object Naming

6.1 Aliases

Within a directory, GARNETS permits the use of alias names for objects. These aliases are tied to the physical object so that the object is not deleted until its last alias is deleted.

6.2 Links

GARNETS designers chose not to use hard links, i.e. links such that the object is shared between its names. A compelling reason for this choice was the fact that the mandatory TCB would prohibit the creation of such links across access classes.

Instead, GARNETS implements symbolic links each of which is a path to a target object. Symbolic link paths may contain links. When access to an object is made via a GARNETS function, the existence of intervening links is transparent to the user. In addition to links to files and named msegs, GARNETS permits symbolic links to be created to directories and to links themselves. As is the case in many file systems using symbolic links, GARNETS does not explicitly check for cycles when starting to traverse a path. Instead, the number of links traversed is counted and when an upper limit is encountered a cycle is assumed and the traversal is aborted.

6.3 Per Access Class Links

GARNETS supports a form of symbolic link which includes a field for an access class these are known as per access class (PAC) links. When resolving a PAC link, GARNETS finds the PAC directory corresponding to the access class in the link. The remainder of the path associated with the PAC link is traversed relative to the PAC directory.

7 Leveraging File System Solutions

Having provided a solution to the gizillion problem, a number of benefits result.

7.1 Use of Single Level Volumes

When the TCB is configured with single level volumes, the GARNETS file system can be distributed by access class by building single level file systems on each volume. Symbolic links permit the file system to be bound into a multilevel data structure, with the underlying volume configuration transparent to applications.

7.2 Creation of Per Access Class Resource Services

Within an operating system certain services are provided to all operating system subjects. In a distributed operating system such as GARNETS providing these services is based on the ability of instances of the system to coordinate their management of those services. The creation of services on a per access class basis permits coordinated services without trusted subjects.

Unique file system identifiers comprise an example of an internal resource which is needed for coherent management of file systems. In general, unique file system identifiers are numbers chosen from a large set, are never reused, and are always bound to a specific object. Unique file system identifiers for multilevel systems present special problems. Constraints of the mandatory access control enforcement mechanism prevent one untrusted source of file system identifiers for all GARNETS subjects. Having guaranteed that there will be resources available at each new access class that can be used by GARNETS to build its internal data bases, GARNETS can dynamically construct file system identifier services at each new access class. To insure that file system identifiers are unique across access classes, the access class is implicitly part of the identifier, viz. if two names refer to objects with the same numerical identifier but different access classes, then the names refer to different objects.

7.3 A Virtual Multilevel File System

When combined with symbolic links, PAC links offer a number of advantages for creating a multilevel file system.

- single level subjects can create and destroy symbolic links, no interaction with subjects or objects at other access classes is required
- links can be created which cross volume boundaries
- symbolic links may be created to objects at either higher or lower access classes
- because GARNETS permits the creation of links to directories, it is possible to create the illusion of upgraded directories, files and named msegs without creating objects which are physically upgraded relative to the directory in which they appear to be located. This permits "upgraded" file system objects to exist in GARNETS without requiring a trusted subject for the deletion of such objects.
- symbolic links need not be bound to preexisting objects. This permits the user to create a "view" of the file system at a particular access class without having to toggle between several other access classes to insure that the objects exist.
- GARNETS users have the ability to create and manage *virtual multilevel directories* without the aid of trusted subjects. These directories are

likely to provide users with substantial file system management benefits, saving users considerable time when interactively navigating through the file system.

7.4 Working Directories

Many file systems support the notion of a current working directory, i.e. a directory relative to which other objects in the file system are named. Because the components of the file system with which an application might be working could be distributed over several different volumes, GARNETS supports multiple working directories. This may be employed by applications to reduce path walking and, if desired, can always be reduced to the degenerate case of only one current working directory by application libraries.

8 GARNETS Self Protection

The GEMSOS TCB creates protection domains and provides a ring mechanism [18]. Although the GEMSOS mechanisms are available to permit an application to protect itself from external tampering or modification, it is necessary for the application to use them effectively. GARNETS succeeds in this respect. The GARNETS system is parameterized to so that its subjects execute in a specific protection domain. Application subjects execute in a less privileged domain. The interpretatively accessed objects such as files and directories that GARNETS presents at its interface, as well as GARNETS' internal data structures are stored in TCB objects which are accessible only by subjects at least as privileged as those in the GARNETS ring.

8.1 GARNETS Ring Brackets

Certain directories within the GARNETS file system are required by GARNETS for its own correct operation. The DACLs used to construct directories are TCB objects which are interpretively accessed by GARNETS, thus the GEMSOS ring mechanism has already be applied against these objects and cannot be used by GARNETS in this instance. To permit GARNETS to distinguish between directories which GARNETS may access on behalf of applications and those that GARNETS reserves for its own use, GARNETS created its own simple ring mechanism. The operating system supports its own ring brackets and provides for caller validation. The GARNETS ring brackets define a range of callers on whose behalf GARNETS will grant selected modes of access.

GARNETS ring brackets apply to all objects within a particular directory and are permanently set at the time of directory creation from a modifiable template in the parent directory.

9 Consistency and Concurrency

9.1 File Consistency

A desirable feature of file systems is file consistency in the face of discontinuities. The GARNETS system contains a mechanism to permit fine-grained robustness selection.

Robustness level is intended to provide users with a flexible mechanism to assign consistency requirements to files. The GEMSOS TCB does not give any guarantees that a segment when added to the address space of a process, swapped into volatile memory, and subsequently modified will be written to secondary storage: instead an explicit function call must be made to the TCB to flush the segment to secondary storage. Guarantees regarding the consistency of data in volatile memory with respect to its version on secondary storage must be provided by non-TCB mechanisms. Following a system discontinuity, users need to know the state of their files. For a file system, a redo log scheme such as those found in databases would be too complex. Instead, GARNETS provides an indicator of file consistency and four robustness levels for maintaining consistency between copies in memory and those on secondary storage. Following a system discontinuity, it is up to the GARNETS administrator or user to examine the consistency indicators and repair files as needed.

In specifying robustness levels, two factors are considered: whether the file block is written to secondary storage and whether the indicator of file consistency is written to secondary storage. Four file robustness levels are available:

none This robustness level provides no file consistency support. It is most appropriate for temporary files.

on_close When a file is opened it is marked potentially inconsistent and the indicator is written to secondary storage. No guarantees regarding file robustness are given while the file is open. When the file is closed, all modified file blocks are guaranteed to have been written to secondary storage and the file is marked as consistent and the indicator is written to secondary storage.

check_pointing This robustness level is intended for use by applications such as databases. Modified file blocks are written to secondary storage as part of each call to the GARNETS *write_file* function. To reduce the overhead associated with updates of the associated consistency indicator, the indicator is updated only when the file is opened and when it is later closed.

on_write Each write is provided with the assurance that modified file blocks are written to secondary storage and the file is marked as consistent following a successful return to the application

For files which may have multiple concurrent writers, robustness is a file attribute assigned on a per-directory basis and may not be modified following directory creation. This design choice reflects the practical observation that users tend to treat all of the files in a particular directory similarly.

An option provides for directories in which access to all contained files is restricted to a single writer at a time. For such files it is possible to allow the writer to set the file's robustness level at the time it is opened. Full file write locks are used to insure single writer access.

9.2 File System Concurrency Control

Atomicity of operations and concurrency control in GARNETS is based on the use of a TCB primitive for atomic updates that utilizes a RAM-based eventcount pair. Using this in its design, GARNETS avoids the introduction of mutual exclusion mechanisms that might conflict with the real-time capabilities of the GEMSOS TCB.

GARNETS does not provide to applications a total ordering on file system operations such as one might achieve using concurrency controls based on Lamport's logical clocks [13]. Instead, an optimistic approach was adopted. Version numbers are associated with interpretively accessed file system objects, viz. files and directories. When objects are read, a version number is returned. Subsequent modification operations require a version number to be submitted as a parameter. In this "Scarlet O'Hara" [15] approach to concurrency control, a process does as it pleases until returned an exception informing it that the version number used in a modify call was unacceptable.

Interesting situations resulting from the fact that directories consist of multiple objects confronted the GARNETS designers. Of particular note was the potential for inconsistent DAC across the DACLs within a directory. This might occur when two processes with DAC access to a directory call the GARNETS function to modify access to the directory. Within the GARNETS domain, their TCB calls to modify the ACLs could interleave with the result that the ACLs might not be consistent. GARNETS deals with this problem by assuming that the likelihood of this situation occurring was remote. Strict two phase access to directory components is required, i.e. all DACLs must be accessed before any modification is begun. Also, all modification to ACLs is conducted in canonical order using a versioning technique. Without using locks, the GARNETS design significantly reduces the possibility of ACL inconsistency in multi-DACL objects.

10 Summary

An analysis of the design of a file system intended for use as an application on a high assurance TCB has been presented. The file system represents the successful implementation of a complex general purpose application in a high assurance context. Major design objectives have been met and the interface is sufficiently flexible to be useful by a broad spectrum of applications.

Through the combined use of symbolic links and a solution to the *gizillion problem*, GARNETS designers have provided an implementation with which applications can create a virtual multilevel file system that can be managed by untrusted subjects and can be employed in a single-level volume configuration. Because GARNETS does permit the creation of upgraded directories and multilevel named msegs, the potential need for trusted subjects still exists. The special user authorizations required to create true multilevel objects is an important feature for controlling the unnecessary proliferation of such objects.

Access to files and named msegs is not tied to access to directories. If the path to a file is known, then that path can be followed and only the ACL on the target object will be used to mediate access to it. This feature is of particular importance due to GARNETS' heavy reliance on symbolic links.

As any file system should, GARNETS protects itself and its interpretatively accessed objects. By using TCB-supplied access control lists (DACLs) for the scaffolding of the file system and thus depending upon the high assurance GEMSOS TCB for mediation of all security policy-related accesses to file system objects, the GARNETS designers have built a file system with highly effective protection mechanisms.

Acknowledgements

The author is grateful to the Naval Postgraduate School for support to conduct this analysis; would like to thank Roger Schell, Mike Thompson, and Tim Levin for their encouragement; and expresses appreciation to an anonymous referee for suggestions.

References

[1] from comments by an anonymous referee.

[2] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD. National Computer Security Center, December 1985.

[3] *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003. National Computer Security Center, September 1987.

[4] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, NCSC-TG-005. National Computer Security Center, July 1987.

[5] Software development specification for Gemini application resource and network support (GARNETS). Technical Report GAR00-SPB00-0101, Gemini Computers, Inc., Carmel, CA, 1994.

[6] Maurice J. Bach. *The Design of the UNIX Operating System*. Prentice Hall, Inc., Englewood Cliffs, NJ, 1986.

[7] David Bell and LaPadula Leonard. Secure computer system: Mathematical foundations and model. Technical Report M74-244, MITRE Corp., Bedford, MA, 1975.

[8] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, MITRE Corp., 1977.

[9] Edsger W. Dijkstra. The structure of the "THE"-multiprogramming system. *Comm. A.C.M.*, 11(5):341–346, 1968.

[10] C.E. Irvine, T.B. Acheson, and M.F. Thompson. Building trust into a multilevel file system. In *Proceedings of the 13th National Computer Security Conference*, Washington, DC, 1990.

[11] C.E. Irvine, R.R. Schell, and M.F. Thompson. Using TNI concepts for the near term use of high assurance database management systems. In *Proceedings of the Fourth RADC Database Security Workshop*, Little Compton, RI, April 1991.

[12] C.E. Irvine, M.F. Thompson, R.R. Schell, T.F. Tao, J.R. Lessin, R.E.and Wopat, and E. Ben-Arieh. Genesis of a secure application: A multilevel secure message preparation workstation demonstration. In *Proceedings of the 4th Aerospace Security Applications Conference*, pages 30–36, Orlando, FL, December 1988.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the A.C.M*, 21(7):558–564, 1978.

[14] T. F. Lunt, R.R. Schell, W.R. Shockley, M Heckman, and D.F. Warren. A near-term design for the seaview multilevel database system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 234–244, Oakland, 1988.

[15] Margaret Mitchell. *Gone With The Wind*. 1936.

[16] E. I. Organick. *The Multics System: An Examination of its Structure*. MIT Press, Cambridge, MA, 1972.

[17] D.P. Reed and R.K. Kanodia. Synchronization with eventcounts and sequencers. *Communications of the A.C.M.*, 22(2):115–123, 1979.

[18] Roger Schell, T. F. Tao, and Mark Heckman. Designing the GEMSOS security kernel for security and performance. In *Proceedings of the 8th DoD/NBS Computer Security Conference*, pages 108–119, 1985.

[19] Michael D. Schroeder and Jerome H. Saltzer. A hardware architecture for implementing protection rings. *Comm. A.C.M.*, 15(3):157–170, 1972.

[20] M.F. Thompson, R.R. Schell, A. Tao, and T. Levin. Introduction to the Gemini Trusted Network Processor. In *Proceedings of the 13th National Computer Security Conference*, pages 211–217, Baltimore, 1987.