CMU-ITC-83-007

Report of the Subgroup on Protection in VICE

1 September 1983
20:20

M. Satyanarayanan

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Table of Contents

List of Figures

Preface

This document describes the design of a mechanism for controlling access to objects in VICE. This design is based on discussions held by a subgroup of the ITC Common System Services Group, consisting of Dave Gifford, M. Satyanarayanan, and Al Spector. The task of the subgroup was to examine the problem of file protection in VICE, and to come up with a strawman design addressing this issue.

As presented here, the problem is cast in a more general light: that of protecting arbitrary objects in VICE. It is our view that VICE will be composed of many relatively independent subsystems, each providing different services. What we propose here is a general mechanism for these subsystems to control access to the objects they are responsible for. The file subsystem is a very important instance of such a subsystem, and the discussion on it assumes the design described in the document "Report of the ITC File System Subgroup."

The principles guiding this design are that it should:

- be richer and more flexible than the Unix file protection scheme.
- be easy to administer.
- allow revocation of privileges after they have been granted.

Note that we do not consider the problem of low-level authentication here. In other words, it is assumed that there is some orthogonal mechanism for determining who the requester is, and for ensuring that communications with the requester are not public. The *login* primitives mentioned at various points in the document establish authentication with each individual subysystem. However the details of such authentication, as well as the maintenance of an authenticated session, are not addressed.

1 Introduction

Associated with each protected object in VICE is an *Access List*, which is a function from a *Protection Domain* to a set of rights. For every member, *m*, of the protection domain, such an access list answers the question "What rights does *m* possess on this object?" The protection domain consists of two kinds of entities: *Users* and *Groups*.

A user represents an accountable entity in the system: one who can be charged for resources and who can be held responsible for actions performed on his behalf. Typically, a user is a human user of the VICE/VIRTUE systems.

A group is a set of users and/or a set of other groups. The implication of this recursive definition is illustrated by Figure 1. The user *U* is a direct member of group *A*, and *A* is a direct member of groups *C* and *D*. The *"is a member of"* relation is transitive, and hence *U* is an (indirect) member of groups *C* and *D*. The rights that *U* possesses on an object is given by the union of the rights that are specified for *U*, *A*, *C*, and *D* on the access list for the object.

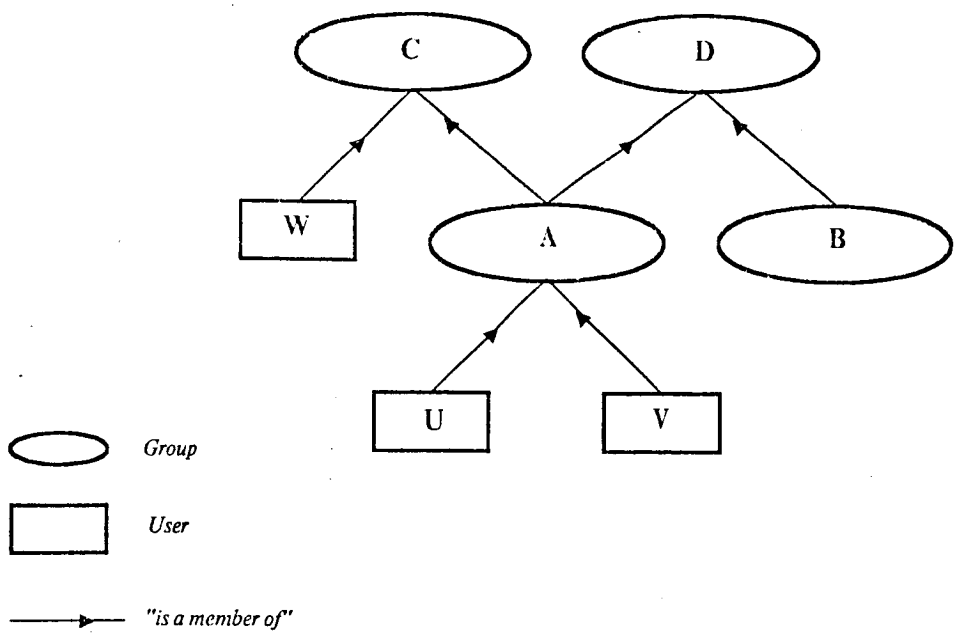

Figure 1: Users and Groups

More generally, the transitive closure of the "is a member of" relation yields the set of all groups that a user is a direct or indirect member of. This set, together with the user himself, is referred to as the *Current Protection Subdomain* of the user. The rights that a user has on an object is the union of the rights specified for his current protection subdomain on the object.

Conversely, the transitive closure of the *"has as a member"* relation yields the set of all members (direct and indirect) of a group. In Figure 1, for example, the membership (direct and indirect) of *C* is *A*, *U*, *V*, and *W*.

Before protected objects may be used, the requester must *Login* to the VICE subsystem that deals with those objects. Logging in establishes two bindings for the duration of the login session: the identity of the requester, and his current protection subdomain. Only users may log in to VICE — groups may not. In most cases, processes in VIRTUE will perform the necessary logins to VICE subsystems, thus avoiding human intervention.

While the protection domain is universal in VICE, the interpretation of rights is object-specific. Each VICE subsystem that implements a type of protected object has to provide the following:

- A correspondence between operations on objects and rights in access lists.
- An access list for each instance of an object, and primitives to manipulate it.
- Enforcement of the protection specified in the access lists.

At the present time, we are concerned with only two VICE subsystems: the subsystem responsible for managing the protection domain, and the file system. Sections 2 and 3 discuss protection in the context of each of these subsystems. As other object types are implemented in VICE, the requirements mentioned above will have to be addressed for each of them.

2 Protection Domain Management

The protection domain management subsystem handles the creation and deletion of users and groups, and the modification of group membership. The protected objects manipulated by this subsystem are users and groups.

Each group has one user who owns it, possesses all rights on it and who is responsible for its administration.[1] Initially, the user who requests the creation of a group is its owner. Ownership can be transferred to any other user.[2] An owner can delegate responsibility for administering a group by allowing group modification privileges to other members of the protection domain.

---

[1] There is a school of thought that believes that joint ownership should be possible. For example, the set of users and groups who have the ability to modify a group could be its owners. Single ownership does, however, simplify issues regarding accountability.

[2] Does the new owner have to acknowledge his willingness to bear this responsibility? This question is one of a set of related questions. Do you have to give permission to be put on an access list? To be included as part of a group? The principle adopted here is that such acknowledgement is unnecessary. If it were necessary, VICE would have to provide some kind of handshaking mechanism, whereby the donor and the recipient of a privilege could both confirm the acceptance of their roles.

A distinguished user called "System" corresponds to the system administrator and posseses certain unique privileges. While all users can create groups, only System can create other users. System is the only user that VICE starts out with.

Initially, VICE has a single, distinguished group called "World", whose sole member is System. Every other group that is created in the system has, for naming purposes, a parent group. The name of a group is thus similar to a Unix file name: if World is the parent of A, and A the parent of B, then a group C created with B as its parent will have the name World.A.B.C. By convention, the "World." prefix can be omitted, and hence the above group can be referenced as A.B.C. Note that parenthood is completely independent of membership. It is quite possible, though highly unlikely, that there are no common members of A, A.B, and A.B.C in the above example. Usually, a group will be a direct member of its parent group. A user may create a group without a parent group — in that case the name of the user is used as a prefix of the group name. Figure 2 illustrates group naming for a small organization. The purpose of this naming scheme, is to prevent the uncontrolled generation of groups which are of limited general interest, but whose names have widespread mnemonic significance.

2.1 Protection Subsystem Rights

Associated with each group is an access list, specifying who may examine the group or modify it. As mentioned before, only System can create and delete users. The rights associated with a user are:

ListMemberShip Allows you to list the groups that this user is a direct member of.

Groups, however, may be manipulated by users if they possess adequate rights. The rights associated with a group are:

ListMembers Allows you to find out who the direct members of the group are.

ListMembership This allows you to find out which groups this group is a direct member of.

ModifyMembers This allows you to add members to or delete members from a group. Ownership automatically bestows this right.

2.2 Protection Subsystem Calls

The following calls are supported:

Login (*user, Authentication Information*)
Establish one's identity.

Logout() Obvious

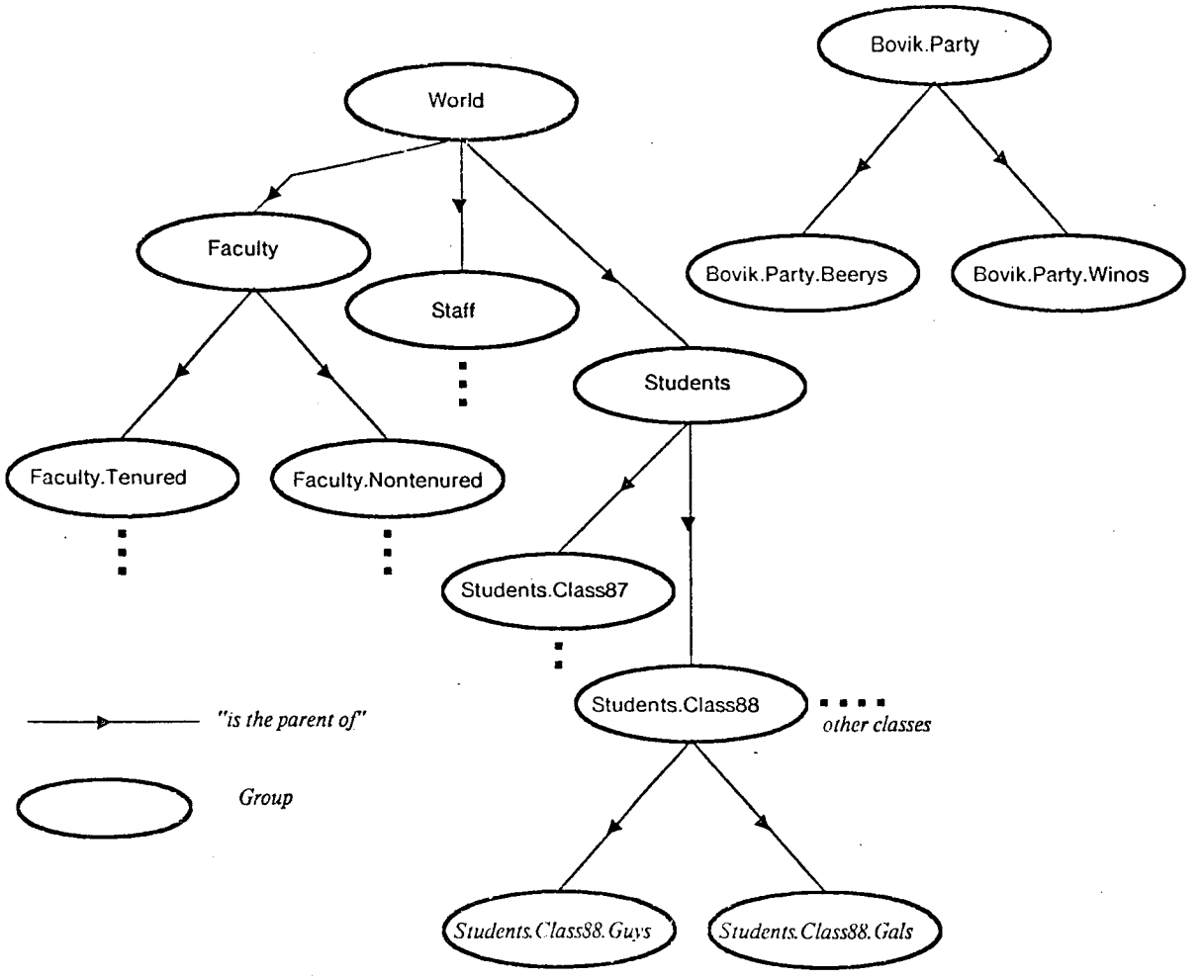

Figure 2: Naming Groups

CreateUser (*username*)

Create a new user, and make him a member of World. Only System may make this call.

RemoveUser (*username*)

Delete the specified user from all groups of which he is a member. Only System may make this call.

CreateGroup (*groupname*, *parent group*)

Create a new group with null membership. The name of the group will be *parent group.groupname*. The caller must have **ModifyMembership** rights on *parent group*. If *parent group* is omitted, the name will be *calling user.groupname*. The calling user is the owner of the group.

RemoveGroup (*group*)

Delete the specified group. All membership links associated with this group are lost. Only the owner of a group can delete it.

AddToGroup (*group1*, *group2*)

Makes *group2* a direct member of *group1*. The caller must possess **ModifyMembership** rights on *group1*.

RemoveFromGroup (*group1*, *group2*)

Make *group2* no longer a direct member of *group1*. The caller must possess **ModifyMembership** rights on *group1*.

GetDirectMembers(*group*)

Return the list of direct members of *group*. The caller must possess **ListMembers** rights on *group*.

GetDirectMembership(*group* or *user*)

Return the list of groups of which the specified *group* or *user* is a direct member. The caller must possess **ListMembership** rights on it.

GetSubdomain(*user*)

Return the current protection subdomain of *user*. The caller must possess **ListMembership** rights on *user*. Typically all VICE subsystems will possess this right on all users, and will make this call when a user tries to log in to them.

ChangeProtection(*group1*, *group2* or *user*, *rights list*)

In the access list entry for *group1*, replace the entry for *group2* or *user* by *rights list*. Only the owner can modify the protection on a group. If the caller is System, *group1* may be a user.

In a simple, non-paranoid implementation of VICE, World would have **ListMembers** and **ListMembership** rights on all groups (and users), and only owners would possess **ModifyMembers** rights on groups.

2.3 Sketch of Protection Subsystem Implementation

For ease of representation, groups and users should be represented as fixed length integers, called group and user IDs. There are mapping tables to convert from names to IDs. IDs are for the internal use of VICE, and are not visible outside it.

The current protection subdomain and the access lists are sorted according to IDs. So finding the available rights on a file is just a matter of running down two sorted lists and ORing the rights masks together.

The relations implicit in the protection domain are captured by three tables:

TableA: *"Is a direct member of"*

For each user and group, this table yields a list of groups of which it is a direct member.

TableB: *"Is a member of"*

For each user and group, this table yields the current protection subdomain. It essentially contains the transitive closure of each entry in **TableA.**

TABLEC: *"Has as direct member"*

For each group, this table yields the list of users or groups who are its direct members.

The most common request to the protection subsystem is likely to be the **GetSubdomain,** from other VICE subsystems, in response to a user login request. This merely involves a lookup of TableB.

When a change is made to a group, the relevant entries in tables A and C are changed. **TableB** now has to be recomputed. In the most general case finding the transitive closure of **TableA** can be an expensive operation. There are a number of approaches to address this issue:

- Restrict groups to have only users. This makes the transitive closure trivial, but severely reduces flexibility.
- Allow groups to have other groups as members, but limit the depth of nesting. This limits the transitive closure to an $O(N^2)$ algorithm rather than $O(N^3)$.
- The direct membership matrix is likely to be sparse. Develop a transitive closure algorithm which exploits this property to run efficiently in the average case.

How is change propagated in the system? There are two views on this, and no consensus has been reached on which of these is preferable:

Slow update — The tables A, B, and C, are replicated at each VICE node and are read-only most of the time. Requests for change are sent to one node which batches them, computes the new tables, and atomically updates them at each node. The slow propagation mechanism is similar to that used by the file system to maintain its replicated database of file locations. The frequency of updates is a system parameter, and is typically about once a day.

Immediate update — The changes take place as they are made — there is no batching of requests. The details of such a mechanism remain to be worked out. It is not clear that the full-fledged protection scheme can support immediate update efficiently enough to make this possible.

Undetermined: Should there be a mechanism to subtract rights? Example, entries in an access list whose rights masks are NANDed rather than ORed during a rights check? This allows rapid, selective revocation.

3 File System Protection

In the file system, the protected objects are files and directories. We feel that it is perfectly reasonable to require that all files in a directory share the same level of protection. Therefore protection can be specified only at the granularity of a directory, and not of an individual file. It is, of course, possible to have directories with exactly one file in them, to handle pathological cases.

The set of rights on a directory are:

LookupFiles Allows GetFileStat() to be performed on files in the directory. Also allows the access list for the directory to be examined.

CreateFiles Enter files into this directory. This is distinct from **WriteFiles** in order to support mail and other similar functions.

ReadFiles Examine the contents of a file in this directory. In the current file system design, this effectively means that Fetches may be done on files in the directory.

WriteFiles Replace an existing file. In the current file system design, this is equivalent to allowing Stores on files in the directory. This right also allows files to be removed from the directory, and allows the access list for the directory to be altered.

The primitives provided to manipulate an access list are:[3]

CreateEntry(*pathname*, *group* or *user*, *initial rights*)
Creates a new entry for *group* or *user* in the access list for *pathname* and assigns *initial rights* to it.

RemoveEntry(*pathname*, *group* or *user*)
Deletes the *group* or *user* from the access list of *pathname*.

ModifyEntry(*pathname*, *group* or *user*, *new rights*)
The entry for *group* or *user* at *pathname* is changed to *new rights*.

ReadAccessList(*pathname*)
Returns the access list for the directory at *pathname*. The requester must possess **LookupFiles** rights on the directory.

When accessing a file, the protection check is made only on the immediate parent of the file — a requester does not have to possess any rights on the intermediate directories of a pathname.

---

[3] Unless otherwise specified, the requester must possess **WriteFiles** rights on the directory in question