GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials (Security 401)"
at http://www.giac.org/registration/gsec

An Introduction to

Implementing Object-Level Security in IBM OS/400

with

Comparisons to Windows and Unix Permissions

by
Jeffrey Alan Gardner

© SANS Institute 2000 - 2005, Author retains full rights.

GIAC Security Essentials Certification (GSEC)
Practical Assignment Version 1.4c - Option 1
May 9, 2005

© SANS Institute 2000 - 2005 Author retains full rights.

Abstract

Access control is a major component of defense in-depth. Object-level security, which is controlling who has access to objects on a system and what type of access they have, is an important part of providing for the appropriate level of confidentiality, integrity, and availability. Securing access using menu security, a concept inherited from the IBM System/38, is not adequate for today's environment. Although the implementation of security is a technical matter, the definition must be prescribed by an organization's security policy to be effective.

The purpose of this paper is to give an introduction to the implementation of object security in OS/400 and show how to keep that implementation simplified, and to make cross comparisons to Unix and Windows permissions where appropriate. It will serve as an introduction to OS/400 object-level authority for those new to or unfamiliar with OS/400, and will be a guide for those expanding their understanding of OS/400 object authority into Unix or Windows file systems and permissions, and how they are implemented in OS/400's Integrated File System (IFS).

The integrated object-level security of OS/400, IBM's midrange object-based operating system, provides the ability to grant fine-grained authority for individual objects to users or groups of users. Security administrators need to understand how to implement OS/400's object-level security simply in its various file systems to allow them to correctly configure the proper access control for the AS/400. Since the OS/400 Integrated File System (IFS) security is based on Unix permissions with object-based extensions, it is important that security administrators understand the Unix concepts and how they apply to OS/400's object-based model. Seeing the parallels in Windows and Unix systems will give the security administrator a broader understanding of object-level security, another layer in the defense in-depth model.

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

Contents

Abstract 2

Chapter 1 – Introduction 4
- Background 4
  - The need for object-level security 4
  - The inadequacy of menu security 4
  - The need for object security policy 5
- Delimitation 5
- Definition of terms 6
- Overview 6
- Significance of the study 7

Chapter 2 – Implementing OS/400 Object Authority 8
- Object-based operating system 8
- OS/400 system values 9
- User profiles 10
  - Special authority 10
  - User authentication 11
- Authority management objects 11
  - Group profiles 11
    - Consolidate object ownership 12
    - Consolidate authorities 12
  - Authorization lists 14
- Object authority and the IFS 14
  - IFS description 14
  - QSYS.LIB file system 15
    - Object and data authority 15
    - Authority examples 17
  - QOPT file system 20
  - root (/), QopenSYS, and UDFS file systems 20
    - Object and data authority 20
    - Overall security considerations 22
    - Authority examples 23
- NetServer and shared resource security 25

Chapter 3 – Summary 27
- Recommendation for further study 27

Appendix 28
- The “Common user id” myth 28

References 30

© SANS Institute 2000 - 2005 Author retains full rights.

Chapter 1 – Introduction

Access control is a major component of defense in-depth. Object-level security, which is controlling who has access to objects on a system and what type of access they have, is an important part of providing for the appropriate level of confidentiality, integrity, and availability. Securing access using menu security, a concept inherited from the IBM System/38, is not adequate for today's environment. Although the implementation of security is a technical matter, the definition must be prescribed by an organization's security policy to be effective.

The purpose of this paper is to give an introduction to the implementation of object security in OS/400 and show how to keep that implementation simplified, and to make cross comparisons to Unix and Windows permissions where appropriate. It will serve as an introduction to OS/400 object-level authority for those new to or unfamiliar with OS/400, and will be a guide for those expanding their understanding of OS/400 object authority into Unix or Windows file systems and permissions, and how they are implemented in OS/400's Integrated File System (IFS).

Background

The need for object-level security

A proper defense in-depth strategy has access control as a major component. Object-level security provides a low-level, inner layer of protection by implementing access control on individual objects. The essence of this is that each authenticated user on a system is authorized to perform certain actions on each object to which he needs access in order to perform his task. This provides for the implementation of the access control principle termed *least privilege*–"a principle of giving the least amount of access possible to accomplish a task".[1]

One recent survey and report showed that 70 percent of security breaches were caused by insiders.[2] Security at the object level provides another protection against malicious and accidental damage.

The inadequacy of menu security

A security model inherited from the System/38 roots of the AS/400 is that of menu security. It developed in the days when the primary means of access to the System/38 and the AS/400 was via twinax connected non-programmable terminals (the venerable "dumb terminal").

Most of the access in this environment is controlled by the initial menu that is assigned to the user. If a program is not on the user's menu, he cannot call that program, and cannot perform the function. By restricting users' access to a command line, the range

[1]SANS Institute, Track 1–SANS Security Essentials, Volume 1.2, Defense In-Depth, (SANS Press, September 2004), p. 119.

[2]Marguerite Reardon, "Securing data from the threat within," CNET News, January 11, 2005, April 20, 2005. <http://news.com.com/Securing+data+from+the+threat+within/2100-7347_3-5520016.html?tag=st.prev>

© SANS Institute 2000 - 2005 Author retains full rights.

of commands and functions is controlled by the menu. This is referred to as "menu security" because the menu controls the function or application that can be accessed.

The security structure was often a simplistic one in which users had much authority, many times all authority, to objects on the system. However, their access to the objects was controlled by the menu options they were given.

The problem with this method is that menu security is only effective for 5250 and Telnet terminal methods of AS/400 access. This problem has been discussed for some time in AS/400 publications.[3] This has led some to erroneously conclude that OS/400 security does not apply across all access interfaces to the AS/400.[4] In today's environment, there are so many ways to access objects on an AS/400 other than through the "green screen" terminal access that security at the object level is essential. Menu-based security is no longer adequate.

The need for object security policy

The technology of object-level security or any other kind of security is not the total answer to security. A security policy, with the support of management, is crucial to define what must be secured and at what level. Policy defines technology usage. Once the doctrine is set in the security policy, technology can provide the methods for achieving it. Policy is "what" – Technology is "How."[5]

Delimitation

This paper will cover how to implement the integrated object authority supplied by V5R3 of the OS/400 operating system. There are other security models discussed in the AS/400 press, but they will not be covered. Two that are discussed frequently are:

- using system-supplied "exit points," a method of defining a user written "exit program" to be called during a system function,[6]
- using what is called "Application-only Access" that recommends only granting authority to programs that inherit the required authority at run time. All other objects would have no private authorities for any user and no *PUBLIC authority.[7]

These other models would make good studies of their own.

---

[3]John Earl, "Purchased Software Can Jeopardize your Security," iSeries News, December 1, 1999. (Also available at iSeries Network, December 1, 1999. April 15, 2005. <http://www.iseriesnetwork.com/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=3255&channel=&subart=>)

[4]Midrange dot COM Mailing List Archive, "Re: RMTCMD Anomaly," May 19, 2000, April 20, 2005. <http://archive.midrange.com/midrange-l/200005/msg00996.html>

[5]SANS Institute, Track 1–SANS Security Essentials, Volume 1.2, Defense In-Depth, p. 63.

[6]John Earl, "Exit Programs Tighten AS/400 Security," PowerTech.Com, [no date], April 25, 2005. <http://www.400security.com/pt-about_news-art_FA0603.html>

[7]Wayne Evans, "Application-only Access: An AS/400 Resource Security Strategy," Wayne O. Evans Consulting, [No date] March 29, 2005 <http://www.woevans.com/AOA.pdf>

© SANS Institute 2000 - 2005 Author retains full rights.

Definition of terms

To clarify the usage of terms used in this paper, the following definitions are given:

"Object" refers to anything in OS/400. OS/400 stores all information as objects, and each object has an object type such as library (*LIB), program (*PGM), file (*FILE), data area (*DTAARA), etc. IBM-supplied objects have names that begin with the letter "Q" so that they are easily recognized (as long as the user does not name his own objects starting with a "Q" also.)

"Authority" is the OS/400 term for the assigned ability to perform certain functions on an object. Systems such as Unix and Windows use the term "permissions." The OS/400 graphical interface iSeries Navigator also uses "permissions." These terms refer to the same concept.

IBM has gone through several name changes for the AS/400 mostly based on hardware enhancements. The various versions do not affect the discussion here because object authorities have always been a part of OS/400. This paper will use the following terms:

> "AS/400" is used for the name of the computer. This is the original name of the system, but it has been renamed twice, first to iSeries, and more recently i5. Many users still use the original name AS/400 and will occasionally use the names interchangeably.
>
> "OS/400" is used for the name of IBM's midrange object-based operating system that is used on an AS/400 machine. The most current release of OS/400, V5R3, has been renamed i5/OS, but many users still use the term OS/400.
>
> "iSeries Navigator" is the Windows-based graphical interface for managing an AS/400. Previous versions were called "Operations Navigator." The commonly used abbreviation "Ops Nav" will also be used in this paper.

As IBM moves more and more of the management functions into iSeries Navigator, the terminology used changes in some cases. In this paper, the Ops Nav term will be supplied after the OS/400 "green screen" term.

Overview

Chapter 2 will start with a brief introduction to the OS/400 object-based operating system concepts. Then it will cover several system objects that affect security and security management across all of OS/400 in each file system of the IFS. Then object authority will be discussed in several of the IFS file systems, starting with the "native" file system and progressing to other IFS systems. A brief discussion of OS/400's file serving service, NetServer, follows. Chapter 3 has a summary and recommendations including further projects.

© SANS Institute 2000 - 2005 Author retains full rights.

Significance of the study

There are some within the AS/400 user community who have a misconception that object security does not work over all access methods to the AS/400, others who believe it is overly difficult to implement, and some who still depend on menu security. The information in this study will contribute to dispelling these ideas.

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

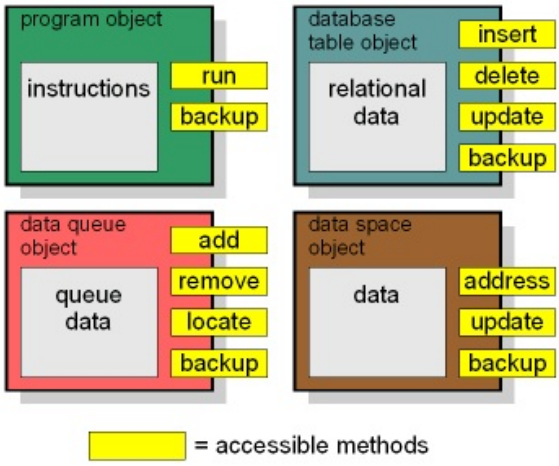
Chapter 2 – Implementing OS/400 Object Authority

The integrated object-level security of OS/400, IBM's midrange object-based operating system, provides the ability to grant fine-grained authority for individual objects to users or groups of users. Security administrators need to understand how to implement OS/400's object-level security simply in its various file systems to allow them to correctly configure the proper access control for the AS/400. Since the OS/400 Integrated File System (IFS) security is based on Unix permissions with object-based extensions, it is important that security administrators understand the Unix concepts and how they apply to OS/400's object-based model. Seeing the parallels in Windows and Unix systems will give the security administrator a broader understanding of object-level security, another layer in the defense in-depth model.

Object-based operating system

OS/400 is an object-based operating system, i.e., all information on an AS/400 is kept by the operating system as objects. This is in contrast to systems such as Unix and Windows which are byte-string file-based, i.e., all information is kept in simple byte-string files.

Operations, or "methods" in object-oriented terminology, that are allowed on an object are defined by the object type which is stored with the information in the object. The example below shows four object types and their associated methods.[8]



For instance, one cannot execute a CALL command on a database file (object type *FILE, attribute PF) and run its contents as though it were a program. Likewise, there is no "read" or "write" function to view or change the contents of a program object (*PGM), This encapsulation of data and methods provides integrity for the operating environment.

[8] IBM Corporation, "Overview of iSeries (OS/400) Architecture," Virtual Innovation Center for Hardware, [No date], April 15, 2005. <http://www-1.ibm.com/servers/enable/site/porting/iseries/overview/overview.html>

© SANS Institute 2000 - 2005 Author retains full rights.

Each object in OS/400 is owned by an object called a user profile, *USRPRF. Even user profile objects are owned by a profile. A *USRPRF might be owned by itself, or more likely it would be owned by another profile such as the security officer profile. (User profiles are discussed in more detail below.)

OS/400 stores several security attributes in every object. Those that are specifically related to our discussion of security are:

- the name of the user profile object (*USRPRF) that owns the object
- the owner's authority to the object
- public authority to the object. *PUBLIC authority is used when a user doesn't have private, group, or inherited authority to an object.
- the name of an optional primary group profile and the group's authority

Storing these attributes on the object gives a performance advantage in checking. The fastest authority checking occurs for owner, *PUBLIC, and primary group authority since the system only has to check the object for the authority.

Unix files are similarly configured except that all Unix files have a "group owner" comparable to the OS/400 optional primary group. On the other hand, not all Windows files have their own permissions. By default, new files and folders (directories) do not have their own permissions. Rather, permission checking defers to the parent directory (which might propagate all the way up to the root directory).

OS/400 system values

System values are objects that contain values or parameters that apply across the entire system. (Ops Nav uses both terms *system values* and *policies*.) These objects can be compared to the Windows *policies* and *user rights*.

The first system value that needs to be considered in a discussion of security is named QSECURITY. This is the overall security level of the system. The valid values are:

10=Physical security only (no longer supported).
20=Password security only
30=Password and object security
40=Password, object, and operating system integrity
50=Password, object, and enhanced operating system integrity

Level 10 is supported only when a machine is already at that level and is upgrading to a new version of OS/400. Level 10 makes the AS/400 act as though it were simply a very powerful DOS machine with no object authority checking. If you can get to a connected terminal, you have full access to the system. Level 20 grants all object authority to all users, so it too bypasses object authority checking. Level 30 is the absolute minimum to be considered since it uses user profiles and passwords and users have different authorities. Level 40 is the recommended one because it adds protection against accessing system objects in ways other than the supplied APIs. New systems ship with

© SANS Institute 2000 - 2005 Author retains full rights.

the QSECURITY system value set to 40. A value of 50 is recommended only for very high security environments.

Other system values apply only to certain file systems and will be discussed under these. There are also other security-related system values that do not relate to object security such as password criteria, invalid signon attempts, that will not be covered.

User profiles

Users are defined via *user profiles*, object type *USRPRF. (Ops Nav–*users*.) Among the many attributes stored in the *USRPRF, four are of interest in this discussion:

- the list of objects owned by that profile
- the list of private authorities to objects not owned by the profile
- a list of *Special Authorities (SPCAUT)* (Ops Nav–*privilege class*)
- a main group profile (if any) and a list of supplemental group profiles (these will be discussed below in the section "Authority management objects" below)

One of the benefits of storing the list of owned objects and the list of private authorities in the user profile object itself is that when the profile is deleted, references to that profile are deleted also. All of the private authorities that the profile has are deleted when it is deleted. Also, since all objects must be owned by a *USRPRF object, those objects owned by the profile must be assigned to another profile, or they must be deleted. The Delete User Profile (DLTUSRPRF) command has a parameter that specifies whether to automatically change the owner of objects, automatically delete the objects, or abort the operation if the profile owns any objects. (Ops Nav–the *delete* function for a user has the same options.)

This avoids the situation encountered in Windows with leftover authorities and objects with no owner (except for a mysterious SID), and it eliminates the reuse problem sometimes found on Unix systems when a UID is reused. Since a *USRPRF is an object, not just a number as in Windows and Unix, deleting a user profile deletes the object, and making a new profile makes a whole new object–all traces of the old profile are gone.

Special authority

Two of those special authorities are of interest to this discussion. One is the *All object (*ALLOBJ)* authority (Ops Nav–*all object access*). This enables the user to perform any operation on any object without restriction–the same unrestricted access that the system security officer profile, QSECOFR, has. Obviously, this special authority must be handed out with extreme caution.

In comparison, in a Windows environment an Administrator or Domain Administrator can be restricted, for example, from viewing the contents of a file. He then has to change permissions or take ownership of an object to work around that. *ALLOBJ authority trumps any individual authority an object has. One could revoke all authority to an object from a profile with *ALLOBJ, and the *ALLOBJ special authority will still allow that user to access the object. It is a very powerful authority indeed.

© SANS Institute 2000 - 2005 Author retains full rights.

Another special authority of interest is the *save system (*SAVSYS)* authority (Ops Nav–*save and restore*). A user profile can save any object that it owns or any that it has authority to use. *SAVSYS special authority allows the user profile to save any object on the system, even if it has no authority to the object. It can also restore any object provided it has been granted *USE authority to the restore commands. (All OS/400 restore commands are shipped with no *PUBLIC authority in order to restrict who can restore objects to the system.) For instance, a file could be saved or restored, but the user profile would have no authority to read the contents of the file. This would be given to a system operator whose job involves running system backups.

User authentication

In OS/400 at security level 20 and above, the primary method for user authentication is by logging on with a user profile and password. As of OS/400 V5R2, Enterprise Identity Mapping and Kerberos authentication is also available for many functions to provide for a single signon environment[9].

In either case, the authentication maps to an OS/400 user profile object which is authorized to perform functions on an object. (See *Appendix, The "Common user id" myth*, for a discussion of an enduring myth in the AS/400 community that can completely negate any effort spent in configuring correct object-level security.)

Authority management objects

OS/400 provides two objects that assist in managing and simplifying object authorities in OS/400, Group Profiles and Authorization Lists. Using these two objects can produce a highly customized but simplified and manageable authority structure.

Group profiles

A group profile (Ops Nav–*Group*) is a regular *USRPRF object. OS/400 identifies it as a group profile when it is assigned a group id (GID) or when another *USRPRF is added to it as a group. A group profile can be used as a logon profile or to run jobs. However, setting up separate *USRPRF objects as group profiles to be simply authority holders simplifies management. Users and groups are two different "objects" in Windows and Unix, unlike in OS/400, so a group in Windows and Unix cannot log on or run jobs.

Each user profile can be a member of up to 16 group profiles using the Create User Profile (CRTUSRPRF) or Change User Profile (CHGUSRPRF) commands. (A group profile cannot be a member of another group.) In the partial view of the prompted CHGUSRPRF command below, the user profile USERNAME is a member of GROUP1 as its group profile, and GROUP2 as a supplemental group.

[9] IBM Corporation, "Enterprise Identity Mapping," iSeries Information Center–V5R2, [No date], April 19, 2005. <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzalv/rzalvmst.htm>

© SANS Institute 2000 - 2005 Author retains full rights.

```
================================================================
                        Change User Profile (CHGUSRPRF)
User profile . . . . . . . . . . USRPRF       > USERNAME
.
.
.
Group profile  . . . . . . . . . GRPPRF         GROUP1
Owner  . . . . . . . . . . . . . OWNER          *USRPRF
Group authority  . . . . . . . . GRPAUT         *NONE
Group authority type . . . . . . GRPAUTTYP      *PRIVATE
Supplemental groups  . . . . . . SUPGRPPRF      GROUP2
                          + for more values
================================================================
```

Consolidate object ownership

Group profiles can be used to control and consolidate object ownership. Normally a *USRPRF owns and has all authority to objects it creates. In the above example, the "Owner" parameter is listed as *USRPRF, which says the user profile will own all created objects.

Specifying *GRPPRF instead says whatever group profile is named on the GRPPRF parameter, GROUP1 in this example, will own any objects the user profile creates, and that group profile will be granted all authority.

Optionally, instead of making the group profile the owner, specifying the "Group authority (GRPAUT)" parameter grants that authority to the group profile. Then the "Group authority type (GRPAUTTYP)" parameter determines if the authority is a private authority (stored on the group profile) or a primary group authority, which makes the group profile the primary group for the object and stores the authority on the object (giving the performance enhancement as was discussed previously). A group profile can become an object's primary group also by using the Change Object Primary Group (CHGOBJPGP) command or the Change Primary Group (CHGPGP) command (Ops Nav–"Primary Group" button in Permissions dialog for the object).

An example of a common use of this group profile ownership and authority feature is in an environment with multiple programmers working on a single project from a common library.

This feature can be compared to the Unix "Group owner permission" on files. In Unix there is only one group identified on an object, and every object has a group owner. An OS/400 object can have no primary group, have a group profile specified as its primary group, or be owned by a group profile. The same group profile cannot be both the owner and the primary group.

Consolidate authorities

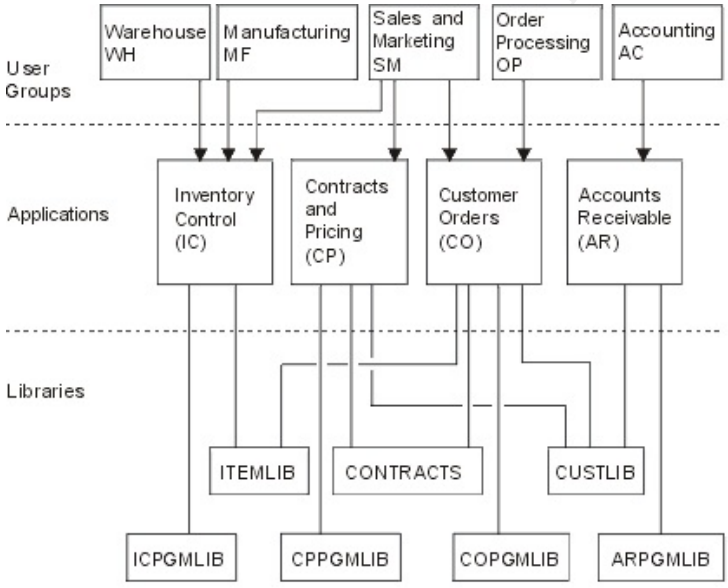
Group profiles also can be used to consolidate authorities. Although each user profile can be granted authority to each object it needs for the tasks it performs on the system, with a large number of users and the changes that invariably occur, managing them can quickly become overwhelming. In addition, since user profiles and their lists of authorities are saved during a system save (SAVSYS) or a Save Security Data

© SANS Institute 2000 - 2005 Author retains full rights.

(SAVSECDTA), as users and authorities are added, those saves can become longer and longer in time and space requirements.

A user profile inherits object authorities and special authorities from the group profile. If the user profile has no private authority to an object and has not been specifically excluded by revoking all authority to the object for the user, each group to which the profile belongs will be checked for authority. The authorities from all group profiles listed on the user profile will be combined to determine if the user has enough authority to an object.

Grouping users by function is an efficient way to manage authorities on a system. The following diagram[10] gives an example of how a company might define user groups based on function and access requirements for certain applications and objects. Each user group listed would be made a *USRPRF object to which other users are added, making them group profiles.



The grouping of authorities is very similar to the way Windows uses groups–multiple groups can be authorized to a file, and users from the various groups inherit that authority.

This is different from the Unix single group ownership of a file.[11] The Unix concept can be compared to OS/400's granting the group profile authority to the file or specifying the primary group on an object.

---

[10]IBM Corporation, "Identifying User Groups," iSeries Information Center–V5R3, [No date], April 15, 2005. <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/index.htm?info/rbapk/rbapkrbapk4i3exampleidug.htm>

[11]SANS Institute, Track 1–SANS Security Essentials, Volume 1.6, Unix Security, (SANS Press, September 2004), p. 191.

© SANS Institute 2000 - 2005 Author retains full rights.

Authorization lists

An authorization list (*AUTL) is another object that provides authority management. An authorization list is an object that contains a list of other objects that all need the same security settings. One way to think of it is that an *AUTL works in reverse of a group profile. Whereas a group profile grants the same authority over an object to all users in the group, an *AUTL grants a specified user profile the same authorities over a list of objects.

For an example of this, suppose there is a set of files whose object authorities are the same for any user profile that needs to access them. Rather than authorizing each user to each file, there are two options: (1) group the users and authorize the group profile or (2) add all of these files to an authorization list and grant authority for the user or group that needs to access the set of files *to the authorization list object*. Whatever authority the user profile has to the *AUTL, it has to the objects on the list.

However, start with the situation above but add the stipulation that each user needs authority different from other users to the set of files. This cannot be done using a group profile. To group the authorities efficiently, the *AUTL must be used.

An advantage of the *AUTL , like the group profile, is that the number of authorities has been reduced making for smaller user profiles. Another advantage to authorization lists is in managing authorities on files (object type *FILE). Authorities cannot be changed on a file when it is open. However, if authorities are managed via an *AUTL, they can be changed because all authority actions are performed on the *AUTL, not on the file. This can be a big help in always-on environments.

The example given above of using a group profile and the group ownership setting for programmers working on the same projects can be recast using the authorization list. Instead of the group profile owning all objects, set up a common library on an authorization list. Then grant authority to all programmers that need to work in that library. Any object created in that library will have the *AUTL as its authority setting, and all programmers authorized to the list will have authority to the objects as well.

Object authority and the IFS

OS/400 supports multiple file systems under the umbrella name of the Integrated File System (IFS). Object authority applies at all times across all file systems.

IFS description

To display the file systems on an AS/400, execute the Work with Object Link (WRKLNK) command from a command line or open iSeries Navigator and, under the system name, select *File systems/Integrated File System*. iSeries Navigator shows them more clearly since the WRKLNK view shows all of the file systems under the "/" or "root" file system (which is literally how they are structured).

The IFS file systems which will be discussed in this paper are:

- QSYS.LIB–the original file system. Objects in libraries, one level deep.

© SANS Institute 2000 - 2005 Author retains full rights.

- QOPT–the AS/400's CD-ROM
- root (/)–directory and stream file access, case-insensitive names (like on a PC)
- QOpenSys–like root but uses case-sensitive names (like Unix systems)
- User defined file system (UDFS)–file systems located on auxiliary storage pools. Same attributes as the root system, but can be configured to use case-sensitive or case-insensitive file names.

The root, QopenSYS, and UDFS will be discussed together because the security is handled the same in these file systems.

The following file systems will not be discussed because they access remote file systems, and security is handled by the remote server:

- QNTC–for accessing a Windows server either remotely or on an integrated Windows server (an AS/400-hosted PC hardware running Windows and using AS/400 disk as PC disks)
- QNetWare–for accessing a remote NetWare server
- QFileSrv.400–for accessing a remote AS/400's IFS
- NFS–for accessing a NFS server

In addition, the QDLS file system (document and library services file system, also called shared folders) will not be covered. QDLS is the Document Library Object file system for OfficeVision/400 and for providing shared folder access for PCs. It existed before the development of the IFS. It is a slow-performing file system, and its use is no longer recommended since the root file system of the IFS has much better performance.[12]

QSYS.LIB file system

The QSYS.LIB file system is the "native" file system of the AS/400, inherited from the System/38. This file system supports the hundreds of object types unique to OS/400. The structure of this file system is a single-level library structure rather than a multi-level directory structure. All objects exist in a library, and all libraries exist in the system library QSYS which is sometimes called the "system context."

Object and data authority

Authorities in QSYS.LIB are managed using the commands Grant Object Authority (GRTOBJAUT), Revoke Object Authority (RVKOBJAUT), and Edit Object Authority (EDTOBJAUT), and using iSeries Navigator under *File systems*.

Because of the object-based nature of OS/400, each authority has two parts:

- *Object authority* allows certain operations on the *object as a whole*
- *Data authority* allows operations on the contents or *data* of the object.

[12]Jake Kugel, "Destination: IFS, Part 1," iSeries News, June 1, 2002. (Also available at iSeries Network, June 1 2002. April 29, 2005. <http://www.iseriesnetwork.com/resources/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=14477&channel=art&PageView=Search>)

© SANS Institute 2000 - 2005 Author retains full rights.

Each separate object and data authority is described below. Also described are grouped authority that can be used in all authority commands. The descriptions below of the authorities are summarized from the OS/400 Online Help Information (help text) for the "Authority (AUT)" parameter of the abovementioned commands.

Individual data authorities

- *READ–get the contents of an entry in an object.
- *ADD–add entries to an object.
- *UPD–(Update) change the entries in an object.
- *DLT–(Delete) remove entries from an object.
- *EXECUTE–run a program or locate an object in a library.

Individual object authorities

- *OBJOPR–(Object operational) look at the description of an object and use the object. (*OBJOPR authority can be considered the most fundamental authority. Without it, no other authority is effective.)
- *OBJMGT–(Object management) specify the security for the object, move or rename the object, and add members to database files.
- *OBJEXIST–(Object existence) control the object's existence and ownership.
- *OBJALTER–(Object alter) alter the attributes of an object. If the user has this authority on a database file, the user can add and remove triggers, add and remove referential and unique constraints, and change the attributes of the database file. If the user has this authority on an SQL package, the user can change the attributes of the SQL package. This authority is currently only used for database files[13] and SQL packages.
- *OBJREF–(Object reference) reference an object from another object such that operations on that object may be restricted by the other object. If the user has this authority on a physical file, the user can add referential constraints in which the physical file is the parent. This authority is currently only used for database files.

Grouped authorities

- *USE–The user can perform basic operations on the object, such as running a program or reading a file. The user cannot change the object. *USE authority provides *OBJOPR, *READ, and *EXECUTE.
- *CHANGE–Change authority allows the user to perform all operations on the object except those limited to the owner or controlled by object existence authority and object management authority. *CHANGE authority provides *OBJOPR and all data authority.
- *ALL–The user can perform all operations except those limited to the owner or controlled by authorization list management authority. The user can control the object's existence, specify the security for the object, change the object, and

[13] *OBJALTER and *OBJREF authorities can apply to database files because DB2-400 is integrated into OS/400, and database functions are controlled with OS/400 object authorities rather than through an application-level security model.

© SANS Institute 2000 - 2005 Author retains full rights.

perform basic functions on the object. The user also can change ownership of the object.
- *EXCLUDE–The user cannot access the object.
- *AUTL–the public authority of the authorization list securing the object is used for *PUBLIC authority.

Authority examples
We will show how these authorities work using some of the most commonly used objects in OS/400–file (*FILE), program (*PGM), and library (*LIB).

*File object (*FILE)*
A file in OS/400 is an object used for program I/O–a program reads from and writes to *FILE objects. There are display device files (attribute DSPF), printer files (PRTF), database physical files (PF), database logical files (LF), among others.

For our example, we will consider a database physical file. This is very similar to a data file in Windows and Unix. The data portion of this object is often what is meant when the word *data* is used–information stored as records in a file. To read existing records, *READ authority is required; to add a new record, *ADD is required; to delete a record, *DLT is required; to change an existing record, *UPD. Although *DLT will allow a user to delete one record at a time, to use the Clear Physical File Member (CLRPFM) command to delete all records at once, object management authority (*OBJMGT) is required.

This type of fine-grained control on allowed data operations on a file is not available in Unix or in Windows. In Unix, R permission allows reading a file, W allows writing to the file (adding new, updating or deleting existing records) including completely clearing the file and rewriting all its content. The original NTFS file permissions have been expanded and subdivided in Windows 2000 and above into finer controlled permissions via the Advanced Permissions entry,[14] however the data access is still not as fine-grained as these data authorities.

Physical files can have multiple *members*, each containing different data records. This characteristic of a database physical file can be compared to a directory in Windows and Unix. In fact, iSeries Navigator shows physical files as directories and the members as files. However, each member of a file has the same authority as the *FILE object.

This function is duplicated in Windows by leaving the default “Inherit permissions” attribute set on a file or directory. In Unix, however, changing the directory permissions does not apply to the files within it.

[14]Jeffrey R. Shapiro and Jim Boyce, Windows 2000 Server Bible. (New York: Hungry Minds, Inc., 2000), p. 807.

© SANS Institute 2000 - 2005 Author retains full rights.

*Program object (*PGM)*
A program is an object that executes instructions, and it is similar in function to program files in Windows and Unix, except that it is an encapsulated object. The data portion of a program contains the program's executable code. The authorities needed to execute a CALL command on a program are *OBJOPR and any data authority. Typically *USE authority is granted which results in *OBJOPR, *READ, and *EXECUTE.

A program has an attribute, USRPRF, that controls which profile's authority is used to execute the program's instructions. It can be set to *USER or *OWNER at compile time or using the Change Program (CHGPGM) command. *USER is the default setting and causes the program to use the authorities of the user running the program. *OWNER causes the program to use the authorities of the program's owner. This is called "using adopted authority" or simply "adopting authority." All functions performed by the program, including CALLs to other programs, adopt the authority of the *OWNER program's owner.

This attribute to adopt authority performs like the Unix set-UID flag on an executable file. The Windows "Run as..." setting is different in that the setting is not an attribute of the executable file itself, rather it is set on a shortcut. The Windows Runas command functions the same way by starting a process running under a different user id. Also, both methods of "Runas" can specify any user id. This is similar to using the USRPRF parameter on the OS/400 Submit Job (SBMJOB) command which allows an entire job to be run under a user profile and its authorities different from the one submitting the job. (The *USRPRF submitting the job must have *USE authority to the other user profile.)

Principles such as *least privilege* and *access control* should be practiced when using *OWNER programs:

- carefully parcel out *USE authority to a *OWNER program in order to control which users can run it
- restrict *READ authority to it because *READ authority allows a user to run debug on a program and possibly bypass the established security by changing the program's variables and parameters. This is an identical warning as given in the SANS Unix Security book concerning set-UID files in Unix.[15] Accomplish this by granting only *OBJOPR and *EXECUTE authorities, enough to CALL the program but not "read" it in the debugger.
- the owner of a *OWNER program should be one that has just the amount of authority it needs to do the task, rather than the "quick and dirty" method of making the owner QSECOFR, the system security officer profile, or some other *ALLOBJ authority user profile.
- a *OWNER program should perform a minimal number of functions. Design the application to keep that program at the bottom of the call stack so it does not propagate its adopted authority to programs it calls.
- if the *OWNER program must call other programs, use the library qualifier on the CALL command, which forces the call to the specific program in the specific

[15]SANS Institute, Track 1–SANS Security Essentials, Volume 1.6, Unix Security, p. 269.

© SANS Institute 2000 - 2005 Author retains full rights.

library, rather than allow a library list call, which can lead to a “Trojan horse” effect by someone manipulating his library list.

*Library object (*LIB)*
A library is a container object in OS/400 in which other objects exist. A directory in Windows or Unix is also a container object. However, whereas directories in Windows and Unix can contain other directories, libraries cannot contain other libraries.

The object authorities to a library control access to the library object itself. The data authorities control access to the library’s data, which are the objects contained in the library.

For instance, in order to add a new file or program to a library, a user needs to have *ADD data authority to the library. The other data authorities, *READ, *UPD, *DLT, *EXECUTE, control operations on the objects in the library, but are limited by the authorities on the objects themselves. For example, a user profile might have *DLT data authority for a library, allowing it to remove entries from the library (delete an object), but if it does not have *OBJEXIST on an object, it cannot delete that object. This is different from Windows and Unix in which directory permissions control “object” functions for the files in the directory.

Each library has a setting called Create Authority (CRTAUT), which is specified on the Create Library (CRTLIB) or Change Library (CHGLIB) command. The setting controls the *PUBLIC authority on new objects. The setting can have the values *USE, *CHANGE, *ALL, *EXCLUDE, or the special value *SYSVAL, which says to use the value in the system value QCRTAUT which can have the values *USE, *CHANGE, *ALL, or *EXCLUDE. In iSeries Navigator under the QSYS.LIB file system, open permissions for a library and click the “New objects” button.

A new object does not inherit any authority from the library’s authority. There are two scenarios for making new objects:

- using the various CRTxxx commands to make the new object, e.g. Create Physical File (CRTPF), the new object only has owner and *PUBLIC authorities granted. The owner is granted *ALL authority. The new object’s *PUBLIC authority can be specified using the “Authority (AUT)” parameter on the command, or it can default to the library’s CRTAUT setting. This is in contrast to Windows and Unix in which new files inherit permissions from the directory in which they are made.
- using a Create Duplicate Object (CRTDUPOBJ) or a Copy File (CPYF) to make a new object based on an existing one, all authorities for the old object are duplicated onto the new object except that the owner of the new object is the user profile which made the new object and it is granted *ALL authorities.

With these basics understood, we can look at the rest of the IFS and how it handles object authority.

© SANS Institute 2000 - 2005 Author retains full rights.

QOPT file system

The QOPT file system is used to access the AS/400's optical CD-ROM drive. Since it is a removable medium, securing directories and objects is not available. However, it is possible to restrict users' access to the CD-ROM itself. A device description object (*DEVD) is used to access the CD-ROM, and therefore the QOPT file system. Control access to the device description, and access to the file system is controlled.

First, revoke all authority (RVKOBJAUT) from *PUBLIC to the device description (named OPT01 by default). Then grant *USE authority to users or groups that are to access the CD-ROM using the GRTOBJAUT command.

This can also be accomplished using an authorization list. Make a *AUTL (perhaps calling it OPT01). Change the device description's authority to use the authorization list GRTOBJAUT OPT01 AUTL(OPT01), change *PUBLIC authority to *AUTL, then use the Add Authorization List Entry (ADDAUTLE) command to add users and their authorities.

root (/), QopenSYS, and UDFS file systems

These file systems are designed to emulate both a Windows PC file system and a Unix file system. The root file system is a case-insensitive system, QOpenSys is case-sensitive to mimic Unix, and the UDFS can be defined either way. The most common objects in the systems are the directory (*DIR) and the stream file (*STMF).

The primary reason for the QOpenSys file system is to provide file access to OS/400's Qshell POSIX-compliant Unix command interpreter and utilities[16], and to OS/400 PASE (Portable Application Solutions Environment), the integrated Unix run-time environment designed to allow Unix applications to run under OS/400.[17]

Object and data authority

Authority in the IFS is managed using the Change Authority (CHGAUT) and Work with Authority (WRKAUT) commands, and via iSeries Navigator *File systems*. A unique aspect of these file systems is that authority checking must satisfy all three conditions, OS/400, PC, and Unix. This requires Unix-like permissions with an object twist to them. Because of this mix of Unix- and OS/400-type authorities/permissions, managing authority in these file systems can be bewildering to OS/400-oriented users and to Unix-oriented users alike.

Authority is composed of object and data authority, as in QSYS.LIB. The OS/400 Online Help Information is summarized below for these authorities, and if appropriate, compared to the grouped authorities in QSYS.LIB:

[16]IBM Corporation, "Qshell," iSeries Information Center–V5R3, [No date], May 5, 2005. <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzahz/intro.htm>

[17]IBM Corporation, "OS/400 PASE," iSeries Information Center–V5R3, [No date], May 5, 2005. <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/index.htm?info/rzalf/rzalfintro.htm>

© SANS Institute 2000 - 2005 Author retains full rights.

Individual data authorities
The data authorities are the familiar Unix RWX permissions and combinations.

- *R–open an object for reading. Provides *OBJOPR and *READ.
- *W–open an object for writing, for a directory add, delete, and rename files in the directory). Provides *OBJOPR, *ADD, *UPD, *DLT.
- *X–run a program file or search a library or directory. Provides *OBJOPR, *EXECUTE.

- *RX–perform basic operations on the object, such as run a program or display the contents of a file. The user is prevented from changing the object. Provides*OBJOPR, *READ, *EXECUTE (like *USE).
- *RW–view and change the contents of an object. Provides *OBJOPR, *READ, *ADD, *UPD, *DLT (which on a data file is like *CHANGE).
- *WX–change the contents of an object and run a program or search a library or directory. Provides *OBJOPR, *ADD, *UPD, *DLT, *EXECUTE.
- *RWX–perform all operations on the object except those limited to the owner or controlled by object existence, object management, object alter, and object reference authority. The user can change the object and perform basic functions on the object. Provides *OBJOPR and all the data authorities (like *CHANGE).
- *EXCLUDE–Exclude authority prevents the user from accessing the object.
- *AUTL–The public authority of the authorization list specified in the AUTL parameter is used for the public authority for the object.

Individual object authorities

- *OBJEXIST–delete, save, or restore a file
- *OBJMGT–rename, move, or look at authorities (*but not set*–this is different from QSYS.LIB)
- *OBJREF, *OBJALTER–these currently only apply to database files in QSYS.LIB and have no meaning in the IFS.[18]

These descriptions are summarized in the following table of comparisons (*W and *WX are incorrect in the source and have been corrected).[19] The first thing to note is that only data authorities are granted to the object using the RWX format–no object authorities are granted. Also, *OBJOPR, called an *object authority* in QSYS.LIB, is called a *data authority* in the IFS model.

[18]Jeff Parker, "Integrated File System Security–Fundamentals," COMMON Conference Handout, March 2003, Session 440138.

[19]Carol Woodbury, "Security Patrol: Security Considerations for the Integrated File System (IFS)" MCPress Online, April 2003, April 11, 2005. <http://www.mcpressonline.com/mc?50@118.ksLEccBImF2.9@.6ae64331>

© SANS Institute 2000 - 2005 Author retains full rights.

Authorities	*R	*W	*X	*RW	*RX	*WX	*RWX
Object							
*OBJMGT							
*OBJEXIST							
*OBJALTER							
*OBJREF							
Data							
*OBJOPR	X	X	X	X	X	X	X
*READ	X			X	X		X
*ADD		X		X		X	X
*UPD		X		X		X	X
*DLT		X		X		X	X
*EXECUTE			X		X	X	X

We can also compare how the grouped authorities in QSYS.LIB map into the IFS version of the authorities. Authority for QSYS.LIB objects can be displayed using the object-oriented command DSPOBJAUT and using the IFS-oriented command DSPAUT. Execute from an OS/400 command line a GRTOBJAUT OBJ(lib_name/obj_name) command to grant *EXCLUDE, *USE, *CHANGE, and *ALL to an object. Display those authorities using the DSPOBJAUT command, then display them using the DSPAUT OBJ('/qsys.lib/lib_name.lib/obj_nam.obj_type'). The following figure shows the two models merged into a single display. As expected, *EXCLUDE grants no authorities, and object authorities (other than *OBJOPR) are only granted when granting *ALL authority to an object.

Grouped Authority	RWX Data Authority	OBJ Mgt	OBJ Exist	OBJ Alter	OBJ Ref	Opr	Read	Add	Update	Delete	Execute
*EXCLUDE											
*USE	*RX					X	X				X
*CHANGE	*RWX					X	X	X	X	X	X
*ALL	*RWX	X	X	X	X	X	X	X	X	X	X

Overall security considerations

System-supplied IFS authorities
The default authorities that ship with OS/400 on the root and QOpenSys are set to *PUBLIC *ALL (*RWX and all object authorities). It is highly recommended that these be changed to more restrictive authorities.

At the minimum, remove *W and all object authorities from *PUBLIC leaving only *RX. This will still allow any user profile to read the contents of the root directory in each file system and to traverse the directory to lower level directories, but will prohibit adding or

© SANS Institute 2000 - 2005 Author retains full rights.

deleting objects under those root directories. Remove this authority for the /home directory also, since it is a supplied directory and has the same settings as the root '/' directory.

An even more restrictive setting, *X, could be used to allow only directory traversal, but this impairs the ability to display the current directory path since each directory must be read to string the path together.

Controlling QSYS.LIB access via the IFS
The QSYS.LIB file system can be accessed using the IFS. There is an authorization list, QPWFSERVER, that can control access to QSYS.LIB through iSeries Navigator, NetServer file sharing (discussed below), and the QFileSrv.400 file system (which allows remote access to another AS/400's IFS). *PUBLIC has *USE authority to this *AUTL, so by default, anyone can access QSYS.LIB via those interfaces. However, it does not restrict access via FTP, ODBC, or DDM (distributed data management) files.

The complete solution to this is to use the integrated object authority discussed in this paper to restrict access to the objects at the object level. Object authority works regardless of access interface.

Authority examples
In general in the IFS, only the owner or a user with *ALLOBJ special authority can set permissions on an object. Just as in Unix, there is no "manage security" attribute as there is in the QSYS.LIB file system (*OBJMGT authority) and in Windows (*Change Permissions* permission).

Adopted authority does not work in the IFS, i.e., an AS/400 program compiled as *OWNER will not adopt authority when it works in the IFS to process files. Other means must be employed such as user profile switching for the job. This is beyond the scope of this paper.
In order for many file-level functions to work in the IFS, the user profile must have *OBJMGT or *OBJEXIST authorities on the objects.

- *OBJEXIST–delete, save, restore
- *OBJMGT–rename, read (but not set) permissions

*W authority on the directory is not sufficient (as in Unix) because OS/400 also checks the *object* for proper authority. A user profile might have *W on a directory, but if it does not have *OBJEXIST on the file, OS/400 will not allow the user to delete the file. This is analogous to a user profile having *DLT data authority on a *LIB object but not having *OBJEXIST on a program or file in the library. Most authority problems occur because these have been removed from a directory or user profile.

Default object authorities
The default authorities on a new object depend on the interface used to make the new object. The following is true regardless of the interface used:

© SANS Institute 2000 - 2005 Author retains full rights.

- The user profile making the new object owns it. Ownership cannot be automatically assigned to the user profile's group profile using the user profile's OWNER(*GRPPRF) setting as can be in QSYS.LIB.

New object–from PC
When a PC makes a new object via a shared directory, the new object inherits some of its attributes from its parent folder, some are assigned to the owner by default, and some can be controlled with the parent folder attributes.

- The owner gets *ALL authority to the new object (This is true as of V5R2. Previously, the owner got the permissions that the *owner of the parent directory had*. This change in default owner authority can eliminate many of the authority problems stemming from restricted authorities for the owner of a directory.)
- *PUBLIC authority is inherited from the directory's *PUBLIC authority
- The object's primary group assignment can be set using iSeries Navigator (starting in V5R2). Open the properties of the directory, select the Security tab, and under "Default primary group" there are two choices, "Use folder primary group" or "Use user value." The latter assigns the user's GRPPRF as the primary group. This mimics the Unix set-GID on a directory.[20]
- Private authorities and the authorization list are inherited from the parent directory.
- The owner of the parent directory is granted the same authorities to the object as it has to the directory (unless it is a system-supplied security officer level user profile, i.e. QSYS and QSECOFR)

New object–from Unix-type APIs
Both OS/400 and Unix-environment programs can use the Unix-type APIs. The data authorities for the new object (owner, primary group, *PUBLIC) are specified on the Unix-type API calls. Since Unix does not know about object authorities, object authorities for all three–owner, primary group, and *PUBLIC–are inherited from the parent directory's authorities for these. Also, since private authorities and authorization lists are unknown in Unix, the APIs provide an "Inherit mode" parameter that, if set, will inherit all private authorities and the authorization list. If not set, neither is inherited

New object–from OS/400 commands
There are three commands that produce new objects: Create Directory (CRTDIR), Copy to Stream File (CPYTOSTMF), and COPY.

The CRTDIR command has parameters that specify how the authorities are granted to the new directory.

- The value of *INDIR will cause the new directory to inherit all authority from the parent directory. All private authorities propagate down, including the owner of the parent directory (unless that owner is QSYS or QSECOFR).
- Specifying authorities in the parameters results in *PUBLIC being granted the authorities specified on the command. Only the new owner and *PUBLIC

[20]SANS, Unix Security, p. 258.

© SANS Institute 2000 - 2005 Author retains full rights.

authorities are granted.
- In any case, the new owner gets *RWX data authorities and inherits the object authorities of the parent directory's owner.

The CPYTOSTMF command will copy a database file from the QSYS.LIB file system into a stream file in the IFS. The authorities on the new object are as follows:
- *PUBLIC is granted *NONE data authority and inherits object authority from the parent directory's *PUBLIC authority
- The owner of the new file is granted *RWX data authority and inherits object authority from the parent directory's owner

The COPY command and the alias CPY produce a new file by copying the data and other object attributes, most notably the authorities. This command is perhaps the most difficult for OS/400-oriented users to grasp because it is more than a simple data copy like the Copy File (CPYF) command. It functions more like a Create Duplicate Object (CRTDUPOBJ) command in that it accesses the object for information and therefore requires *OBJMGT authority to the source object. The authorities on the new object are as follows:
- *PUBLIC inherits both data and object authority from the source object's *PUBLIC authority
- the owner of the new object inherits both data and object authority from the source object's owner authority
- the primary group is inherited from the source object. This requires the new owner (the user profile executing the COPY) to be a member of the group profile specified as the primary group of the source object.

In all of these examples, a very important concept to remember is how *PUBLIC and owner authorities are granted to a new object. In the IFS there are multiple ways for these authorities to be granted. In most cases they differ from how authority is granted in QSYS.LIB. The inherited authority (or lack thereof) for the owner can become troublesome because the parent directory's owner authority determines the authority of the new object's owner. A recommended method to combat this is to use the CHGAUT command immediately after creating a new object to grant the authorities needed for further processing.[21]

NetServer and shared resource security

The NetServer service that can run on an AS/400 provides a Windows Network Neighborhood-compatible SMB file and print service. Windows or Unix machines which can run an SMB client can access file and print shares, and Windows machines can browse for the resources using Windows Explorer.

Shares are configured much like they are in Windows. They must be configured using iSeries Navigator. The share-level permissions are simpler than for a Windows share.

[21] Jeff Parker, "Integrated File System Security–Advanced Topics," COMMON Conference Handout, March 2003, Session 402088.

© SANS Institute 2000 - 2005 Author retains full rights.

The only options are whether the share is “Read-only” or is “Read/Write” access for everyone. Below this, object authority restricts access.

The entire IFS is available to be shared, starting at the root directory (however, that is not recommended). File and printer access is controlled purely by the object authority settings in OS/400.

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

Chapter 3 – Summary

OS/400 provides integrated object-level security that security administrators need to understand to take full advantage of the AS/400's security. Although the authority implementation can become complex, keeping it simple by employing the system-supplied management objects–group profiles and authorization lists–can avoid many run-time surprises and security breaches. Because the AS/400 provides multiple environments, the "native" QSYS library system, a Unix command environment, a Unix run-time application environment, and Windows file sharing, the security administrator needs to understand how these security models work using OS/400 object-level security to take full advantage of the AS/400's capabilities.

Recommendation for further study

The following are suggested for possible future projects and study:

- A case study on converting from an insecure terminal-based menu security system to using object authorities to define user access. There are still many AS/400's that have menu security as their model, even though they have opened up access methods beyond simple 5250 or Telnet terminal access. A case study of this process would be a helpful addition to AS/400 knowledge of system reconfiguration.
- Research project comparing the two security models mentioned in the Delimitation section, "exit point" security and "Application-only Access"
- A case study covering the implementation of one or the other or both of these models

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

Appendix

The "Common user id" myth

One practice that bypasses the benefits of implementing object-level security is that of allowing users to share user profiles and passwords. An example of this is the practice at some AS/400 sites of connecting users to an AS/400 via IBM Client Access using a single shared user profile and password. The user then signs on to an AS/400 5250 terminal session using his own user profile and password.

Although AS400 subsystems have always allowed default user profile names for a remote job startup via the Add Communications Entry command (ADDCMNE), using a default user profile as a method for connecting user desktops seems to have caught on starting in 1995 because of the unfortunate use of the term "Common user id" in the setup documentation for Client Access for Windows '95.[22] "Common user id" meant a user profile name that a user shared between AS/400 systems. Hence, it was *common* for that user among the systems he needed to access. The myth became that it meant a user profile that was common to all the *users* who needed to access an AS/400.

There are other articles from 1995 and 1996 that discuss connecting users via a default user id, but many at least warn of the possible security risk of using the same profile for multiple users,[23] and another IBM document also states that this recommendation is really only appropriate for a server machine that needs to perform a function in an unattended environment.[24]

It is still occasionally touted as a way to avoid multiple logon screens. For example, a January 2005 Search400.Com article makes the recommendation and goes further by recommending a Windows registry key so that the connection is made at startup.[25]

It is distressing to see this type of recommendation still being made. The problem with this method is that every other program or function that accesses the AS/400 uses the initial connection user profile and authorities, not the profile and authorities of the person signing on to the 5250 session. This practice violates security principles

---

[22]IBM Corporation, "Client Access for Windows 95: 802.2 Token-Ring and Ethernet Connectivity Steps," Software Knowledge Base, 1996. April 10, 2005. <http://www-912.ibm.com/s_dir/slkbase.NSF/0/eb270413e2ea2ded862565c2007ca0b8?OpenDocument>

[23]Wayne Madden, "26 C/S Tips & Techniques: Signing On to the Router," iSeries News, December 1995, p.83. (Also available at iSeries Network, December 1, 1995. April 10, 2005. <http://www.iseriesnetwork.com/resources/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=1291&channel=art&PageView=Search>)

[24]IBM Corporation, "CWBSY: New Client Access R312 Sign On API: cwbSY_LogonUser API," Software Knowledge Base, 1996. April 10, 2005. <http://www-912.ibm.com/s_dir/slkbase.NSF/0/71fb41549d30b364862565c2007caf5e?OpenDocument>

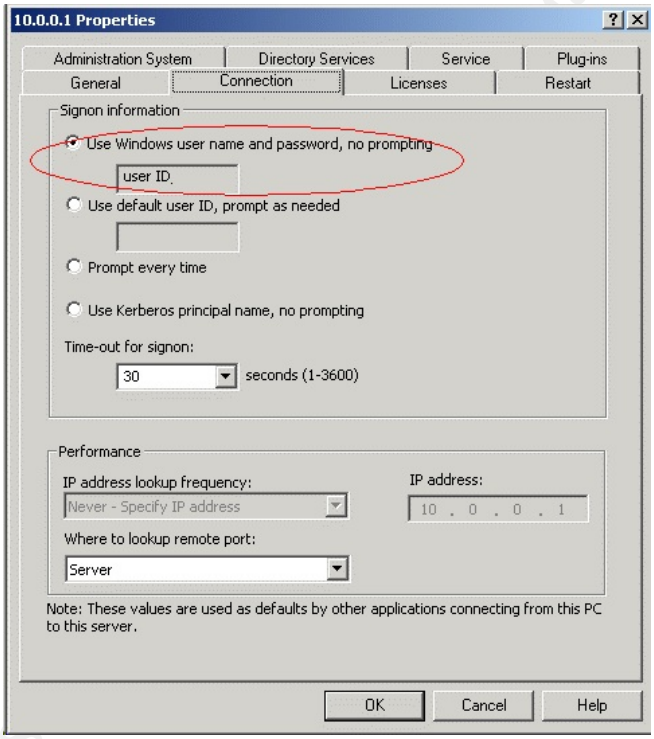
[25]Timothy Grove, "Client Access Express password cache," Search400.Com, January 3, 2005. April 1, 2005. <http://search400.techtarget.com/tip/1,289483,sid3_gci1041437,00.html>

© SANS Institute 2000 - 2005 Author retains full rights.

including least privilege and separation of duties, and it frustrates any auditing because everything occurs under a single user profile.

During a call to IBM Support Line about this issue, an IBM Client Access technician was asked whether IBM had ever recommended this method to connect users to the AS/400. He did not know of an instance where this was recommended as a general connection method. However, he said that this method might be a good choice for a PC server that needs to perform a function in an unattended and secure environment.[26]

Multiple logons can be controlled by the following procedure. Start iSeries Navigator, right click on the system name listed under "My Connections" and select "Properties". Select the "Connections" tab. Under "Signon information" are the various options. For instance, if the Windows and AS/400 user names are synchronized, select the "Use Windows user name and password, no prompting" option.



These methods preserve the access control advantage in establishing object authorities because a user makes his connection to the AS/400 using his user profile and consequently works under that profile's restrictions.

[26]IBM Support Line, technical support, problem number 32006,422, Rochester, MN, April 18, 2005.

© SANS Institute 2000 - 2005 Author retains full rights.

References

Earl, John. "Exit Programs Tighten AS/400 Security." PowerTech.Com. [No date] April 25, 2005. <http://www.400security.com/pt-about_news-art_FA0603.html>

_____. "Purchased Software Can Jeopardize your Security." iSeries News, December 1, 1999. (Also available at iSeries Network, December 1, 1999. April 15, 2005. <http://www.iseriesnetwork.com/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=3255&channel=&subart=>)

Evans, Wayne. "Application-only Access: An AS/400 Resource Security Strategy." Wayne O. Evans Consulting. [No date] March 29, 2005. <http://www.woevans.com/AOA.pdf>

Grove, Timothy. "Client Access Express password cache." Search400.Com. January 3, 2005. April 1, 2005. <http://search400.techtarget.com/tip/1,289483,sid3_gci1041437,00.html>

IBM Corporation. "Client Access for Windows 95: 802.2 Token-Ring and Ethernet Connectivity Steps." Software Knowledge Base, 1996. April 10, 2005. <http://www-912.ibm.com/s_dir/slkbase.NSF/0/eb270413e2ea2ded862565c2007ca0b8?OpenDocument>

_____. "CWBSY: New Client Access R312 Sign On API: cwbSY_LogonUser API." Software Knowledge Base, 1996. April 10, 2005. <http://www-912.ibm.com/s_dir/slkbase.NSF/0/ 71fb41549d30b364862565c2007caf5e?OpenDocument>

_____. "Enterprise Identity Mapping." iSeries Information Center–V5R2. [No date] April 19, 2005. <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzalv/rzalvmst.htm>

_____. "Identifying User Groups." iSeries Information Center–V5R3. [No date], April 15, 2005. <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/index.htm?info/rbapk/rbapkrbapk4i3exampleidug.htm>

_____. iSeries Security Reference–Version 5. 8th ed. May, 2004. <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/info/rbapk/sc415302.pdf>

_____. OS/400 Online Help Information, V5R2.

_____. "OS/400 PASE." iSeries Information Center–V5R3. [No date], May 5, 2005. <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/ic2924/index.htm?info/rzalf/rzalfintro.htm>

_____. "Overview of iSeries (OS/400) Architecture." Virtual Innovation Center for Hardware. [No date] April 15, 2005. <http://www-1.ibm.com/servers/enable/site/porting/iseries/overview/overview.html>

© SANS Institute 2000 - 2005 Author retains full rights.

_____. "Qshell." iSeries Information Center–V5R3. [No date], May 5, 2005. <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzahz/intro.htm>

IBM Support Line. Technical support, problem number 32006,422. Rochester, MN, April 18, 2005.

Kugel, Jake. "Destination: IFS, Part 1." iSeries News, June 1, 2002. (Also available at iSeries Network, June 1 2002. April 29, 2005. <http://www.iseriesnetwork.com/resources/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=14477&channel=art&PageView=Search>)

Madden, Wayne. "26 C/S Tips & Techniques: Signing On to the Router." iSeries News, December 1995. p.83. (Also available at iSeries Network, December 1, 1995. April 10, 2005. <http://www.iseriesnetwork.com/resources/artarchive/index.cfm?fuseaction=viewarticle&CO_ContentID=1291&channel=art&PageView=Search>)

Midrange dot COM Mailing List Archive. "Re: RMTCMD Anomaly." May 19, 2000. April 20, 2005. <http://archive.midrange.com/midrange-l/200005/msg00996.html>

Minasi, Mark. Mastering Windows NT Server 4. Alameda, CA: Sybex Inc., 1999.

Parker, Jeff. "Integrated File System Security–Fundamentals." COMMON Conference Handout, March 2003, Session 440138.

Parker, Jeff. "Integrated File System Security–Advanced Topics." COMMON Conference Handout, March 2003, Session 402088.

Reardon, Marguerite. "Securing data from the threat within." CNET News. January 11, 2005. April 20, 2005. <http://news.com.com/Securing+data+from+the+threat+within/2100-7347_3-5520016.html?tag=st.prev>

SANS Institute. Track 1–SANS Security Essentials. Volume 1.2, Defense In-Depth. SANS Press, September 2004.

_____. Track 1–SANS Security Essentials. Volume 1.5, Windows Security. SANS Press, September 2004.

_____. Track 1–SANS Security Essentials. Volume 1.6, Unix Security. SANS Press, September 2004.

Shapiro, Jeffrey R. and Jim Boyce. Windows 2000 Server Bible. New York: Hungry Minds, Inc., 2000.

Woodbury, Carol. "Security Patrol: Security Considerations for the Integrated File System (IFS)." MCPress Online. April 2003. April 11, 2005. <http://www.mcpressonline.com/mc?50@118.ksLEccBImF2.9@.6ae64331>

© SANS Institute 2000 - 2005 Author retains full rights.