CHAPTER 6

Object-Level Security

Once again, I want to acknowledge the reason why you are reading this book. It is because there is data, either on systems you've been tasked to administer or on systems you are auditing, that needs to be protected. How tightly that data should be protected depends on its classification. Some data will require "deny by default" protection so that only users with a specific job requirement can access it. Other data might be classified such that all users can read the data but not update it. This chapter introduces you to the features available in IBM i for protecting data. Later in the book, I provide some practical examples that use these features to implement security schemes such as deny by default and read-only access.

Before a user can access an object, the user's profile must have authority to the object. Authority can come from several places: an object's *PUBLIC authority, private authority given to the user's individual or group profile, a group profile, an authorization list securing the object, adopted authority, or (as you learned in Chapter 4) *ALLOBJ special authority. In this chapter, I discuss each source of authority, how to manage the various options, and why you might use each one. The examples I'll discuss use the QSYS file system (which uses the traditional library/object structure). Authority considerations for the integrated file system (IFS) are discussed in Chapter 7.

Private Authorities

Authorities granted to individual or group profiles — often called *private authorities* — permit users to access and use specific objects. You can grant (give) or revoke (take away) specific authorities by using one of the following commands:

- GRTOBJAUT (Grant Object Authority)
- RVKOBJAUT (Revoke Object Authority)
- EDTOBJAUT (Edit Object Authority)

As the diagram in Figure 6.1 illustrates, IBM i provides two categories of authorities: object authorities and data authorities. *Object authorities* authorize actions that affect an entire object, such as renaming, changing, saving, and deleting the object, as well as a few functions for specific types of objects, such as defining referential integrity rules for database files. *Data authorities* apply at the content level for objects (e.g., database files, data queues, libraries, output queues) and authorize very specific actions, such as adding, updating, reading, and deleting object contents (e.g., records, entries).

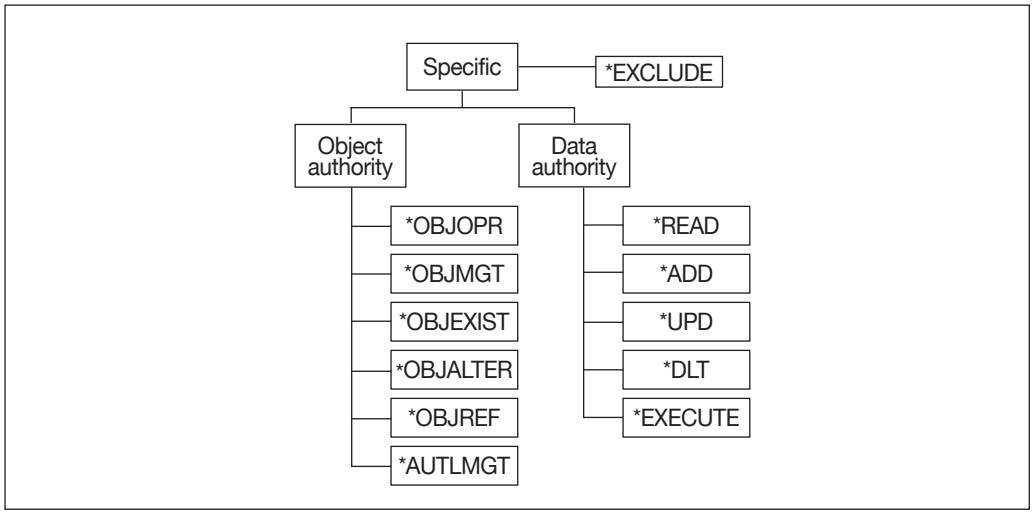

Figure 6.1: Authority categories

Object Authorities

IBM i provides six object authorities, which I'll group into three categories for purposes of explaining their functions. The first category consists of three object authorities that apply to all the objects on your system: *OBJOPR, *OBJMGT, and *OBJEXIST.

Object authority	Description
*OBJOPR	*Object operation* authority authorizes the use of an object.
*OBJMGT	*Object management* authority authorizes the user to move and rename the object, change its attributes, and grant and revoke the authority of other user profiles to the object.
*OBJEXIST	*Object existence* authority lets the user delete, save, and restore the object and transfer ownership of the object.

Users who have *OBJOPR authority to an object can use the object. What exactly does it mean to be able to "use" an object? *OBJOPR authority lets the user view the object's description, regardless of the object's type. If the object contains data or executable code, the user can access the object's description but cannot access the data or run the program without having the appropriate data authority.

*OBJMGT authority lets the user change the object's attributes, including the authorities that other users have to the object. *OBJMGT is a superset of the object authorities *OBJALTER and *OBJREF (which I discuss momentarily). If you don't want a user to have *OBJALTER or *OBJREF authority, you must revoke not only *OBJMGT authority but also the *OBJALTER and *OBJREF authorities specifically.

*OBJEXIST authority lets the user perform save/restore operations on the object, transfer object ownership, and delete the object.

To reduce the risk of an object being improperly used or inadvertently deleted, you should restrict *OBJMGT and *OBJEXIST authorities to the object's owner. Unfortunately, this recommendation can make some operations a challenge, such as when a file is deleted, a physical file member is added or cleared, or a file is re-created when a query is run. In later chapters, I'll discuss techniques for managing these scenarios.

The second category of object authorities consists of those that apply primarily to database files. These authorities are the aforementioned *OBJALTER and *OBJREF authorities.

Object authority	Description
*OBJALTER	*Object alter* authority authorizes the user to add, clear, reorganize, and initialize database file members and to alter the attributes of database files (SQL tables or physical and logical files). Authorized actions include adding and removing triggers and changing the attributes of tables, files, and SQL packages.
*OBJREF	*Object reference* authority lets the user specify a database file or table as the parent file in a referential constraint.

A user with *OBJALTER authority to a database file can change the file's attributes and can add and remove database file triggers. If you want to avoid granting a database administrator or programmer *OBJMGT authority, you can grant only *OBJALTER authority to limit the user to database-specific functions.

You should grant *OBJREF object authority to only the database administrator or the programmer responsible for enforcing referential integrity constraints. This authority is necessary only for the parent file in a referential constraint. For example, say you want to ensure that every new order record in the ORD_MAST file references an existing customer in the CUST_MAST file. You can do so by adding a referential constraint that links the two files by the customer number that is common to both files. In this case, you must have *OBJREF authority to the CUST_MAST file to add the constraint.

The third category of object authorities consists of just one authority, *AUTLMGT (authorization list management), and it applies only to authorization lists. *AUTLMGT

authority lets you add and remove users' authorities to an authorization list. I cover authorization lists in more detail later in this chapter.

Data Authorities

Data authorities apply to all object types, but they are most easily explained by using objects that have contents. For example, a physical file (or table) has an object description, and a physical file has contents: records. Other examples of objects that have contents include libraries (object type *LIB), data queues (object type *DTAQ), data areas (object type *DTAARA), and output queues (object type *OUTQ).

There are five data authorities:

Data authority	Description
*READ	Lets the user read and retrieve data and entries.
*ADD	Lets the user insert a new data record or entry.
*UPD	Lets the user modify existing data and entries.
*DLT	Lets the user delete and remove data and entries. (Remember that *OBJEXIST is required to delete the object itself.)
*EXECUTE	Lets the user locate an object in a library or directory and execute a program, a service program, or an SQL package.

The first four data authorities let you read and manipulate the contents of an object. You would typically grant users who need to manipulate data all four of these data authorities. However, if you prefer, you can grant users only a subset of the four.

*EXECUTE authority lets you search the contents of a library or directory. To access an object, you need not only authority to the object itself but also at least *EXECUTE authority to the library or directory in which the object resides. *EXECUTE authority is also one of the authorities you need to run a program, service program, or SQL package (you also need *OBJOPR authority).

As I mentioned earlier, data authorities also apply to objects that don't have contents. For example, to submit a job that uses a job description, you need *OBJOPR, *READ, and *EXECUTE authorities to that job description.

Authority Relationships

One point of confusion for many people when they first encounter IBM i's object security implementation is the difference between having authority to a library and having authority to objects within the library. To access or use an object, you must have authority both to the library in which the object resides and to the object itself. If you don't have authority to the library, you won't be able to access the objects in the library. However, having authority to a library does not give you authority to the objects within it. You need authority to both in order to access an object in a library.

Think of library-level authority as a gate to what goes on with the objects that reside in the library. To access the objects, you must first open the gate; that is, you must have appropriate authority to the library for the request being made. For example, to look for objects in the library, you must have *OBJOPR authority to the library. To read the description of objects in the library, you must also have *READ authority to the library. To add objects to the library, you must have *ADD authority. And so on.

*DLT authority can also be confusing. You would think that if you had *DLT authority to a library, you could delete objects from the library. However, to delete an object from a library, you need only *USE authority to the library. Whether you can delete an object is determined by the authority you have to the object itself. To delete an object, you must have *OBJEXIST authority to that object. To delete a library, you need *OBJEXIST authority to the library.

Now for a quick preview of the next chapter: If you work with directories on the system — that is, if you work with file systems other than the QSYS file system (i.e., libraries) — you need to know that the relationship between directory and object authorities is nearly identical to the relationship I just described between library and object authorities. The only difference is that to access an object in the file system, you must have *EXECUTE authority to all directories in the directory path. For example, to access the file SkyView_Partners/Policy_Minder/Reports/Compliance_report.pdf, you must have authority to every directory in the path that leads to the file (i.e., to the SkyView_Partners, Policy_Minder, and Reports directories) as well as proper authority to the Compliance_report.pdf file itself.

Authority Groupings

To make object and data authorities easier to manage, IBM i defines several authority groupings. You can think of these groupings as shorthand: They let you specify commonly used combinations of object and data authorities using a single value. Except when working with database files, most security administrators use one of these authority groups rather than individual object and data authorities. Table 6.1 lists the authority groupings you can use with libraries and their objects.

Table 6.1: Authority groupings for use with authority commands for libraries and object

	Authority classes			
Object and data authorities	*ALL	*CHANGE	*USE	*EXCLUDE
*OBJOPR	X	X	X	None
*OBJMGT	X			
*OBJEXIST	X			
*OBJALTER	X			

continued

Table 6.1: Authority groupings for use with authority commands for libraries and object (continued)

	Authority classes			
Object and data authorities	*ALL	*CHANGE	*USE	*EXCLUDE
*OBJREF	X			
*READ	X	X	X	
*ADD	X	X		
*UPD	X	X		
*DLT	X	X		
*EXECUTE	X	X	X	

To use authority groupings to grant user profile CYOUNG object operational and all data rights to physical file ACCTPAY_PF in the QSYS file system, you could use this GRTOBJAUT command:

```
GRTOBJAUT  OBJ(MYLIB/ACCTPAY_PF) OBJTYPE(*FILE) USER(CYOUNG) +
           AUT(*OBJOPR *READ *ADD *UPD *DLT *EXECUTE)
```

Or, you could use

```
GRTOBJAUT  OBJ(MYLIB/ACCTPAY_PF) OBJTYPE(*FILE) USER(CYOUNG) +
           AUT(*CHANGE)
```

Both commands have the same effect.

*EXCLUDE authority is a bit odd in that users who have this authority are actually prevented from accessing the object rather than being given some type of access rights. Some operating systems, such as Unix, prevent users from accessing a file simply by not granting them any authority to the file. In other words, the absence of authority prevents the users from accessing the file. IBM i works differently. With IBM i, users always have some authority to the object. If they don't have authority through another means, they will, by default, have authority to the object's *PUBLIC authority. (I explain *PUBLIC authority a bit later in this chapter.) To prevent a user from accessing an object, you must grant that user *EXCLUDE authority to the object. To set the object's access to "deny by default," you would set the object's *PUBLIC authority to *EXCLUDE.

Group Profiles

Not only can you grant or revoke private authorities to or from individual users, but you can do the same for group profiles. Group profiles, as you learned in Chapter 4, are a management tool that provides an efficient way to allow all the members of a group (or role) to have the same access to an object. You use the same commands to grant or revoke authority to group profiles. Simply specify the name of the group profile instead of an individual user profile name on the commands.

Multiple Group Profiles

Recall from Chapter 4 that you can specify a user profile as a member of up to 16 groups. To do so, you first specify a group in the user profile's GRPPRF (Group profile) parameter. Then you use the SUPGRPPRF (Supplemental groups) parameter to specify any other group profiles you want the user to be a member of.

When a member of multiple groups creates an object, IBM i refers to the user profile's OWNER (Owner), GRPAUT (Group authority), and GRPAUTTYP (Group authority type) parameters to determine the authority, if any, that the first group (i.e., the group profile named in the GRPPRF parameter) has to the new object. When you assign multiple groups to users, you should assign the most frequently used groups first. When the operating system determines whether a user has sufficient authority to access an object, it searches each group profile listed in the user profile's GRPPRF and SUPGRPPRF parameters until it finds enough authority to perform the task (or determines that the user lacks sufficient authority). The fewer groups the system has to search to accumulate sufficient authority, the better the performance.

Why Grant Authority to Group Profiles?

Group profiles simplify security management by letting you grant similar authorities to multiple user profiles that perform similar functions (i.e., to a role). Many user profiles in your organization probably have similar basic authorities. Combining similar user profiles into a single group profile lets you administer authorities for the entire group instead of for each user individually. Because many users — especially in IT — have many roles, the ability to make a user profile a member of more than one group is particularly helpful.

When you grant authority to a group profile, an option you have is to use *primary group authority* rather than a private authority. The effect of granting primary group authority is the same as granting a private authority to the group. Only a couple of differences exist between the two authorities. One difference is that primary group authority is stored with the object itself, while private authorities are stored in the user's user profile object. This point has two ramifications. First, primary group authority goes with the object when it is saved and restored. To associate a private authority with an object, you must either save the object by running the SAVSECDTA (Save Security Data) command and then running the RSTAUT (Restore Authority) command after the object is restored or you must specify to save private authorities when you save the object and specify to restore the private authorities when you restore the object. (The ability to save/restore private authorities with the object was added in V6R1.) The second ramification is that retrieving the authority from the object's header is ever so slightly faster than retrieving the private authority from the user's profile. You can grant primary group authority for an object to only one group. Rather than give the group profile private authority to the object, you can grant the group primary group authority by using the CHGOBJPGP (Change Object Primary Group) command or the WRKOBJPGP (Work with Objects by Primary Group) command.

From a performance standpoint, primary group authority is a *slightly* faster version of private authority for group profiles. Primary group authority isn't considered a private authority. Rather, it is stored with the object itself, thus shortening the time the system needs to find the proper authorities during an authorization search. However, the performance gain is negligible, so if you find the concept of primary group authority too confusing, go ahead and grant the group private authority if it makes management easier.

Note

I have seen only one IBM i site that used primary group authority to grant groups authority to libraries and files. I have encountered a few more instances in which primary group authority was granted to files and directories in the IFS. The bottom line is this: Unless this concept makes your security administration easier, you shouldn't bother with implementing (or even attempting to understand) primary group authority.

Public Authority

The *PUBLIC authority of an object is assigned when a user creates the object. This is the authority users have to an object if they don't obtain authority from any other source. To view an object's authorities, including *PUBLIC authority, you can use the EDTOBJAUT command or the DSPOBJAUT (Display Object Authority) command.

Establishing Public Authority

All objects on the system are created either by using one of the many create (CRTxxx) commands or by restoring the object from a save file or save media. The initial public authority for an object is determined when the object is created. (You can always change it later if the original setting isn't appropriate.) When you use a create command to create an object, the command's AUT (Authority) parameter establishes the object's public authorities. The AUT parameter has the following possible values:

AUT value	Description
*LIBCRTAUT	Uses the library's CRTAUT (Create authority) attribute to determine the object's *PUBLIC authorities.
*ALL	Grants all object and data rights.
*CHANGE	Grants *OBJOPR object authority and all data authorities.
*USE	Grants *OBJOPR object authority and *READ and *EXECUTE data authorities.
*EXCLUDE	Grants *PUBLIC *EXCLUDE authority.
Authorization_list_name	Secures the object with an authorization list. *PUBLIC authority for the object comes from the authorization list's *PUBLIC authority. (You can't specify an authorization list for the AUT parameter of a user profile or of another authorization list.)

For most create commands, the default value for the AUT parameter is *LIBCRTAUT, which causes the system to use the object library's CRTAUT (Create authority) attribute to determine the object's public authority. Thus, the CRTAUT library attribute specifies the default public authority that all new objects in the library will have unless you override the default authority on the create command.

You can use the library's CRTAUT parameter to define the default public authority for all objects subsequently created in that library (existing objects aren't affected). The CRTAUT parameter allows the following values:

CRTAUT value	Description
*SYSVAL	Uses the value specified in the QCRTAUT (Create Authority) system value
*ALL	Grants all object and data rights
*CHANGE	Grants *OBJOPR and all data rights
*USE	Grants *OBJOPR, *READ, and *EXECUTE rights
*EXCLUDE	Grants *PUBLIC *EXCLUDE authority
Authorization_list_name	Uses the specified authorization list to determine public authorities

The default value for the CRTAUT parameter is *SYSVAL, which instructs the operating system to reference the QCRTAUT system value to determine the default public authorities for the objects in the library. When you create a library and don't specify a CRTAUT value, the *SYSVAL default value causes the library to use the value found in the QCRTAUT system value.

Technical Note

System value QCRTAUT has no effect on objects created in directories.

Technical Note

Changing the QCRTAUT system value has no effect on objects that already exist. A change to QCRTAUT affects only objects that are created after the system value is changed.

Figure 6.2 illustrates the trickle effect of the default values for the library's CRTAUT attribute and the object's AUT attribute. From bottom to top, the values shown for AUT, CRTAUT, and QCRTAUT are the defaults shipped with IBM i. As you can see, if you don't change these defaults, the system automatically gives the public *CHANGE authority to almost every object created on the system.

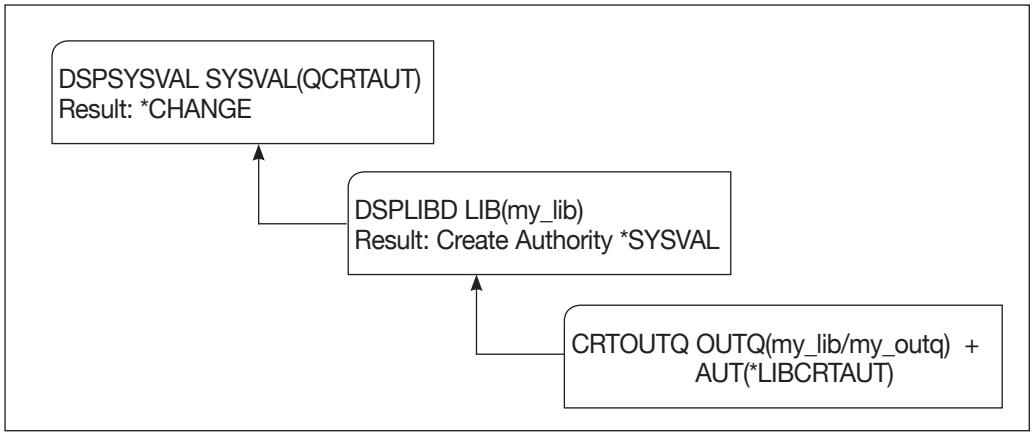

*Figure 6.2: How *PUBLIC authority is determined*

The only tricky part of how *PUBLIC authority is determined comes when you create a library. As implemented, IBM i actually creates all libraries in library QSYS. You can think of QSYS as the "master" library. So, when you create a new library — say, PAYLIB — and specify *LIBCRTAUT for the library's AUT parameter, PAYLIB's public authority comes from the QSYS library's CRTAUT value.

Technical Note

A library's CRTAUT parameter has no effect when you move, duplicate, or restore an object to a library. Instead, the object's public authority is the public authority that the object had before you moved, duplicated, or restored it.

Technical Note

When you create an object and specify REPLACE(*YES), the new object will have the public authority of the replaced object instead of the authority specified in the library's CRTAUT parameter.

Using Default Public Authority

Because IBM ships the QCRTAUT system value set to *CHANGE, the default public authority for most objects created in any library is *CHANGE. In today's environment, where users have access to tools such as File Transfer Protocol (FTP), Open Database Connectivity (ODBC), and SQL and where applications can use sockets or the HTTP server to connect to the system, public authority *CHANGE isn't appropriate — espe-