2.2. Access Control Matrix Model

The simplest framework for describing a protection system is the **access control matrix model**, which describes the rights of users over files in a matrix. Butler Lampson first proposed this model in 1971 [608]; Graham and Denning [279, 413] refined it, and we will use their version.

The set of all protected entities (that is, entities that are relevant to the protection state of the system) is called the set of **objects O**. The set of **subjects S** is the set of active objects, such as processes and users. In the access control matrix model, the relationship between these entities is captured by a matrix **A** with **rights** drawn from a set of rights **R** in each entry **a**[**s**, **o**], where **s** ∈ **S**, **o** ∈ **O**, and **a**[**s**, **o**] ⊆ **R**. The subject **s** has the set of rights **a**[**s**, **o**] over the object **o**. The set of protection **states** of the system is represented by the triple (**S**, **O**, **A**). For example, Figure 2-1 shows the protection state of a system. Here, process 1 can read or write file 1 and can read file 2; process 2 can append to file 1 and read file 2. Process 1 can communicate with process 2 by writing to it, and process 2 can read from process 1. Each process owns itself and the file with the same number. Note that the processes themselves are treated as both subjects (rows) and objects (columns). This enables a process to be the target of operations as well as the operator.

Figure 2-1. An access control matrix. The system has two processes and two files. The set of rights is {read, write, execute, append, own}.

	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

Interpretation of the meaning of these rights varies from system to system. Reading from, writing to, and appending to files is usually clear enough, but what does "reading from" a process mean? Depending on the instantiation of the model, it could mean that the reader accepts messages from the process being read, or it could mean that the reader simply looks at the state of the process being read (as a debugger does, for example). The meaning of the right may vary depending on the object involved. The point is that the access control matrix model is an **abstract** model of the protection state, and when one talks about the meaning of some particular access control matrix, one must always talk with respect to a particular implementation or system.

The **own** right is a distinguished right. In most systems, the creator of an object has special privileges: the ability to add and delete rights for other users (and for the owner). In the system shown in Figure 2-1, for example, process 1 could alter the contents of **A**[**x**, file 1], where **x** is any subject.

EXAMPLE: The UNIX system defines the rights "read," "write," and "execute." When a process accesses a file, these terms mean what one would expect. When a process accesses a directory, "read" means to be able to list the contents of the directory; "write" means to be able to create, rename, or delete files or subdirectories in that directory; and "execute" means to be able to access files or subdirectories in that directory. When a process accesses another process, "read" means to be able to receive signals, "write" means to be able to send signals, and "execute" means to be able to execute the process as a subprocess.

Moreover, the superuser can access any (local) file regardless of the permissions the owner has granted. In effect, the superuser "owns" all objects on the system. Even in this case however, the interpretation of the rights is constrained. For example, the superuser cannot alter a directory using the system calls and commands that alter files. The superuser must use specific system calls and commands to create, rename, and delete files.

Although the "objects" involved in the access control matrix are normally thought of as files, devices, and processes, they could just as easily be messages sent between processes, or indeed systems themselves. Figure 2-2 shows an example access control matrix for three systems on a local area network (LAN). The rights correspond to various network protocols: **own** (the ability to add servers), **ftp** (the ability to access the system using the File Transfer Protocol, or FTP [815]), **nfs** (the ability to access file systems using the Network File System, or NFS, protocol [166, 981]), and **mail** (the ability to send and receive mail using the Simple Mail Transfer Protocol, or SMTP [814]). The subject **telegraph** is a personal computer with an **ftp** client but no servers, so neither of the other systems can access it, but it can **ftp** to them. The subject **nob** is configured to provide NFS service to a set of clients that does not include the host toadflax, and both systems will exchange mail with any host and allow any host to use **ftp**.

Figure 2-2. Rights on a LAN. The set of rights is {ftp, mail, nfs, own}.

host names	*telegraph*	*nob*	*toadflax*
telegraph	own	ftp	ftp
nob		ftp, nfs, mail, own	ftp, nfs, mail
toadflax		ftp, mail	ftp, nfs, mail, own

At the micro level, access control matrices can model programming language accesses; in this case, the objects are the variables and the subjects are the procedures (or modules). Consider a program in which events must be synchronized. A module provides functions for incrementing (**inc_ctr**) and decrementing (**dec_ctr**) a counter private to that module. The routine **manager** calls these functions. The access control matrix is shown in Figure 2-3 Note that "+" and "–" are the rights, representing the ability to add and subtract, respectively, and **call** is the ability to invoke a procedure. The routine **manager** can call itself; presumably, it is recursive.

Figure 2-3. Rights in a program. The set of rights is {+, –, call}.

	counter	*inc_ctr*	*dec_ctr*	*manager*
inc_ctr	+			
dec_ctr	–			
manager		call	call	call

In the examples above, entries in the access control matrix are rights. However, they could as easily have been functions that determined the set of rights at any particular state based on other data, such as a history of prior accesses, the time of day, the rights another subject has over the object, and so forth. A common form of such a function is a locking function used to enforce the Bernstein conditions,[2] so when a process is writing to a file, other processes cannot access the file; but once the writing is done, the processes can access the file once again.

[2] The Bernstein conditions ensure that data is consistent. They state that any number of readers may access a datum simultaneously, but if a writer is accessing the datum, no other writers or any reader can access the datum until the current writing is complete [805].

2.2.1. Access Control by Boolean Expression Evaluation

Miller and Baldwin [715] use an access control matrix to control access to fields in a database. The values are determined by Boolean expressions. Their objects are records and fields; the subjects are users authorized to access the databases. Types of access are defined by the database and are called **verbs**; for example, the Structured Query Language (SQL) would have the verbs Insert and Update. Each **rule**, corresponding to a function, is associated with one or more verbs. Whenever a subject attempts to access an object using a right (verb) **r**, the Boolean expression (rule) associated with **r** is evaluated; if it is true, access is allowed, but if it is false, access is not allowed.

The Access Restriction Facility (ARF) program exemplifies this approach. It defines subjects as having attributes such as a name, a level, a role, membership in groups, and access to programs, but the user can assign any meaning desired to any attribute. For example:

name	role	groups	programs
matt	programmer	sys, hack	compilers, editors
holly	artist	user, creative	editors, paint, draw
heidi	chef, gardener	acct, creative	editors, kitchen

Verbs have a default rule, either “closed” (access denied unless explicitly granted; represented by the 0 rule) or “open” (access granted unless explicitly denied; represented by the 1 rule):

verb	default rule

verb	default rule
read	1
write	0
paint	0
temp_ctl	0

Associated with each object is a set of verbs, and each (object, verb) pair has an associated rule:

name	rules
recipes	write: 'creative' in subject.group
overpass	write: 'artist' in subject.role or 'gardener' in subject.role
.shellrct	write: 'hack' in subject.group and time.hour < 4 and time.hour > 0
oven.dev	read: 0; temp_ctl: 'kitchen' in subject.programs and 'chef' in subject.role

The system also provides primitives such as **time** (which corresponds to the current time of day), **date** (the current date), and **temp** (the current temperature). This generates the following access control matrix between midnight and 4 A.M.:

	matt	holly	heidi
recipes	read	read, write	read, write
overpass	read	read, write	read, write
.shellrct	read, write	read	read
oven.dev			temp_ctl

At other times, the entry **A**[**matt**, **.shellrct**] contains only **read**. The **read** rights in the last row are omitted because, even though the default in general is to allow read access, the default rule for the object **oven.dev** is to deny **read** access:

	matt	holly	heidi
recipes	read	read, write	read, write
overpass	read	read, write	read, write
.shellrct	read	read	read

	matt	holly	heidi
oven.dev		temp_ctl	

2.2.2. Access Controlled by History

Statistical databases are designed to answer queries about groups of records yet not reveal information about any single specific record. Assume that the database contains **N** records. Users query the database about sets of records **C**; this set is the **query set**. The goal of attackers is to obtain a statistic for an individual record. The **query-set-overlap control** [304] is a prevention mechanism that answers queries only when the size of the intersection of the query set and each previous query set is smaller than some parameter **r**.

EXAMPLE: Consider the database below and set **r** = 2.

name	position	age	salary
Celia	teacher	45	$40,000
Heidi	aide	20	$20,000
Holly	principal	37	$60,000
Leonard	teacher	50	$50,000
Matt	teacher	33	$50,000

Let the first query set be $\mathbf{C}_1$ = "position is teacher"; the query **sum_salary**($\mathbf{C}_1$) will be answered. If the next query is $\mathbf{C}_2$ = "age less than 40 and teacher" then the query **count**($\mathbf{C}_2$) will also be answered. But if the query set is $\mathbf{C}_3$ = "age greater than 40 and teacher", the response to the query **sum_salary**($\mathbf{C}_3$) would allow one to deduce the salary of an individual. But that query would not be answered because the set has two elements in common with $\mathbf{C}_1$ (Celia and Leonard).

This control can be modeled using an access control matrix. The subjects are the entities querying the database, and the objects are the elements of the power set[3] of records. Let $\mathbf{O_i}$ be the union of the objects referenced in accesses 1 through **i**, inclusive. The function **f**($\mathbf{O_i}$) is **read** if $|\mathbf{O_i}| > 1$ and is Ø otherwise. Thus, the **read** right corresponds to the database being able to answer the query **sum_salary** for that particular query set. The elements of the matrix corresponding to this control are **A**[**s**, **o**] = **f**($\mathbf{O_{i+1}} \cap \{ \mathbf{o} \}$) for query **i**, where $1 \leq \mathbf{i}$ and $\mathbf{O}_0 = \emptyset$.

When $\mathbf{C}_1$ is referenced, $|\mathbf{O}_1| = 3$ and **A**[**asker**, **(Celia**,**Leonard**,**Matt)**] = **read**, so the query is answered. When $\mathbf{C}_2$ is used, $|\mathbf{O}_2| = 2$ and **A**[**asker**, **(Celia**,**Leonard)**] = **read**, so that query is also answered. But with query $\mathbf{C}_3$, $|\mathbf{O}_3| = 1$, so **A**[**asker**, **(Holly**,**Leonard**,**Matt)**] = Ø, and the query is not answered.

[3] If **S** is a set, then its power set **P**(**S**) is the set of all subsets of **S**. That is, **P**(**S**) = {**x** | **x** ⊆ **S**}.

In practice, this control would not be used for several reasons, including the need to remember all prior queries. However, it affords an excellent example of an access control matrix whose entries depend on previous accesses to objects.