Access Control: The Matrix Model

by

Amit Konar

Dept. of Math and CS, UMSL

What is “Protection state” of a system?

The **state** of a system is the collection of the current values of all memory locations, all secondary storage, and all registers and all other components of the system.

The subset of this collection that deals with protection is the **protection state** of the system.

Security Policy

Let

P be the **possible states** of a system,

Q (subset of P) be the set of states in which the system is **authorized to reside**,

The **security policy** attempts to keep the system states as elements of **Q**. It **prevents** the system **from entering a state in P - Q.**

Secure and Precise Security Mechanism

Suppose a security mechanism restricts the states of the system in R($\subseteq$ P).

If **R $\subseteq$ Q**, the **security mechanism** is **secure.**

If **R = Q**, the **security mechanism** is **precise**.

The **access control matrix model** is the most precise model used to describe protection states.

What is access control matrix model?

The **access control matrix model** describes **the rights of the users over files in a matrix.**

Let

S be the set of subjects (processes/users)

O be the set of objects, and

a[s, o], the elements of matrix A$\subseteq$ R, the set of rights, i.e., subject s has the right a[s,o] over object o.

Example: access control matrix

$$\begin{array}{c|cccc|} & \text{File 1} & \text{File2} & \text{Proc.1} & \text{Proc. 2} \\ \text{Proc.1} & \text{r/w/own} & \text{r} & \text{r/w/exe/own} & \text{w} \\ \text{Proc.2} & \text{append} & \text{r/own} & \text{r} & \text{r/w/exe/own} \end{array}$$

Proc.1 can communicate with proc.2 by writing on to it, and Proc. 2 can read from Proc. 1.

What does reading from a process mean?

Depending on the instantiation of the model,

It could mean that:

the reader accepts message from the process being read, or

the reader simply looks at the state of the process being read (as debugger does, for example).

Example : The UNIX System

The UNIX system defines the rights: r/w/exe.

1. When a process accesses a directory, “read” means to be able to create, rename or delete files or subdirectories in that directory, and exe means to be able to access files/sub-directories in that directory.
2. When a process accesses another process, “read” means to be able to receive signals, “write” means to be able to send signals and exe means to be able to exe. The process as a sub-process.

Example: Access Control Matrix for a LAN system

The rights on a LAN:

Own: the ability to add servers

ftp: the ability to access the system using FTP

nfs: the ability to access the system using the Networks File System

mail: the ability to send and receive mail using the Simple Mail Transfer

Example: Rights on a LAN

$$\begin{array}{llll}
\textbf{Host name} & \text{telegraph} & \text{nob} & \text{toadflax} \\
\text{telegraph} & \text{own} & \text{ftp} & \text{ftp} \\
\text{nob} & & \text{ftp/nfs/mail/own} & \text{ftp/nfs/mail} \\
\text{toadflax} & & \text{ftp/mail} & \text{ftp/nfs/mail/own}
\end{array}$$

The subject telegraph is a PC with an ftp client but no servers. So, neither of the other systems can access it, but it can ftp to them.

Modeling Programming Language Accesses Using Access Control Ma

Objects: variables; counter

Subjects: Procedu./Modules;Inc_ctr, Dec_ctr, Ma

Rights: {+, -, call};

$$\begin{array}{l|cccc|} & \text{Counter} & \text{Inc\_ctr} & \text{Dec\_ctr} & \text{manager} \\ \hline \text{Inc\_ctr} & \mathbf{+} & & & \\ \text{Dec\_ctr} & \mathbf{-} & & & \\ \text{Manager} & & \textbf{call} & \textbf{call} & \textbf{call} \\ \hline \end{array}$$

State Transition by operation on System States

Let

the initial state be $X_0 = (S_0, O_0, A_0)$.

Then

$$X_i \underset{op_{i+1}}{\vdash} X_{i+1},$$

which means that system state X_i has a transition to X_{i+1} due to use of operator op_{i+1}.

State transition by commands

Let

C_{i+1} be a command, which is used to change the states of the system from X_i to X_{i+1}.

$p_{i+1,\,1}; \ldots\ldots p_{i+1,m}$ are parameters of C_{i+1}.

Then

$$X_i \vdash \qquad\qquad X_{i+1}.$$

$$c_{i+1}\ (p_{i+1,1};\ \ldots\ldots;p_{i+1,m})$$

Primitive Commands

1. **create subject s;** no rights added
2. **create object o;** no rights added
3. **enter r into a[s, o];** adds right r
4. **delete r from a[s, o];** deletes right r
5. **destroy subject s;** deletes row/columns for s
6. **destroy object o;** destroys rows and columns for o.

Generating Complex commands using primitive commands

Example: In UNIX system, process p created a file f with owner read and write permission. The command capturing resulting changes in access control matrix would be:

```
Command create.file (p, f)
    create object f;
    create own into a[p, f];
    enter r into a[p, f];
    enter w into a[p, f]; end
```

Command indicating that Process p wishes to create a new process q

```
Command spawn.process (p, q)
  create subject q;
  enter own into a [p, q];
  enter r into a[p, q];
  enter w into a[p, q];
  enter r into a[q, p];
  enter w into a[q, p];
  end
```

Example: Mono-operational command

Mono-operational means having a single primitive command in the complex command.

Suppose we want to add p as the owner of file f. The old owner remained.

```
command make.owner(p, f)
  enter own into a[p, f];
end
```

Conditional commands

If p is the owner of a file f, he/she can allow someone q to read the file. The following command does this.

```
Command grant.read.file.1(p, f, q)
   if own in a[p, f]
   then enter r into a[q, f];
 end
```

Inclusion of Boolean and in a command

Suppose that a system has the distinguished right c. If a subject p has the rights c and r over an object, it may give r-right to q.

```
command grant.read.file.2(p, f, q)
 if r in a[p, f] and c in a[p, f]
 then enter r into a[q,f];
end
```

Boolean Or, Negation not allowed

1. **If** r **not in** a[p, f]: **wrong use**

2. **If** own **in** a[p, f] **or** a **in** a[p, f]
   then enter r **into** a[q, f]; **wrong use**