Essay 9

Rule-Set Modeling of a Trusted Computer System

Leonard J. LaPadula

This essay describes a new approach to formal modeling of a trusted computer system. A finite-state machine models the access operations of the trusted computer system while a separate rule set expresses the system's trust policies. A powerful feature of this approach is its ability to fit several widely differing trust policies easily within the same model. We will show how this approach to modeling relates to general ideas of access control, as you might expect. We will also relate this approach to the implementation of real systems by connecting the rule set of the model to the system operations of a Unix System V system. The trust policies we demonstrate in the rule set of the model include the mandatory access control (MAC) and discretionary access control (DAC), to increase the availability of diverse, assured security policies.

- Make it feasible to configure a system with security policies chosen from a vendor-provided set of options with confidence that the resulting system's security policy makes sense and will be properly enforced.
- Construct the model in a manner that allows one to show that it satisfies an accepted definition of each security policy it represents.

The remainder of this essay has three parts:

- First we discuss the Generalized Framework for Access Control view of a trusted system. GFAC motivates the approach we have taken to modeling.
- Next we describe the modeling approach.

- Finally we illustrate elements[1] of the state-machine model and the rule-set model — the two components of the complete model.

Overview of the Generalized Framework for Access Control

The Generalized Framework for Access Control thesis asserts that all access control is based on a small set of fundamental concepts [ABRA90]. Borrowing some of its terminology and concepts from the ISO "Working Draft on Access Control Framework" [ISO90], GFAC starts with the premise that all access control policies can be viewed as rules expressed in terms of attributes by authorities. The three main elements of access control in a trusted computer system are:

Authority: An authorized agent must define security policy, identify relevant security information, and assign values to certain attributes of controlled resources.

Attributes: Attributes describe characteristics or properties of subjects and objects. The computer system will base its decisions about access control on the attributes of the subjects and objects it controls. Examples of attributes are:

security classification
type of object
domain of process
date and time of last modification
owner identification

Rules: A set of formalized expressions defines the relationships among attributes and other security information for access control decisions in the computer system, reflecting the security policies defined by authority.

The generalized framework explicitly recognizes two parts of access control — adjudication and enforcement. We use the term access control decision facility (ADF) to denote the agent that adjudicates access control requests, and the term access control enforcement facility (AEF) for the agent that enforces the ADF's de-

[1] The reader will find additional analysis and a complete policy model in my report [LAPA91].

cisions. In a trusted computer system, the AEF corresponds to the system functions of the trusted computing base (TCB) and the ADF corresponds to the access control rules that embody the system's security policy, also part of the TCB. Figure 1 depicts the generalized framework in the terms just described.

Figure 1. Overview of the Generalized Framework for Access Control.

Formal modeling approach

Background. The GFAC goals translate into these objectives for our formal model:

- Develop a modeling technology in which it is easy to express various policies besides traditional MAC and DAC.
- Fashion the modeling technology to enable the selection of a desired set of security policies from some preevaluated set without having to reevaluate the resulting collection of policies.

- Provide for showing that a model satisfies an accepted definition

formal methods closer to the final stages of implementation — complete functional design and coding.

Appendix A: Model language and constructs

Language for expressing rules. The method for expressing the model's rules departs from the traditional use of mathematical notation. A mixture of programming language statements and limited mathematical notation creates a specification language that is intuitively understandable to a broad audience.

Both rules of operation and rules of the rule set are defined in a language that looks like a programming language. Two basic language constructs are used to organize statements and show their interrelationships: **SELECT CASE** and **IF THEN ELSE**.

The **SELECT CASE** statement has the following syntax:

```
SELECT CASE attribute
  CASE attribute-value1
    statement-block-1
  CASE attribute-value2
    statement-block-2

  .
  .
  .

  CASE ELSE
    statement-block-n
END SELECT
```

A statement-block is one or more statements. Individual statements are terminated by a semicolon. The value of the **SELECT CASE** statement is the value of the statement-block following the **CASE** identified by the current value of the selected attribute. For example, the next **SELECT CASE** has the value of statement-block-2 when the "amount" is $200:

```
SELECT CASE amount
  CASE $100
    statement-block-1
  CASE $200
    statement-block-2
  CASE ELSE
```

statement-block-n
END SELECT

If the current value of the selected attribute is not identified by one of the **CASE**s given, then the value of the **SELECT CASE** statement is the value of the **CASE ELSE** statement-block.

A final word on the **SELECT CASE** statement. The **END SELECT** part of the statement will be omitted when no ambiguity results — the use of indentation will make clear the scope of a **SELECT CASE**.

The **IF THEN ELSE** statement has the following syntax:

IF
Boolean-expression
THEN
statement-block
ELSE
statement-block

The **IF THEN ELSE** statement has its usual meaning. A Boolean expression is an expression consisting of attributes and relational or logical operations and having a value of **TRUE** or **FALSE**.

A **FOR-EACH** statement is also useful. Its syntax is

FOR-EACH process:
statement-block
END-FOR-EACH

Because attributes may apply to more than one kind of entity, the language clarifies an ambiguous reference to an attribute by qualifying each attribute with the name of the entity the attribute belongs to. For example, the attribute "security-level" applies to processes and several kinds of objects. "security-level(process)" refers to the security level of the process.

Rules of operation use the form "[* . . . *]" to identify a system operation. For example, the Open rule uses the statement [* truncate the file *] to stand for the Unix operation that deletes the data in a file. Rules may use the form "(* . . . *)" to enclose a comment, such as (* the directory search was valid and the file exists *) appearing in the Open rule.

Boolean expressions and all statements except the **SELECT CASE** and the **IF THEN ELSE** end with a semicolon. Boolean expressions use the usual inequality operators "<" and ">" and use "==" for expressing equality. Logical operators such as **AND** and **OR** are used in obvious ways.

Rules use the specifications "set-attribute" and "set-attributes" to manage the values of attributes. The rules of the rule-set model use

"set-attribute" to designate the value that an attribute should have if the current request is granted. The syntax for this use is

set-attribute(attribute_name, attribute_value)

The rules of the state-machine model use "set-attributes" to indicate that they are carrying out the set-attribute specifications given by the rules of the rule-set model. Suppose, for example, the state-machine model invokes the rule-set model with a create-file request. Suppose that the rules of the rule-set model approve the request and give two set-attribute specifications:

set-attribute(security-level(file), **SECRET**)
set-attribute(object-category(file), **general**)

Then, the portion of the create rule that carries out the create request will include a set-attribute statement. The meaning of the statement is that the security-level of the file is set to the value **SECRET** and the object-category of the file is set to the value **general**.

Constructs of the state-machine model

Types. A type is a class that is defined by the common attributes possessed by all its members. The name of each type suggests a useful interpretation for the class. The model uses the following types:

request: {alias, alter, change-owner, change-role, clone, create, delete, delete-data, execute, get-permissions-data, get-status-data, modify-access-data, modify-attribute, modify-permissions-data, read, read-attribute, read&write-open, read-open, search, send-signal, terminate, trace, write, write-open}
process
file
directory
ipc
scd
signal
object: [a file, directory, ipc, or scd]
phase: {"active," "unused," "inaccessible"}
flag: {ON, OFF}
mode: {"read," "write," "read&write"}

Variables. A variable is an alterable entity. The variables of the state-machine model define the system states. We can think of variables as functions whose domains are types. Just as naturally, we can regard them as records of information containing one or more items of data. The model uses the following variables:

current_process: process
new_process: process
file_name: file
directory_name: directory
truncate_option: flag
create_option: flag
STATUS(object): phase
OPEN(process, object): set(mode)

Constants

TRUE
FALSE
ON
OFF

Expressions

Access-Rules(request, process/object, process/object):
Extended-Boolean

Effects. An effect is an action of the state machine. The model uses the following effects:

normal-exit
error-exit
set-attributes
save
restore

Appendix B: Summary of the Clark-Wilson integrity model

Certification Rule 1: All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

Certification Rule 2: All TPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate,

the security officer must specify a "relation" which defines that execution. A relation is thus of the form: (TPi, (CDIa, CDIb, CDIc, ...)), where the list of CDIs defines a particular set of arguments for which the TP has been certified.

Enforcement Rule 1: The system must maintain the list of relations specified in Certification Rule 2, and must ensure that the only manipulation of any CDI is by a TP, where the TP is operating on the CDI as specified in some relation.

Enforcement Rule 2: The system must maintain a list of relations of the form (UserID, TPi, (CDIa, CDIb, CDIc, ...)), which relates a user, a TP, and the data objects that TP may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

Certification Rule 3: The list of relations in Enforcement Rule 2 must be certified to meet the separation of duty requirement.

Enforcement Rule 3: The system must authenticate the identity of each user attempting to execute a TP.

Certification Rule 4: All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

Certification Rule 5: Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected. Typically, this is an edit program. [Note to the reader: My model of Clark-Wilson integrity allows a TP to access any UDI in the normal manner for access to an object by a process in this system, subject to the constraints of the other (than integrity) policies implemented by the ADF. It is up to the certification process to ensure that the TP accesses only those UDIs it should access for a particular execution. But this is not in keeping with the spirit of moving as much as possible from certification to enforcement, as suggested by Clark and Wilson. One possibility for changing this approach is to add the names of the allowed UDIs for a particular TP to the triples or, perhaps better, to the TP-CDIs relation, which would have to be added to the model since it is currently not included. Doing so would mean that the TP-CDI relation is no longer redundant with the triples.]

Enforcement Rule 4: Only the agent permitted to certify entities may change the list of such entities associated with other entities — specifi-

cally, those associated with a TP. An agent who can certify an entity may not (that is, must not) have any execute rights with respect to that entity.

Acknowledgments

I thank Marshall Abrams for his fundamental insights on access control that led to the modeling approach described in this essay and for his encouragement during the writing of this essay. I thank James Williams of the MITRE Corporation for sharing with me his views on the stages of elaboration of requirements for trusted systems and for many conversations about formal modeling. I thank Charles W. Flink II of AT&T Bell Laboratories for his patient and comprehensive explanations of many design aspects and system calls of System V/MLS, Release 1.2.1.