ANALYSIS OF ACYCLIC ATTENUATING SYSTEMS FOR THE SSR PROTECTION MODEL

Ravinderpal Singh Sandhu

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

ABSTRACT

The distribution of privileges in *domains* of subjects defines the *protection state* of a system. Operations which change this state are themselves authorized by privileges in the current state. This poses an analysis problem of characterizing states which are derivable from a given initial state. Analysis is particularly difficult if creation of new subjects is permitted. Also the need for tractable analysis conflicts with the need for generality in specifying policies. The *Schematic Send-Receive* (*SSR*) *model* resolves this conflict by classifying subjects and objects into *protection types*. The domain of each subject consists of a static type-determined part specified by an *authorization scheme* and a dynamic part consisting of *tickets* (capabilities). We analyze a restricted class of systems in SSR. Specifically, the scheme authorizes creation via a binary relation on types. Our major constraint is that this relation be *acyclic* excepting loops which authorize a subject to create subjects of its own type. Our constraints admit a large class of useful systems.

1. INTRODUCTION

The *access control* or *protection* problem arises in any computer system which permits sharing of information. Such systems are viewed as consisting of *subjects* and *objects*. Subjects model active entities such as users and processes whereas objects model passive entities such as text files. Protection is enforced by ensuring that only those operations for which the invoking subject possesses *privileges* in its *domain* actually get executed. Operations may be performed on objects, e.g., reading a text file, and on subjects, e.g., blocking a process. We regard subjects and objects as mutually exclusive and use *entity* to denote either a subject or object. By definition objects do not possess privileges. Passive entities which possess privileges are modeled as subjects. Our viewpoint is marginally different from the prevalent one where subjects are regarded as a subset of objects.[1, 2, 3]

The distribution of privileges in domains of subjects defines the *protection state* of a system. Henceforth we understand state to mean protection state. *Inert privileges* authorize operations which do not modify the state, e.g., reading a file. *Control privileges* authorize operations which modify the state, e.g., user X authorizes user Y to read file Z. The paradigm is that an initial state is established and thereafter evolves as constrained by control privileges. The challenge is to construct an initial state such that all derivable states are consistent with the underlying policy.

At the simplest level an *authorization policy* defines a set of *safe* states where the distribution of privileges is consistent with the underlying objectives. At all times the system must be in a safe state. Safety considerations are typically concerned with classes of entities rather than individuals, e.g., the policy that only users in department D can access files internal to department D. At a more sophisticated level it is not enough that the system be in a safe state we must additionally ensure the system arrived at the safe state in a proper manner. For instance, the policy that users outside department D may access internal files of department D provided the chairperson of D approves.

A *protection model* provides a framework and formalism for policy specification and must be general enough to conveniently accommodate the kinds of issues outlined above. But generality by itself is not enough. To understand the formal statement of a policy and to assure it captures our intent, we need to characterize the states that a system may arrive at from a given initial state. Since subjects are usually authorized to create new subjects and objects, we are confronted with unbounded systems and it is not certain that such analysis can be decidable let alone tractable without sacrificing generality.

The central point of this paper is to demonstrate that the conflicting goals of convenient generality

CH2150-1/85/0000/0197$01.00 © 1985 IEEE

and tractable analysis can be balanced. Analysis issues were first formalized[4] in context of the well-known access-matrix model.[1, 2, 3] Not surprisingly, analysis is undecidable in this general setting. Moreover, the access-matrix lacks any structure to conveniently address policy concerns. On the other hand the take-grant model[5, 6, 7] and its variations[8] while efficiently analyzable accommodate only a very specific class of simple policies. Thus this conflict between generality and analysis is very real.

In section 2 we present the *Schematic Send-Receive* or *SSR model.* In part SSR is based on Minsky's send-receive transport model.[9] SSR adopts some simplifying assumptions and has a set-theoretic formulation in contrast to Minsky's production-rule formulation. In sections 3, 4 and 5 we develop some basic concepts and terminology for analysis. In section 6 we discuss the restrictions under which the analysis of section 7 is carried out. Section 8 concludes the paper.

2. THE SSR MODEL

Our discussion of SSR here is necessarily abstract. We refer the reader to Sandhu[10, 11] for additional motivational details and applications. The key concept in SSR is *protection types.* Intuitively, instances of the same protection type are treated uniformly by control privileges. Henceforth, we use type as synonymous with protection type. A critical assumption in SSR is *strong typing*, i.e., every entity is created to be of exactly one type which cannot change thereafter. SSR treats a subject's domain as consisting of two parts: a static part determined by the subject's type and a dynamic part which varies with the protection state.

Dynamic privileges are represented as *tickets* (capabilities) of the form Y/x where Y identifies some unique entity and the *right symbol* x authorizes the possessor of this ticket to perform some operation on Y. Capabilities with multiple right symbols are modeled as sets of tickets. Static type-dependent privileges are determined by the *authorization scheme* defined by the *security administrator* when a system is first set-up. Thereafter the scheme cannot be changed. We find it useful to view an authorization scheme as analogous to a database schema and the distribution of tickets as analogous to an extensional database.

TYPES AND RIGHT SYMBOLS

The first step in defining a scheme is to specify the disjoint sets of *object types* $\mathbf{T_O}$ and *subject types* $\mathbf{T_S}$. Their union $\mathbf{T}$ is the entire set of *entity types.* By convention types are named in lower case boldface and entities in upper case normal script.

The next step is to define the right symbols carried by tickets. The set of right symbols $\mathbf{R}$ is partitioned into two disjoint subsets: $\mathbf{R_I}$ the set of inert rights and $\mathbf{R_C}$ the set of control rights. $\mathbf{R_C}$ is fixed and will be defined shortly. $\mathbf{R_I}$ is specified by the security administrator and regarded by SSR as a set of uninterpreted symbols.

Every right symbol x comes in two variations x and xc where c is the *copy flag.* The only difference between Y/x and Y/xc is that the former ticket cannot be copied from one domain to another whereas the latter possibly may be. If follows that presence of Y/xc in a domain subsumes the presence of Y/x but not vice versa. We use x:c to denote either x or xc with the understanding that multiple occurrences of x:c in the same context are either all read as x or all as xc.

We define the ***type of a ticket*** Y/x:c to be τY/x:c, where the ***type function*** τ returns the type of the entity. Conventions for representing tickets, especially regarding the copy flag, extend in an obvious way to ticket types. In particular τY/x and τY/xc are different ticket types. This is an important distinction because of the role of the copy flag. The entire set of ticket types is $\mathbf{T}\times\mathbf{R}$.

The remaining components of a scheme are defined in terms of $\mathbf{T_S}$, $\mathbf{T}$ and $\mathbf{T}\times\mathbf{R}$. SSR recognizes three operations which change the protection state: *copy*, *demand* and *create.*

THE COPY OPERATION

The copy operation moves a copy of a ticket from the domain of one subject to the domain of another leaving the original ticket intact. We often speak of copying a ticket from one subject to another although technically a ticket is copied from one subject's domain to another's domain. In addition to the copy flag this operation is authorized by the link relation defined by control rights and by the filter function which is a component of the scheme.

SSR defines two control rights s and r, read *send* and *receive* respectively, along with their copiable variants, i.e., $\mathbf{R_C}$ = {s, r, sc, rc}. These control rights are interpreted by the *link relation* defined below where *dom* denotes the set of tickets possessed by a subject.

$$\text{link}(A,B) \Leftrightarrow [B/s \in \text{dom}(A) \wedge A/r \in \text{dom}(B)]$$

Existence of link(A,B) is necessary but not sufficient for copying tickets from A to B. A link is established by a send ticket at the source and a receive ticket at the destination. The motivation for choosing these control rights, in preference to say take-grant,[5, 6] is discussed in Minsky[9] and Sandhu.[10]

The final condition required for authorizing a copy operation is defined by the filter function f: $\mathbf{T_S}\times\mathbf{T_S} \rightarrow 2^{\mathbf{T}\times\mathbf{R}}$. The interpretation is that Y/x:c can be copied from dom(A) to dom(B) if and only if all of the following are true.

1. Y/xc $\in$ dom(A)
2. link(A,B)
3. τY/x:c $\in$ $\mathbf{f}$(τA,τB)

In this manner the copy flag, the link relation and the filter function together authorize a copy operation. The first two conditions depend on the distribution of tickets whereas the third condition depends on the scheme. Selectivity in the copy operation is controlled by the third condition and specified entirely in terms of types. SSR imposes no assumptions regarding the role of A and B in this copy operation. It is equally acceptable that copying take place at the initiative of A or B alone or require both to cooperate.

THE DEMAND OPERATION

The demand operation allows a subject to obtain tickets simply by demanding them. A scheme authorizes this operation by the *demand function* d: $\mathbf{T_S} \rightarrow 2^{\mathbf{T}\times\mathbf{R}}$. The interpretation of a/x:c $\in$ d(b) is that every subject of type b can demand the ticket A/x:c for every entity A of type a. In particular control tickets can be demanded. Demand extends some of the conveniences of access-control lists to what is basically a capability framework.

THE CREATE OPERATION

The create operation introduces new subjects and objects in the system. There are two issues here: what types of entities can be created and what happens after a create operation occurs. The first issue is specified in a scheme by the *can-create relation* cc $\subseteq$ $\mathbf{T_S}\times\mathbf{T}$. The interpretation is that subjects of type a are authorized to create entities of type b if and only if cc(a,b).

The second issue is specified by a *create-rule* for every pair in cc. Let subject A of type a create entity B of type b. If B is an object the $\langle$a,b$\rangle$ create-rule tells us which tickets for B are placed in dom(A) as a result of this operation. If B is a subject the create-rule must also tell us which tickets for A are placed in dom(B). SSR requires every create-rule be *local* in that the only tickets introduced are for the creating and created entities in the domains of the creating and created entities. The idea is that frequently occurring incremental events such as creation should immediately have only a local incremental impact on the state. Each ticket generated by a create-rule may or may not carry the copy flag as specified in the rule.

The facility to generate copiable control tickets for a created subject is certainly useful. The policy decision as to whether the creator or the created subject or both get these tickets is properly left open by the model. Placing copiable control tickets for the creator in the created subject's domain is also a valid policy option. For example, a policy may allow an "ordinary user" U to create a very powerful subject M of type "system manager" with the intention that M be used solely for experimentation by U in isolation from the rest of the system. If M gets copiable control tickets for U then M may create a complex sub-system with which U can interact. Placing copiable control tickets for the creator in the creator's own domain is a more subtle issue. As we will see in section 6, this turns out to be useful in working around some of the problems introduced by insisting a scheme be defined entirely in terms of types. At any rate there is no obligation to use this facility.

We emphasize the create-rule may be different for each pair in cc. We omit the symbolic formalism for specifying create-rules. The important point is the create-rules are specified in terms of types and SSR only requires locality.

SUMMARY

In summary, the SSR model requires the security administrator to specify an authorization scheme by defining the following components.

1. The entity types $\mathbf{T}$ partitioned into subject types $\mathbf{T_S}$ and object types $\mathbf{T_O}$.
2. The rights $\mathbf{R}$ partitioned into inert rights $\mathbf{R_I}$ and a fixed set of control rights $\mathbf{R_C}$ = {s, r, sc, rc}.
3. The filter function f: $\mathbf{T_S}\times\mathbf{T_S} \rightarrow 2^{\mathbf{T}\times\mathbf{R}}$.
4. The demand function d: $\mathbf{T_S} \rightarrow 2^{\mathbf{T}\times\mathbf{R}}$.
5. The can-create relation cc $\subseteq$ $\mathbf{T_S}\times\mathbf{T}$.
6. A local create-rule for each pair in cc.

A *system* is specified by defining an authorization scheme, the initial set of entities and the initial distribution of tickets. Thereafter the system state evolves by copy, demand and create operations.

REVOCATION AND DELETION

SSR lacks facilities for revocation of tickets. This is a deliberate decision justified by adopting the *restoration principle* that whatever can be revoked can be restored. If a ticket obtained by a copy or demand operation is revoked it is easily restored by repeating the operation. However, if a ticket introduced by a create operation is revoked it may not be restorable by repeating the operation since each created entity is unique. Also tickets distributed in the initial state may not be restorable. If we assume tickets distributed in the initial state or introduced by create-rules are irrevocable the restoration principle does not entail any loss of generality in context of SSR.

A similar argument applies to deletion of entities. Here the restoration principle requires that an entity which can be deleted should be replaceable by an equivalent entity. In general this rules out deletion of entities present in the initial state. Regarding deletion of entities created subsequently it is always possible to re-create an entity of the same type as was deleted. In other words the individuality of created entities is not significant whereas the individuality of entities in the initial state is significant.

Revocation and deletion policies consistent with the restoration principle can be ignored in a worst-case scenario where we assume all subjects will cooperate with one another. This leads to monotonic evolution of the system state which greatly simplifies analysis. The problem of specifying revocation and deletion policies still remains. For now we have chosen to set aside this problem and focus on policies for acquisition of tickets and creation of entities.

3. STATES AND HISTORIES

A change in state caused by a single copy, demand or create operation is called a *transition.* A transition is *legal* provided there is proper authorization for the operation causing it. A *history* is a sequence of legal transitions. Histories are denoted by upper case letters and states by lower case letters or special symbols. Unless otherwise mentioned a history is applied to the initial state. Any state that can be derived by a history is *derivable.*

Because the authorization for creates and demands is entirely in terms of types, we can assume without loss of generality that all create operations occur first followed by demand operations and finally followed by copy operations. We say such histories are in *canonical form.* Formally we have the following property.

Lemma 1: For every history H deriving state h there is a history H' in canonical form which derives state h.

Proof: Obtain H' from H by rearranging the operations in H to conform to the canonical form while preserving the relative order of each kind of operation. The legality of the transitions in H' follows from their legality in H.

In analysis we are interested in functions and relations which depend on the state, e.g., dom and link. When appropriate we qualify these with a superscript to identify the state, e.g., dom^h and link^h identify the context as state h. The initial state is identified by the superscript 0. The set of subjects and entities in state h are respectively denoted by SUB^h and ENT^h.

Both dom and link exhibit a monotonic property because of the absence of revocation and deletion, i.e., if g is derived from h then $\text{link}^h \subseteq \text{link}^g$ and for all $A \in \text{SUB}^h$, $\text{dom}^h(A) \subseteq \text{dom}^g(A)$. Because the functions and relations used in analysis depend on the presence rather than absence of tickets in domains, this monotonic property extends to all functions and relations we consider.

4. THE FLOW FUNCTION

The flow function expresses the authorization for copying tickets from one subject to another in a given state accounting for indirect as well as direct copying. For every pair of subjects its value is a set of ticket types determined by the state and scheme. Its definition is based on the following notion.

Definition 2: There is a *path*^{h} from A to B provided either $\text{link}^h(A,B)$ or there exists a sequence of subjects $C_1,...,C_n$ such that all of the following are true.

1. $\text{link}^h(A,C_1)$
2. $\text{link}^h(C_i,C_{i+1})$, i=1..n-1
3. $\text{link}^h(C_n,B)$

In the former case we say the path consists of a single link while in the latter case we say the path *traverses* subjects $C_1,...,C_n$.

Consider a path from A to B which traverses $C_1,...,C_n$. Let $Y/xc \in \text{dom}(A)$. Y/xc can be copied from A to B using this path provided Y/xc can be copied across each link in the path. Further, Y/x can be copied from A to B using this path provided Y/xc can be copied from A to C_n and Y/x copied from C_n to B, that is the copy flag must be copied on all except the last link. This leads to the following definition.

Definition 3: Define the *capacity* of a path^h from A to B as follows: if the path consists of a single link its capacity is $f(\tau A,\tau B)$ otherwise the path traverses subjects $C_1,...,C_n$ and y/x:c is in its capacity if and only if all of the following are true.

1. $y/xc \in f(\tau A,\tau C_1)$
2. $y/xc \in f(\tau C_i,\tau C_{i+1})$, i=1...n-1
3. $y/x{:}c \in f(\tau C_n,\tau B)$

Note that only the types of entities involved in this definition are significant, not their specific identities. We are now ready to define the flow function.

Definition 4: For every state h define the *flow* function flow^h: $\text{SUB}^h \times \text{SUB}^h \rightarrow 2^{\mathbf{T}\times\mathbf{R}}$ by $\text{flow}^h(A,B) = \{y/x{:}c \mid$ there exists a path^h from A to B whose capacity includes $y/x{:}c\}$. By convention $\text{flow}^h(A,A)$ is $\mathbf{T}\times\mathbf{R}$.

An alternate definition is that $\text{flow}^h(A,B)$ is the union of the capacity of all paths in state h from A to B. The convention concerning $\text{flow}^h(A,A)$ is consistent with the underlying intuition and is convenient. Computation of flow^h is straightforward in principle and of polynomial complexity in the number of subjects in state h.

The fundamental issue in analysis is to predict behavior of the flow function. This is especially so since create and demand operations are authorized solely by the scheme whereas copy is authorized by both the scheme and the distribution of tickets. Because flow is monotonic, for a given pair of subjects it can only increase in derived states. From this fact it can be shown there exists a derivable state with the maximum value of flow between all subjects in SUB^0. We call such a state a *maximal state*. In general the maximal state is not unique. Indeed the system can continue to evolve indefinitely by creation of new entities. A maximal state is a worst-case concept and will be derived only if all subjects cooperate towards this end. We now formalize this concept and prove that maximal states indeed exist.

5. MAXIMAL STATES

To focus on changes in flow with respect to subjects in SUB^0 we introduce the following notions of reducibility and equivalence.

Definition 5: A derivable state h is *0-reducible* to a derivable state g written $h \preceq_0 g$ if and only if for all subjects A, B $\in \text{SUB}^0$, $\text{flow}^h(A,B) \subseteq \text{flow}^g(A,B)$.

For a given system two derivable states h and g are *equivalent* written $h \equiv_0 g$ if and only if $h \preceq_0 g$ and $g \preceq_0 h$.

Because of its focus on the initial set of subjects, this equivalence relation partitions the derivable states into a finite collection of equivalence classes. For future reference we state this as a lemma.

Lemma 6: For every system there are a finite number of equivalence classes of derivable states.

Proof: For every pair of subjects in SUB^0, flow can take on at most $|2^{\mathbf{T}\times\mathbf{R}}|$ distinct values. Hence there are at most $|\text{SUB}^0|^2$ times $|2^{\mathbf{T}\times\mathbf{R}}|$ distinct equivalence classes, which is clearly finite.

We are now ready to formalize the notion of maximal state.

Definition 7: For a given system a derivable state m is a *maximal state* if and only if for every derivable state h, $h \preceq_0 m$.

Clearly all maximal states are equivalent. The flow function in a maximal state completely defines the potential for copying tickets between subjects present in the initial state.

The existence of maximal states is a consequence of the monotonic nature of state transitions in SSR. Consider a state h in which operation op is authorized. If *op* is a demand or copy operation it continues to be authorized in every state derived from h, because the conditions on which the authorization depends cannot be revoked. If *op* is a create operation the situation is slightly different, because each create operation is unique and cannot be repeated. We can account for creates by the formulation: if *op* is authorized in state h then in every history applied to h either *op* will have occurred or will continue to be authorized. This leads to the following property.

Lemma 8: Given an arbitrary finite collection $\mathcal{H}$ of derivable states there exists a derivable state m such that for every $h \in \mathcal{H}$, $h \preceq_0 m$.

Proof: By induction on size of $\mathcal{H}$. The basis follows by setting $\mathcal{H}$ to ϕ and m to the initial state. Assume the lemma holds for all $\mathcal{H}$ of size n and consider $\mathcal{H}$ of size n+1. Then $\mathcal{H} = \mathcal{G} \cup \{h\}$ where $|\mathcal{G}|$ is n. By hypothesis there is a derivable state g which satisfies the lemma for $\mathcal{G}$. For the induction step we show for every pair of derivable states g, h there exists a derivable state m such that $g \preceq_0 m$ and $h \preceq_0 m$. Let g, h be established by histories G, H respectively. Let N be any interleaving of G and H which preserves the relative order of the transitions within G and H. Construct M by eliminating the second occurrence of every duplicate create operation in N. That every transition in M is legal follows from the discussion above. Let m be the state established by M. By construction every path in state g, and every path in state h, is duplicated in state m. That completes the induction step and the lemma follows.

Proving the existence of maximal states is now straightforward.

Theorem 9: For every system there exists a maximal state.

Proof: By lemma 6 there are a finite number of equivalence classes of derivable states. Let $\mathcal{H}$ be a collection of derivable states which contains exactly one representative from each equivalence class. The theorem follows by applying lemma 8 to $\mathcal{H}$.

Unfortunately this proof is non-constructive and thereby does not provide a method for computing maximal states. It is easy to demonstrate that in order to derive a maximal state from the initial state we generally need to create new subjects.[10] The problem is to determine which new subjects need to be created. Although we conjecture this problem is decidable, and even possibly tractable, in the general case we have not discovered a method and can only offer approximations.[10] We do have exact solutions in several special cases of interest one of which we discuss in the remainder of this paper.

The most troublesome aspect in deriving a maximal state is the create operation. If creation of subjects is not allowed a maximal state is trivially derived by repeatedly executing demand and copy operations until the state stabilizes. It is easy to construct straightforward polynomial algorithms for this. From lemma 1 we know all create operations can be assumed to occur at the beginning of a history, so in particular a maximal state can be derived by such a history. The real problem is to determine the initial sequence of creates needed for this purpose. This idea underlies the analysis of section 7. But first we must discuss the restrictions under which this analysis is carried out.

6. ACYCLIC ATTENUATING SYSTEMS

Visualize the **cc** relation as a directed graph with vertex set **T** and an edge from **a** to **b** if and only if **cc(a,b)**. We say **cc** is *acyclic* if this graph contains no cycles excepting *loops*, i.e., edges which connect a vertex to itself. An object type can only have incoming edges so the acyclic restriction has no effect regarding creation of objects. For subject creation the acyclic restriction states that if subjects of type **a** can directly or indirectly create subjects of type **b** then it is not possible for subjects of type **b** to directly or indirectly create subjects of type **a**, unless **a** = **b**. This is a fairly natural and intuitive restriction and is consistent with modern approaches to system design such as layering, information hiding, data abstraction, etc. We say a scheme is acyclic if its **cc** relation is acyclic. Similarly a system is acyclic if its scheme is acyclic.

To motivate the one additional constraint we need for our analysis consider subject A of type **a** such that **cc(a,a)**. Subject A can create a child A' of type **a** which in turn can create a child A" of type **a** and so on. The possibility of indefinitely long chains of create operations complicates analysis. The acyclic restriction eliminates certain kinds of indefinitely long chains but does not eliminate them completely because loops are allowed. We will account for loops in **cc** by insisting that the child A' be "no more powerful" than its creator A. Since A and A' are both of type **a** this is a somewhat curious requirement. The crucial difference between A and A' lies in the tickets introduced by the ⟨a,a⟩ create-rule when A creates A'.

Our first restriction on the ⟨a,a⟩ create-rule is that immediately after the create operation dom(A') ⊆ dom(A). This is consistent with the principle of attenuation[12] in that a newly created subject does get more tickets than its creator. The control tickets that can be placed in dom(A') by a local create-rule are a subset of {A'/s:c, A'/r:c, A/s:c, A/r:c}. Placing A'/s or A'/r in dom(A') is meaningless and can be assumed not to occur. Placing A/s or A/r in dom(A') respectively requires A/s or A/r be placed in dom(A) which again is meaningless and can occur without loss of generality. So the restriction has practical impact only with respect to copiable control tickets.

Our second restriction is more subtle. It requires that if a ticket for A' is placed in dom(A) the corresponding ticket for A should also be placed in dom(A). With respect to non-copiable control tickets this again requires at most that A/s or A/r be placed in dom(A) and entails no loss of generality. The crucial implication is that A'/sc or A'/rc in dom(A) implies respectively that A/sc or A/rc in dom(A). Without this restriction it is possible copiable control tickets for A' exist but not for A. Since this facilitates establishment of links to and from A', A' may be more powerful than A even though they are of the same type.

The formal definition of our restrictions is as follows.

> **Definition 10:** A scheme is *attenuating* if every ⟨a,a⟩ create-rule is such that when A of type **a** creates A' of type **á** the tickets introduced by the create-rule satisfy the following constraints.
>
> 1. Tickets placed in dom(A') are also placed in dom(A).
> 2. If A'/x:c is placed in dom(A) then A/x:c is also placed in dom(A).
>
> A system is attenuating if its scheme is attenuating.

Note the attenuating restriction applies only to create-rules for loops in **cc**. The net effect is that copiable tickets for a created subject A' can be generated by a ⟨a,a⟩ create-rule only if the creator A gets the corresponding copiable ticket for itself.

A slightly different formulation of this effect is to require that copiable tickets for A' be generated by a ⟨a,a⟩ create-rule only if the creator A possesses the corresponding copiable ticket for itself. The latter formulation is contrary to SSR since the create-rule is influenced by the creator's domain whereas SSR create-rules are determined entirely by types. Definition 10 captures the spirit of this formulation within the SSR framework. Indeed once A has created one subject of type τA both formulations are essentially equivalent.

A scheme is ***acyclic attenuating*** if it is acyclic and attenuating. Similarly a system is acyclic attenuating if its scheme is acyclic attenuating system. We believe such systems will suffice for a wide variety of policies arising in practise. It is of interest that all schemes discussed in Sandhu[11] are acyclic attenuating.

7. FLOW ANALYSIS FOR ACYCLIC ATTENUATING SYSTEMS

Our strategy for computing a maximal state is as follows. From the given initial state we first derive a state u entirely by create operations, with the objective that entities in state u will account for all possible entities that can exist. We achieve this by defining a mapping σ (read surrogate) from all possible entities to entities in state u such that entity A is simulated by σA. In particular σ maps every entity present in the initial state to itself. In our proof we show that for every history H which derives h from the initial state there exists a history G, without create operations, which derives g from u such that $\text{flow}^h(A,B) \subseteq \text{flow}^g(\sigma A,\sigma B)$. Since G has no creates a maximal state can be computed from u by executing demand and copy operations until the state stabilizes. Moreover, if the construction of u introduces only a polynomial number of new subjects the entire computation can be done in polynomial time. The remainder of this section elaborates and formalizes this argument in context of acyclic attenuating systems.

Consider a subject A of type **a**. By definition **cc** is acyclic. For the moment assume **cc** contains no loops. We say A is *unfolded* if A creates one entity of each type **b** such that **cc**(**a**,**b**). For each **b** the entity created in this manner is called the **b**-*surrogate* of A. The idea is the **b**-surrogate of A will simulate all type **b** children of A. To account for descendants of A's children we apply the unfolding construction recursively to the surrogates of A and so on until all descendants of A are unfolded. Because **cc** is acyclic and without loops this construction eventually terminates. At this point we say A is *fully unfolded.* The initial state is fully unfolded if all subjects in SUB^0 are fully unfolded.

If **cc** contains loops we first eliminate the loops and fully unfold the initial state. Then for every A in the resulting state such that **cc**(τA,τA), we let A create A' of type τA. The intention here is that the τA children of A will be simulated by A itself rather than by A'. Why then create A' at all? The reason for this goes back to the motivation underlying our attenuating restriction on create-rules for loops in **cc**, i.e., it is possible this create operation may generate copiable control tickets for A in dom(A).

We now formally define this construction.

> **Definition 11:** Given any initial state 0 with an acyclic attenuating scheme derive the *fully unfolded state* u as follows.

```
1. Let cc' = cc - {⟨a,a⟩| cc(a,a)}.
2. Mark all subjects in $\text{SUB}^0$ as folded.
3. While there exists a folded subject A do
      Mark A as unfolded
      For all b such that cc'($\tau$A,b) do
         Let A create B of type b
         Call B the b-surrogate of A
         If B is a subject mark it as folded
```

```
4. For all subjects A in the resulting state do
       If cc($\tau$A,$\tau$A) then let A create
              a subject of type $\tau$A
```

Clearly each create operation in this construction is authorized by **cc**, so u is a derivable state. Because of the absence of cycles and loops in **cc'** this construction is guaranteed to terminate.

Lemma 12: The construction of definition 11 terminates.

Proof: We need to show that step 3 of the construction terminates. Consider A ∈ SUB^0. The descendants of A generated by step 3 form a tree with A at the root and each created entity a child of its creator. Because each subject creates only one child of each type, each node in the tree has a finite number of children. If we follow a path in this tree from the root to any of A's descendants the types of entities encountered in this path must all be different, otherwise **cc'** contains a cycle or loop. Since all nodes in this path excepting the last one must be subjects, the maximum length of such a path is $|\mathbf{T_S}|+1$. Hence the depth of the tree is finite.

Next we define a mapping to relate each entity that can be created to the entity in state u which simulates it.

Definition 13: Given any initial state with an acyclic attenuating scheme, for every derivable state h define the *surrogate* function σ: $ENT^h \rightarrow ENT^u$ as follows.

1. If A ∈ ENT^0 then σA is A.
2. If B creates A and $\tau A \neq \tau B$ then σA is the τA-surrogate of σB (step 3 of definition 11).
3. If B creates A and $\tau A = \tau B$ then σA is σB.

Observe that σ preserves types, i.e., $\tau\sigma A = \tau A$. The following is another crucial property.

Lemma 14: For an acyclic attenuating system if A creates B in deriving h then tickets which would be introduced by pretending that σA creates σB are present in $dom^u(\sigma A)$ and $dom^u(\sigma B)$.

Proof: If A creates B and $\tau A \neq \tau B$ then σA indeed creates σB in constructing u from the initial state (step 3 of definition 11). Since σ preserves types the lemma follows. If A creates B and $\tau A = \tau B$ then σA is σB. In constructing u then σA creates A' of type $\tau A = \tau\sigma A$ (step 4 of definition 11). By definition 10 all tickets which would be introduced by pretending that σA creates itself are thereby present in $dom^u(\sigma A)$.

This lemma is crucial because it suggests that in constructing u from the initial state we have managed to account for all possible create operations.

We are now ready to prove the central result of this paper.

Theorem 15: For every acyclic attenuating system, for every history H which derives h from the initial state there exists a history G, without create operations, which derives g from u such that

$$(\forall A,B \in SUB^h)[flow^h(A,B) \subseteq flow^g(\sigma A,\sigma B)]$$

Proof: By lemma 1 we may assume H is in canonical form, i.e., all create operations occur first followed by demand operations and then by copy operations. G is obtained from H by replacing the individual transitions of H as follows while preserving the relative order.

1. Ignore all create operations in H.
2. "A demands B/x:c" in H is replaced by "σA demands σB/x:c" in G.
3. "Copy A/x:c from B to C" in H is replaced by "copy σA/x:c from σB to σC" in G.

We first establish the following assertions.

I. Every transition in G is legal.

II. A/x:c ∈ $dom^h(B) \Rightarrow \sigma A$/x:c ∈ $dom^g(\sigma B)$.

III. $link^h(A,B) \Rightarrow link^g(\sigma A,\sigma B)$

Assertion III follows trivially from II, and is crucial to the second part of the proof. Assertions I and II are proved by induction on the number of copy operations in H.

Basis Case: Let there be no copy operations in H, so H consists of creates followed by demands while G consists entirely of demands.

Assertion I: By construction every operation "A demands B/x:c" in H is replaced by "σA demands σB/x:c" in G. Since σ preserves types, the demand operation in G is legal.

Assertion II: Without no copy operations there are only three ways by which A/x:c can appear in $dom^h(B)$. If A/x:c ∈ $dom^0(B)$ then $\sigma A = A$ and $\sigma B = B$ so assertion II is trivially true. If

$A/x{:}c \in dom^h(B)$ because of a create operation in H assertion II follows from lemma 14. Finally, if $A/x{:}c \in dom^h(B)$ because of a demand operation then assertion II follows from the corresponding demand operation in G.

Induction Step: Assume assertions I and II are true for every history with k copy operations and consider a history H with k+1 copy operations. Since H is in canonical form it consists of an initial sequence H' with k copy operations followed by a single copy operation. Let h' be the state derived by H'. Let G' be the required modification of H'. By hypothesis I, G' is a history. Let g' be the state derived by G'. Let the final operation of H be "copy A/x:c from B to C." By construction the final operation of G is "copy $\sigma A/x{:}c$ from σB to σC."

Assertion I: For the final operation of H to be legal the following conditions must be true.

1. $A/xc \in dom^{h'}(B)$
2. $link^{h'}(B,C)$
3. $\tau A/x{:}c \in f(\tau B,\tau C)$

By hypothesis II and the fact that σ preserves types the corresponding conditions required to authorize the final operation of G are true in state g'.

Assertion II: h differs from h' at most by $A/x{:}c \in dom^h(C)$. By construction the final operation of G ensures that $\sigma A/x{:}c \in dom^g(\sigma C)$. This completes the induction step and we have established both assertions.

It remains to prove that

$$(\forall A,B \in SUB^h)[flow^h(A,B) \subseteq flow^g(\sigma A,\sigma B)]$$

We do so by showing that for every $path^h$ from A to B there is a $path^g$ from σA to σB with the same capacity as the $path^h$. The proof is by induction on the number of links. For the basis case consider a $path^h$ from A to B of length 1, that is $link^h(A,B)$. By assertion III we have $link^g(\sigma A,\sigma B)$. Since σ preserves types the basis case is true. Assume the hypothesis is true for every $path^h$ of length k and consider a $path^h$ from A to B of length k+1. Then there is some C with a $path^h$ from A to C of length k and $link^h(C,B)$. By induction hypothesis there is a $path^g$ from σA to σC with the same capacity as the $path^h$ from A to C. By assertion III we have $link^g(\sigma C,\sigma B)$. Since σ preserves types it follows there is a $path^g$ from σA to σB with the same capacity as the $path^h$ from A to B.

The essence of theorem 15 is that all histories applied to the initial state can be simulated by histories without create operations applied to state u. Define # to be the state which results from state u by repeatedly executing demand and copy operations until the state stabilizes. We have the following corollary.

Corollary 16: For every acyclic attenuating system, for every history H which derives state h from the initial state

$$(\forall A,B \in SUB^h)[flow^h(A,B) \subseteq flow^{\#}(\sigma A,\sigma B)]$$

In particular

$$(\forall A,B \in SUB^0)[flow^h(A,B) \subseteq flow^{\#}(A,B)]$$

Proof: Immediate from theorem 15, definition of # above and definition 13 of σ.

In other words for all derivable states h in a acyclic attenuating system, $h \preceq_0 \#$ and thereby # is a maximal state.

Clearly the # state is derivable. To derive # from u requires time polynomial in $|SUB^u|$. For each subject $A \in SUB^0$ the construction of u from the initial state introduces a constant number of new subjects determined by τA. Thus the entire computation is polynomial in $|SUB^0|$. We take this as evidence that the computation is tractable. Of course, in practise one will need careful engineering of the algorithms to achieve acceptable performance. We mention that $|SUB^u|$ may exceed $|SUB^0|$ by a factor exponential in $|T_S|$. In the worst case the straightforward algorithms for computing $flow^{\#}$ will then be exponential in $|T_S|$. This will happen only if cc is highly non-sparse. At any rate this exponential factor involves $|T_S|$ rather than $|SUB^0|$ and may be tolerable.

8. CONCLUSION

This paper has focused on the problem of balancing generality and analyzability in a protection model. We defined the SSR protection model with the key idea of strong typing of entities. We demonstrated that under reasonable assumptions tractable analysis of systems specified in SSR is feasible. For a detailed application of SSR see Sandhu.[11]

The concept of authorization scheme is applicable to control rights other than send and receive. For example, we can define the Schematic Take-Grant model by modifying the control rights of SSR to be take and grant and suitably changing the definition of link. We can go a step further and allow the control rights and definition of link to be specified as a component of the scheme rather than being fixed. Even further we can allow multiple definitions of link to co-exist each with its own filter function. The analysis result of this paper extends to such generalizations. We have chosen to formulate our model with a single set of specific control rights because the model is complex enough anyway and the particular control rights chosen are of interest.

The analysis developed in this paper is based on the worst-case concept of maximal state. In practise it is desirable to analyze systems under assumptions about the behavior of specific subjects. In SSR such analysis can be reduced to worst-case analysis. SSR, being a model rather than a mechanism, does not insist that behavioral restrictions built into a scheme be enforced at run-time. Any restriction imposed by the scheme can be implemented by one of the following options.

1. Enforce the restriction at run-time.
2. Assume subjects will honor the restriction.
3. Prove subjects will honor the restriction.

The second alternative allows us to incorporate behavioral assumptions as part of a scheme.

In addition to balancing analysis and generality an important criteria for a protection model is that it permit a variety of implementations. The three alternatives listed above can be used in a variety of combinations to implement a scheme. This allows for a trade-off between the degree of trust invested in subjects and the run-time or verification-time cost of enforcement. Investigation of such implementation trade-offs using a variety of underlying mechanisms is a research topic we intend to pursue.

REFERENCES

1. Denning, D.E. and Denning, P.J., "Data Security", *ACM Comp. Surv.*, Vol. 11, No. 3, Sept. 1979, pp. 227-249.

2. Graham, G.S. and Denning, P.J., "Protection - Principles and Practice", *Proc. SJCC*, AFIPS, 1972, pp. 417-429.

3. Linden, T.A., "Operating System Structures to Support Security and Reliable Software", *ACM Comp. Surv.*, Vol. 8, No. 4, Dec. 1976, pp. 409-445.

4. Harrison, M.H., Russo, W.L. and Ullman, J.D., "Protection in Operating Systems", *CACM*, Vol. 19, No. 8, Aug. 1976, pp. 461-471.

5. Lipton, R.J. and Snyder, L., "A Linear Time Algorithm for Deciding Subject Security", *JACM*, Vol. 24, No. 3, Dec. 1977, pp. 455-464.

6. Jones, A.K., Lipton, R.J. and Snyder, L., "A Linear Time Algorithm for Deciding Security", *Proc. 17th Symp. on the Foundations of Comp. Sci.*, IEEE, 1976.

7. Snyder, L., "Formal Models of Capability-Based Protection Systems", *IEEE Trans. Comp.*, Vol. C-30, No. 3, March 1981, pp. 172-181.

8. Lockman, A. and Minsky, N., "Unidirectional Transport of Rights and Take-Grant Control", *IEEE Trans. Softw. Engg.*, Vol. SE-8, No. 6, November 1982, pp. 597-604.

9. Minsky, N., "Selective and Locally Controlled Transport of Privileges", *ACM Trans. Prog. Lang. and Sys.*, Vol. 6, No. 4, October 1984, pp. 573-602.

10. Sandhu, R.S., *Design and Analysis of Protection Schemes Based on the Send-Receive Transport Mechanism,* PhD Thesis, Rutgers University Technical Report DCS-TR-130, 1983.

11. Sandhu, R.S., "The SSR Model for Specification of Authorization Policies: A Case Study in Project Control", *Proc. 8th IEEE COMPSAC*, November 1984, pp. 482-491.

12. Saltzer, J.H. and Schroeder, M.D., "The Protection of Information in Computer Systems", *Proc. of IEEE,* Vol. 63, No. 9, Sept. 1975, pp. 1278-1308.