

Modeling Insecurity: Policy Engineering for Survivability

Prasad Naldurg
Department of Computer Science
University of Illinois at Urbana-Champaign
IL, 61801, USA
naldurg@cs.uiuc.edu

Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
IL, 61801, USA
rhc@cs.uiuc.edu

ABSTRACT

We present an access-control policy specification and verification process that is well-suited to model survivability of information resources under threat of compromise. Our process differs from the traditional policy engineering methodology in many ways. First, we contend that traditional safety-property modeling cannot provide any guarantees when the policy enforcement mechanisms are compromised. Therefore, we extend traditional access control specifications by modeling insecure states and transitions explicitly, to describe possible system behavior after compromise.

Next, we observe that it may not always be possible to recover from an insecure state, and both compromise and recovery impact the availability of information. Based on these observations, we refine traditional information security properties as liveness assertions and explicitly add recovery actions to our specifications, to guarantee resources are available to legitimate users infinitely often, in spite of malicious attacks or inadvertent compromise. We explain our process using an example behavioral specification and show how we can define different measures of availability and verify them using standard model-checking techniques within this framework.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—Access controls

General Terms

Security, Design

Keywords

Security policies, Access control models, Survivability, Liveness, Availability

1. INTRODUCTION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SSRS '03, October 31, 2003, Fairfax, Virginia, USA

Copyright 2003 ACM 1-58113-784-2/03/0010 ...\$5.00.

Traditional security engineering for information protection begins with the *specification* of confidentiality, integrity, and information flow policies. These policies are usually defined within the framework of an abstract state-transition model of the information system. The states correspond to a snapshot of entities, resources and policies, and the transitions are actions (such as read and write) that can change their values. Policies typically specify what is and what is not allowed by different entities in the system, in terms of their information access behavior and interaction [3].

A policy specification divides the states and transitions of a system into two types: authorized and unauthorized. Authorized states correspond to those states explicitly allowed by the specification. Transitions are only allowed between authorized states. A system is never allowed to bootstrap in an unauthorized state. Unauthorized actions from authorized states are constrained by resource-access mechanisms that also enforce the policies. These mechanisms (e.g., reference monitors) typically intercept access requests and apply a test of policy membership to decide if the action is allowed or not according to the policy specification.

Security engineers translate the policy specification into an implementation by initializing the system in a verifiably authorized state, and designing appropriate resource access mechanisms that explicitly check for policy membership. The implementation is derived from the model, and the offline process of state-transition based policy engineering provides security engineers a methodology to validate their designs formally. Actual testing is required after implementation, to verify if a system implements the policies correctly.

In order to keep the analysis of a system model tractable, many researchers focus on finite-state state-transition models that severely constrain transition access behavior. Confidentiality, integrity and some information flow policies are typically modeled as safety properties (“nothing bad happens”) within this framework. The safety question is usually decidable for such models, which is answered by exploring the state space exhaustively for all possible authorized states and transitions to show that bad things do not happen. This modeling and validation process is an attractive choice for security engineers seeking to certify that a given model provides provable guarantees.

We claim that the formal modeling and validation of safety properties using a state-transition model as described does not represent real system behavior. One major drawback of the traditional policy engineering process is that once a policy enforcement mechanism is compromised, the

assurance provided by the model becomes worthless. Safety-property modeling does not account for the fact that parts of the system may become insecure due to vulnerability exposures or attacks, and the same resource may be in authorized and unauthorized states over the lifetime of the system. In real systems that are constantly under threat of attack, discovering new vulnerabilities may lead to compromise and invalidate the assurance.

In this article, we explore the design a new policy engineering process that can specify how to recover from information compromise by explicitly modeling *insecurity*. Instead of designing for safety, which cannot account for unauthorized behavior, we argue that traditional policies need to be refined as liveness properties ("something good eventually happens") to accurately model compromise and enable *recovery-oriented security*.

We first explore how compromised policy enforcement mechanisms can lead to information exposure, and investigate what it means to recover from integrity and confidentiality failures. We observe that a confidentiality failure, e.g., when an unauthorized entity obtains confidential information, can be handled by isolating the entity, and preventing it from disseminating or using that information. Integrity compromise may be handled by rolling-back the state of the system to a previously known uncompromised state.

In order to develop a theory of information protection that explicitly includes the notion of recovery from compromise, we focus on availability, specified as a liveness property, as a *measure of survivability*. The goal of many attackers is to prevent authorized users from accessing their authorized information. As a result of a threat or exposure, information that was once available to authorized users may now become unavailable or unusable.

The rest of this article is organized as follows: In Section 2, we present definitions of standard information-protection policies and describe the traditional policy engineering process. In Section 3, we show we can extend these standard definitions to include notions of compromise and recovery in a state-transition model and introduce availability as a property that can be described in this framework. In Section 4, we show we can model compromise, survivability and recovery. Section 5 presents related work and we summarize our contributions in Section 6.

2. BACKGROUND

Traditionally, information protection is defined in terms of *confidentiality* and *integrity* properties. *Availability* is also often mentioned as a desirable information security property but is not as rigorously studied as the other two. To the best of our knowledge, the specification and use of availability policies for recovery-oriented security has not been proposed before. *Information Flow* security is another desirable property and is intrinsically related to confidentiality. A complete discussion of information flow survivability is outside the scope of this paper.

In this section, we present standard definitions (adapted from [3]) of confidentiality and integrity policies, and describe how they are enforced using access control mechanisms. We also briefly describe the standard state-transition modeling process and show how these properties are described as safety properties within this formal framework. This section provides a notational background to the description of compromise, recovery, and availability policies

in Section 3.

Our model consists of a set S of subjects, set O of objects, R the set of rights, and the access matrix $A[s, o] \in R$, similar to the standard formalism used in [9]. Let the set of requests $Q = \{\langle s, o, r \rangle \mid s \in S, o \in O, r \in R\}$, where each request $q \in Q$ corresponds to an information access command issued by subject s on object o using object interface r . A state or configuration $v \in V$ in our system is the snapshot of the set of all entities and resources that are currently active in the system, as well access rights installed in the access matrix, represented as (S, O, A) . The transition function $T : V \times Q \rightarrow V$ transforms the system from one state to another when the request is executed. Therefore, we define a transition $v \rightarrow v'$ as $\langle (S, O, A), \{q\} \rangle \rightarrow \langle (S', O', A') \rangle$ where $q \in Q$. All or no elements of (S, O, A) may change during the transition. In order to keep the notation compact, we only explicitly include resource sets that change as a result of executing the request on the current state of the system.

2.1 Confidentiality and Integrity

Confidentiality is concealing information, or preventing unauthorized access to a resource, and integrity refers to preventing unauthorized modification. They are formally defined as follows (adapted from [3]):

- **Confidentiality:** A confidentiality policy P_C with respect to subset $C \subseteq O$ of objects partitions the set of subjects S into two sets S_C and \bar{S}_C . Subjects in \bar{S}_C have no knowledge of the existence or contents of the information resources in C , nor can they access it using any of the rights in $A[s', o], \forall s' \in \bar{S}_C$. P_C explicitly specifies rights r that subjects $s \in S_C$ can use to retrieve specific information from C .
- **Integrity:** An integrity policy P_I with respect to subset $I \subseteq R$ of objects partitions the set of subjects S into two sets S_I and \bar{S}_I . Subjects in \bar{S}_I are not allowed to modify the information in I . P_I explicitly specifies rights r that subjects $s \in S_I$ can use to use to modify specific information in I . Changes made by any entity in S_I are trusted by all entities in S_I .

Information flow policies are an alternative way of looking at information protection. An information flow policy quantifies the effect of observing a set of information requests and responses and inferring the protection state of the system from this process.

2.2 Modeling Dynamic Behavior

We abstract information access behavior of our entities, borrowing standard filesystem terminology, in terms of three types of interface methods, i.e., $R = \{read, write, execute\}$. These methods stand as place-holders for particular request types. In real implementations, these methods may map to different application specific commands. The *read* method corresponds to the interface used to retrieve an information resource. A state variable in the reading entity's address space receives the contents of the resource. The *write* method enables modification of the information, and the state of the resource itself is changed. Allowing an *execute* method on a resource may change the state of other information resources in the system. Other methods can be modeled in a similar fashion.

We give examples of a user s_1 trying to read file $file_1$ into file $temp_{s_1}$, trying to write $temp_{s_1}$ into $file_1$, and trying to execute $login.sh$, which reads $passwdfile$:

Read

$$\langle\langle\{s_1\}, \{file_1, temp_{s_1}\}, A\rangle, \langle s_1, file_1, open\rangle\rangle \longrightarrow \langle\langle\{s_1\}, \{temp_{s_1} = file_1\}, A\rangle\rangle$$

Write

$$\langle\langle\{s_1\}, \{file_1, temp_{s_1}\}, A\rangle, \langle s_1, file_1, write\rangle\rangle \longrightarrow \langle\langle\{s_1\}, \{file_1' = temp_{s_1}\}, A\rangle\rangle$$

Execute

$$\langle\langle\{s_1\}, \{login.sh, passwdfile\}, A\rangle, \langle s_1, login.sh, execute\rangle\rangle \longrightarrow \langle\langle\{s_1\}, \{passwdfile'\}, A\rangle\rangle$$

The set of policies $P = \{(s, o, r) | s \in S, o \in O, r \in R\}$, describes the set of authorized requests Q_A in the system. The set $\overline{Q_A} = Q - Q_A$ is the set of unauthorized requests.

As mentioned earlier, we observe that confidentiality and integrity policies can be enforced by intercepting and validating actions against the policy tuples, and denying the action if the corresponding policy tuple cannot be found in the system. Therefore policies act as *guards* on the transition between configurations.

Using the abstractions presented so far, we define a secure transition, which is a conditional transition, augmented with a test of policy membership.

DEFINITION 1 (T1). *A transition between two configurations $\langle\langle S, O, A\rangle, \{q\}\rangle \longrightarrow \langle\langle S', O', A'\rangle\rangle$ is called **secure** if request q also $\in P$. Note it may or may not change the state.*

A null transition describes how a subject cannot change the protection state if he or she is not allowed to do so by the policy. Note the protection state in this transition does not change.

DEFINITION 2 (T2). *A **null transition** leaves the system unchanged for request q in $\langle\langle S, O, A\rangle, \{q\}\rangle \longrightarrow \langle\langle S, O, A\rangle\rangle$ if $r \notin P$.*

The state transition between a configuration with an empty or any request set, and a configuration with a request is unconditional and automatically allowed in the system, regardless of whether this transition is followed immediately by a secure or a null transition. Transitions T1 and T2 describe what is called the “policy membership” test of safety for access control models.

An unconditional transition is defined as follows:

DEFINITION 3 (T3). *An **unconditional transition** is the transition $\langle\langle S, O, A\rangle, \phi\rangle \vee \langle\langle S, O, A\rangle\rangle \longrightarrow \langle\langle S, O, A\rangle, \{q\}\rangle$, where $q \in Q$ is any request.*

The set of requests allowed in a system implementation is a subset of the set of policy tuples then our system is *secure*. If these two sets are equal, our system is *precise*.

This formulation of “good behavior” gives us a methodology to validate that a system implements the policies accurately by comparing the set of all observed transitions for each entity with its set of authorized transitions. If the transitions in the system are restricted to T1, T2 and T3,

we can also show that the system only exhibits authorized behavior.

In the next section, we examine this process critically and identify what we think are its major weaknesses. In particular, we claim that this process of policy engineering does not account for compromise. The security guarantees provided by this process are all-or-nothing. We describe how we solve this problem and provide a framework for describing policy compromise and recovery.

3. ENGINEERING FOR COMPROMISE AND RECOVERY

The traditional policy engineering process and the state-transition abstraction described in Section 2 is attractive because of its simplicity. However, it makes one very strong assumption. It assumes that the policy implementation mechanisms *cannot be compromised*. In this paper, we argue that this assumption is a major weakness of existing policy modeling and analysis techniques. They are simply not rich enough to model the notion of survivability. In order to describe the behavior of a system under attack, we argue that we need to explore how things can “go wrong” with respect to the configurations and transitions described in Section 2.

In this section, we show how we can extend the model of Section 2 to include the notion of compromise in Section 3. 1. In Section 3.2 we explore what it means to recover from confidentiality and integrity compromise. Next, we show how we can describe the notion of recovery using availability as a measure of the survivability in Section 3.3.

3.1 Modeling Compromise

In Section 2.3, we define a secure transition between configurations as a transition that is explicitly allowed by the policy specification. This formulation assumes that the policy membership test is always done correctly, thereby ensuring all transitions are authorized by the policy. This behavior is depicted by the event-graph shown in Figure 1.

Figure 1: Event Graph for Access Control Decisions

In real systems, the policy enforcement mechanisms may be compromised by either maliciously by attackers, or exposed as a vulnerability due to poor software engineering. In particular, different types of “faults” or deviations from authorized behavior may manifest themselves. We show this pictorially by augmenting the event-graph with faults to produce a fault-graph in Figure 2.

Figure 2: Fault graph for Access Control Decisions

In particular, the dashed edges in Figure 2 correspond to the transitions that compromise the policy. The first type of compromise corresponds to a denial of service, i.e., a legitimate entity is prevented from completing a legitimate action. However, the information itself is not compromised and its effect on the system is benign. The second type of compromise is malicious and may lead to irretrievable loss or modification of information by unauthorized entities. We model both type of compromise in our analysis.

In particular, the dashed edges in Figure 2 correspond to the transitions that compromise the policy. We claim

that the first type of compromise corresponds to a *denial of service* (DoS), i.e., an authorized entity is prevented from completing a legitimate request. However, the information itself may not be compromised and its effect on the system is usually benign. The second type of insecurity can lead to *information compromise*, and may lead to irretrievable transfer or modification of information by unauthorized entities. Note the distinction between policy compromise and information compromise. Policy compromise can be DoS or may lead to information compromise. We introduce these two types of insecure behavior into our model using the following transitions:

DEFINITION 4 (T4). *An insecure but **benign** transition leaves the system unchanged, even if it is authorized by the policy. It is a transition of the form $\langle\langle S, O, A \rangle, \{q\}\rangle \rightarrow \langle\langle S, O, A \rangle\rangle$ if $q \in P$.*

Note that this transition has no effect on the state variables (S, O, A) .

DEFINITION 5 (T5). *A transition between two configurations $\langle\langle S, O, A \rangle, \{q\}\rangle \rightarrow \langle\langle S', O', A' \rangle\rangle$ is insecure and **malicious** if request q is $\notin P$.*

Note that this transition changes the protection state (S, O, A) , even though it is not authorized by P .

Once policy compromise occurs, it is useful to explore the nature of damage caused by an insecure transition. In the case of failure to a confidentiality policy, the consequences of the compromise can be malicious or benign as described. To elaborate, a confidentiality compromise could be one of the following:

1. A subject $s \in S_C$ that was allowed to read object o according to the system policy P_C , but is denied this action by policy enforcement mechanisms. Information that could be exchanged freely has now become unavailable. However this compromise cannot propagate confidential information further.
2. A subject $s \in \overline{S_C}$ that was not allowed to read object o now suddenly has access to information in o and can further propagate this confidential information, possibly allowing other unauthorized entities in $\overline{S_C}$ access to this information.

An integrity compromise can also be one of two types:

1. A subject $s \in S_I$ that was allowed to write and modify an object o according to the system policy P_I , but is denied this action by policy enforcement mechanisms. As a result, the information may be stale and not useful to other entities in S_I .
2. A subject $s \in \overline{S_I}$ that was not allowed to modify o now suddenly has the ability to change it. Entities in S_I now have to trust an entity in $\overline{S_I}$. Compromised information may further propagate through the system.

In the next subsection, we explore what it means to recover from compromise, specifically, in the case of confidentiality and integrity properties. In Section 3.3, using the lessons learned from this exercise we define a new measure for the ability to recover from compromise.

3.2 Recovery

Recovering from policy compromise involves preventing authorized users from accessing compromised information in the system in the case of a malicious transition, or restoring the availability of a resource that was denied to authorized users in the case of benign insecurity.

In many cases, it may not be possible to prevent an unauthorized entity from obtaining confidential information, but we may be able to prevent its spread. To do this, the system can constrain future behavior of this entity by denying all actions (such as read and write) that can spread the information. The information itself is not compromised and can be used by other entities safely.

When a resource's integrity has been compromised by an unauthorized user, and is detected at a later point in time, the system can rollback actions and restore values to a previously saved uncompromised state of the system. Sometimes it may be necessary to restart the system in order to accomplish this. This recovery action is expensive and its effect is similar to a benign integrity compromise after recovery. Information is stale but not compromised. For applications that rely on freshness of information, this has the effect of making it unavailable. Other strategies such as storing multiple redundant copies, using error correcting codes, etc., can help restore partial integrity.

Therefore, we observe that a benign confidentiality failure impacts the availability of an information resource to a legitimate entity. Similarly, a malicious confidentiality failure, if detected makes confidential information available to unauthorized users. A benign integrity failure impacts the freshness of the information, and recovering from a malicious integrity failure also has the same impact. Furthermore, recovering from integrity or confidentiality compromise may impact the availability of other resources in the system. The common thread through all these compromise and recovery scenarios is how they all impact availability of a resource at the expense of its security.

Based on this observation, we contend that the availability of a resource is a measure of its survivability. We expand on this notion and describe how to specify availability properties next.

3.3 Defining Availability

The term "availability" is overloaded and has many different definitions in different fields of computer science. We define availability as a measure of the survivability of an information protection system against compromise. Compromise may cause unavailability. Recovery may or may not restore availability after compromise. Availability is the ability of a system to provide legitimate users access to a uncompromised resources infinitely often.

The relationship between availability and traditional confidentiality and integrity is explained next.

- **Availability:** Availability is a refinement of integrity and confidentiality. Whether an entity is allowed access to a specific resource and application interface method or not depends on the confidentiality and integrity policies. Given that an entity can access a resource, an availability policy P_A with respect to subset $A \subseteq C \cup I$ of resources, partitions the set of entities E into two sets E_A and $\overline{E_A}$. Entities in E_A are allowed access to specific resources in A infinitely often, as specified in P_A . Entities in $\overline{E_A}$ are given no

guarantees. Note both entities in E_A and $\overline{E_A}$ have qualified access to resources in A , i.e., both E_A and $\overline{E_A} \subseteq E_C \cup E_I$.

Availability is therefore a transient measure of information protection. Notice the definition of insecure transitions allows us to model information compromise and recovery. Without these explicit notion of insecurity, it is not possible to enable a temporal notion of security.

In the next section, we explore how we can define qualitative and quantitative availability properties and develop validation techniques to verify if a specification can make strong availability guarantees. We explore different definitions of availability to model survivability and recovery, and show how we can analyze them in the framework of our model.

4. AVAILABILITY SPECIFICATION AND ANALYSIS

In this section, we formalize the notion of availability, in both qualitative and quantitative terms with respect to the augmented state-transition model of Section 3. In Section 4.1, we define availability as a property of execution traces. Next, in Section 4.2, we show how we can use standard model-checking and other analysis techniques to validate availability properties of traces. We show example specifications which implement model different behaviors and analyze them with respect to their ability to satisfy availability properties.

4.1 Specifying Availability

We define availability on execution traces of resource access. We define a trace as follows:

- A *trace* t_s is an ordered sequence of configuration and transitions associated with a single subject s in our system.

We represent it symbolically with its end-points as $\langle\langle(S, O, A), Q\rangle \rightsquigarrow_{T1-T5} \langle\langle(S', O', A')\rangle\rangle$, using (\rightsquigarrow) to model the sequence of states and transitions of type T1 – T5.

Using this as the basic abstraction of dynamic behavior, we can define availability properties as temporal logic formulas that can be satisfied by traces of system behavior. We use *CTL* [4] to formulate these properties. Since *CTL* is widely used, we do not define its syntax and semantics in detail.

CTL formulas are interpreted over a computation tree of entity behavior. A *computation tree* is formed by designating a state in a state-transition graph as an initial state and then unwinding the structure, by applying all possible transitions in turn, to form an infinite tree with the designated state as root. A model satisfies a *CTL* formula $\mathbf{A}f$, if we can prove f is true “in all computation paths”, it satisfies formula $\mathbf{E}f$ if it is true “in some computation path”. Similarly the temporal operator $\mathbf{G}f$ asserts that the formula f is true in all states of the model in the future, and the formula $\mathbf{F}f$ asserts that some state that can satisfy the property will be reached eventually. In *CTL*, \mathbf{G} and \mathbf{F} must be immediately preceded by the path quantifiers \mathbf{A} or \mathbf{E} . Therefore *CTL* formulas describe properties of states along paths of computation in a model. Our notion of trace is a projection of the path for a given user.

An information resource o , is *survivable* for subject s using privilege r if we can always guarantee the following:

DEFINITION 6. A model of information access behavior is **survivable** if it can satisfy $\mathbf{AG}(\mathbf{EF}(\langle\langle(S, O, A), \{s, o, r\}\rangle\rangle \rightsquigarrow_{T1-T5} \langle\langle(S', O', A')\rangle\rangle))$

A model that satisfies this property can guarantee that it is globally true along all paths in our model that there exists some path (and hence trace) in the future where we can issue a request and expect to get an appropriate response. We claim that this property captures the notion of *survivability*, in the sense that it describes that the model is *capable* of satisfying a user’s request. It may not be able to satisfy it always because of insecurity. However, if a model augmented with insecure behavior can satisfy this property, it means that the system is capable of surviving misbehavior.

Formulating an availability policy in this manner allows us to exploit temporal logic frameworks to validate if this property can be satisfied by a specification. Specifically, we can use standard automated model-checking techniques [4] to verify this property in a formal model of augmented access control. For denial of service, we argue that we need to capture resource consumption behavior at a finer granularity to be able to specify the effect of system load, which is beyond the scope of this paper. We show how we can build a system and validate whether it is survivable in the next subsection. We also explore how to model recovery, and show how it requires stronger guarantees.

4.2 Validating Availability

In this subsection, we illustrate how defining availability as a liveness property, and explicitly modeling insecurity can describe specify as well as validate models that are survivable and have the ability to recover from compromise. We explain this with the help of a simple example. Consider the following system configuration:

$$\begin{aligned} S &= \{John\} \\ O &= \{foo, bar, temp\} \\ A[John, foo] &= read \\ P &= \{\langle John, read, foo \rangle\} \end{aligned}$$

Figure 3: Behavioral Specification for Safety

John’s behavior can be specified using the behavior graph shown in Figure 3(i). John starts initially in secure state $\langle\langle(S, O, A), \phi\rangle\rangle$. (Note ϕ is the same as the empty set $\{\}$). Whenever John issues an action to *read* file *foo*, a transition of type T1 (secure) is enabled because the tuple $\{\langle John, read, foo \rangle\}$ can be found in P , and the variable *temp* is assigned the value of *foo*. If John attempts to read file *bar*, since it is not explicitly allowed, a null transition (type T2) occurs as shown in Figure 3(ii). We simplify our notation to make it more readable. John’s request to read a file *foo* is written as *John.read(foo)*, and the appropriate termination state as *John.response(foo)*. If the state does not change we represent it as *No.change*. Note the two starting states in both Figure 3(i) and (ii) and in other figures in this section are exactly the same state. We represent it as two separate graphs for convenience.

The traditional definition of access control safety to specify John’s desirable behavior, in a computation tree starting in state $\langle(S, O, A), \{\}\rangle$ is as follows:

PROPERTY 1 (ACCESS CONTROL SAFETY I).
 $\mathbf{AG}(\mathbf{John.read}(foo) \rightsquigarrow_{T1-T3} \mathbf{John.response}(foo))$

This property asserts that John is allowed to read file *foo* if the policy allows it, and obtain an appropriate response. Similarly, the rule saying John cannot read file *bar* for which he does not have access to is represented as:

PROPERTY 2 (ACCESS CONTROL SAFETY II).
 $\mathbf{AG}(\mathbf{John.read}(bar) \rightsquigarrow_{T1-T3} \mathbf{No.change})$

These safety properties can be trivially verified against the example specification by observation.

4.2.1 Modeling Compromise

Now suppose that the policy enforcement mechanisms are compromised, either explicitly by an attacker, or inadvertently by poor software engineering. The behavioral model that defines only authorized states and transitions, presented in Figure 3, is not capable of modeling this behavior. The system may enable transitions of type T4 (benign insecurity) or of type T5 (malicious insecurity). The state-transition graph of Figure 3 is augmented with insecure states as shown in Figure 4.

Figure 4: Behavioral Specification with Compromise

In Figure 4(i), in addition to T1, a compromised enforcement mechanism can arbitrarily deny John access to *foo* by executing type T4 transitions repeatedly. In addition, as shown in Figure 4(ii), John may be able to access file *bar*, executing a type T5 transition, even though it is explicitly forbidden by the policy. Expanding the model into a computation tree, we observe that the safety properties described earlier are violated by transitions T4 and T5.

Now we define a survivable version of access control safety as an availability property. For the given system, we claim that the specification is survivable if it can satisfy the following property for benign insecurity:

PROPERTY 3 (ACCESS CONTROL SURVIVABILITY).
 $\mathbf{AG}(\mathbf{EF}(\mathbf{John.read}(foo) \rightsquigarrow_{T1-T5} \mathbf{John.response}(foo)))$

This property asserts that whenever John issues an authorized request to read file *foo*, along all paths of computation, the trace that corresponds to compromise-free behavior will be satisfied some time in the future. Model-checking this property verifies that this model is *capable* of compromise-free behavior. However, *John’s* requests to read the file may be denied repeatedly by transitions of type T4. Furthermore, we show that the model is capable of malicious behavior (transitions T5). The model in Figure 4(ii) also satisfies:

PROPERTY 4 (ACCESS CONTROL VULNERABILITY).
 $\mathbf{AG}(\mathbf{EF}(\mathbf{John.read}(bar) \rightsquigarrow_{T1-T5} \mathbf{John.response}(bar)))$

4.2.2 Recovery

Next, we examine if we can augment Specification 4(ii) to include an explicit notion of recovery. If the information *bar*

is compromised, as shown, we explore how we can change the specification to define recovery. For this we define the symbol *nil* as an empty file. We augment the specification of Figure 4(ii) with a new transition that resets the value of *temp* to *nil* immediately after it is assigned to *bar*, before John can read *temp*. This hypothetical situation is shown in Figure 5.

Figure 5: Behavioral Specification with Recovery

The access control recovery property is specified as follows:

PROPERTY 5 (ACCESS CONTROL RECOVERY I).
 $\mathbf{AG}(\mathbf{AF}(\mathbf{John.read}(bar) \rightsquigarrow_{T1-T5} \mathbf{No.change}))$

This asserts that it is always possible for the model to recover from an insecure transition of type T5. If we can guarantee this modified behavior, then the system can recover from a confidentiality compromise. Model-checking this property shows that our modified specification can satisfy Property 5. A real system may not be able to make such guarantees. In order to show the model is compromise-free, we have to also show how it can prevent transitions of type T4 in Figure 4(i) as follows:

PROPERTY 6 (ACCESS CONTROL RECOVERY II).
 $\mathbf{AG}(\mathbf{AF}(\mathbf{John.read}(foo) \rightsquigarrow_{T1-T5} \mathbf{John.response}(foo)))$

This property cannot be guaranteed in the model, unless we change the behavioral specification. Therefore the specification is still vulnerable to DoS attacks.

So far we have only modeled a system with one entity. More complicated models can exploit standard model-checking techniques since these policy specifications correspond to standard *CTL* formulas, and the behavior can be easily written as a Kripke model [4]. When we have multiple entities and resources, the specification process can get complicated. When a *vulnerability* or the integrity of a resource is compromised, it may be possible to *prevent* other entities from future compromise by temporarily suspending read and write access by other entities, making it unavailable in the process. In [13], we explore how to change policies dynamically, without sacrificing consistency and preserving existing trust relationships. The mechanisms we describe there can be adapted to this specification process seamlessly.

4.2.3 Recovery Analysis

As mentioned earlier, recovering from information compromise may or may not always be possible. It depends mainly on the way information resources are structured in the system. If the policies can partition groups of resources into disjoint sets, with respect to the entities that are allowed/not allowed to read or write to the resource, information compromise can be contained more effectively. We explain this with an example.

Consider a set of subjects $S = \{s1, s2, s3, s4\}$ and a set of objects $O = \{o1, o2, o3\}$. Let the confidentiality policies be $P_{s1} = \{< s1, o1, read >, < s3, o1, read >\}$, $P_{s2} = \{< s2, o2, read >\}$, $P_{s3} = \{< s1, o3, read >, < s3, o3, read >\}$, $P_{s4} = \{< s2, o4, read >\}$. Now we construct a bipartite graph between entities and resources and connecting them with an edge if there is a policy authorizing the entity to

perform the action on the resource. In this example, we find that entities $s1, s3$ and resources $o1, o3$ form a connected component and they are disconnected from the graph formed by $s2$ and $o2, o4$. As a result information compromise in one connected component does not affect the resources or entities in the other. One the other hand, if the entire graph was one connected component, recovery actions are harder.

We can construct a similar graph for integrity policies for write actions. If we superimpose both graphs, we can easily check if there are any flows of information between them. The structure of these relationships between resources and entities plays an important part in recovery analysis. We plan to study this relationship further in the future.

When there are multiple recovery strategies that may lead to different resources becoming unavailable over different lengths of time, often it may be useful to perform a cost-benefit analysis. For example, we may prefer to perform a more expensive strategy to protect a critical resource and make many less critical resources in the system unavailable temporarily. We are working on formalizing this problem and hope to provide more insights in the future.

In addition to malicious behavior, often times, uncompromised resources may become temporarily unavailable to authorized users, because of poor resource engineering or poor design of resource access mechanisms. In addition, malicious users may take advantage of this to overload resource-access mechanisms with unwanted requests, thereby denying service to authorized users of the resource. In order to capture this aspect of policy compromise, viz., *denial of service behavior* we show in [12] we can extend our simple semantic model to include resource consumption behavior on different entities in the system, and define useful properties to capture this problem and explore and analyze different strategies for recovery.

In the next section, we present related work and situate our work in the context of other research in survivability of information systems.

5. RELATED WORK

In this section, we summarize related research with respect to information survivability. Survivability is defined as the ability of a system to continue operation, especially in the presence of accidental failures or malicious attacks [8]. Most of the research in the past has focused on two aspects of survivability : intrusion tolerance and database survivability. Many of these researchers suggest specification techniques for different notions of availability, dependability, reliability and security.

Intrusion tolerance in distributed systems [6] is a well researched area. The area evolved from the study of the relationship between fault-tolerance and security techniques. More recently, Avizienis et al [2] have defined different attributes of dependability that include availability, reliability, safety, confidentiality, integrity and maintainability.

Researchers have focused on different abstractions and mechanisms of specifying intrusions, as an analogy to the fault classification and analysis. Researchers have worked on design of intrusion tolerant applications [15], communication protocols [7] and software infrastructure. The ITUA [5] project describes a middleware architecture along with a suite of intrusion tolerant communication protocols and analysis techniques that exploit adaptation and unpredictability for tolerating the impact of cyber attacks.

Database survivability is also a well researched topic. In [11, 1] the authors present trusted recovery in database systems as an example, to illustrate the principles of trusted recovery in defensive information warfare [14]. This work defines the the notion of malicious and benign transactions and their dependence. They also present techniques for recovering databases from inconsistencies after an attack.

More recently, Jha and Wing [10] propose a systematic method for survivability analysis based on injecting events into a system model and observing its effects in the form of a scenario graph. They provide a general framework to specify different aspects of survivability as both safety and liveness properties and suggest techniques for reliability and cost-benefit analysis.

We believe that none of these researchers focus on the issue of policy engineering of access control properties for survivability. We believe that defining the relationship between availability and traditional confidentiality and integrity properties is our unique contribution in this article.

6. CONCLUSIONS

In this article, we presented a new policy engineering process that extends the traditional policy specification and verification methodology by explicitly modeling insecure states that correspond to policy compromise. We argued that the traditional safety-based policy engineering frameworks are not rich enough to capture the notion of survivability. In its place, we proposed that one needs to rethink confidentiality and integrity in terms of liveness properties, which specify that good things eventually happen.

We defined secure availability as the new measure of survivable security. Using this formulation, we showed how we can introduce the notion of recovery into the policy engineering process. Information protection, and information warfare in particular, can be modeled as maximizing availability under compromise. In order to guarantee availability of critical resources, we explored the notion of recovery from compromise. We observed that the policies themselves may need to change during the lifetime of a system, in a well-defined and controlled manner, in response to perceived threats and exposures to guarantee availability.

We also explored formal specification of qualitative notions of availability using a trace-based characterization of resource-access in a state-transition graph. We described specific properties for an example specification that includes explicit modeling of compromise and recovery, and showed how we can use standard model-checking techniques to verify if a specification can guarantee these properties. With our analysis, we showed that it may not always be possible to recover from information compromise, but changing policies dynamically, and manipulating the availability of different entities, could prolong the lifetime of critical resources.

7. REFERENCES

- [1] P. Ammann, S. Jajodia, C. D. McCollum, and B. Blaustein. Surviving information warfare attacks on databases. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp.164-174, Oakland, CA, May 1997.
- [2] A. Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability. In *LAAS Report*, April 2001.

- [3] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, ISBN 0-201-44099-7, 2002.
- [4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
- [5] M. Cukier, T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal, R. Watro, and J. Gossett. Providing intrusion tolerance with itua. In *Supplement of the 2002 International Conference on Dependable Systems and Networks*, June 2002.
- [6] Y. . Deswarte, L. Blain, and J. C. Fabre. Intrusion tolerance in distributed systems. In *IEEE Symp. on Research in Security and Privacy, Oakland, CA USA*, April 1991.
- [7] B. Dutertre, H. Saïdi, and V. Stavridou. Intrusion-tolerant group management in enclaves. In *International Conference on Dependable Systems and Networks (DSN'01)*, pages 203–212, Göteborg, Sweden, July 2001.
- [8] R. Ellison, D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead. Survivable network systems: An emerging discipline. In *Technical Report CMU/SEI-97-153*, November 1997.
- [9] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. In *Communications of the ACM, Vol 19(8)*, Aug 1976.
- [10] S. Jha and J. Wing. Survivability analysis of networked systems. In *International Conference on Software Engineering (ICSE)*, May 2001.
- [11] P. Liu and S. Jajodia. *Trusted Recovery and Defensive Information Warfare*. Kluwer Academic Publishers, ISBN 0-7923-7572-6, 2002.
- [12] P. Naldurg and R. Campbell. Specification and verification of network denial of service. Work in Progress.
- [13] P. Naldurg and R. Campbell. Dynamic access control: Preserving safety and trust in computer network defense. In *ACM Symposium on Access Control Models and Technologies, Como, Italy*, June 2003.
- [14] B. Panda and J. Giordano. Defensive information warfare. In *Communications of the ACM, Vol. 42, No. 7, p. 31-32,*, July 1999.
- [15] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.