MODELING INSECURITY: ENABLING RECOVERY-ORIENTED SECURITY WITH DYNAMIC POLICIES

BY

PRASAD G. NALDURG

B.E., University of Mysore, 1996
M.S, University of Illinois at Urbana-Champaign, 2000

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

To my mother and sister, in memory of my father.

Our only security is our ability to change.

John Lilly

Abstract

Policy engineering for access-control security has traditionally focused on specification and verification of safety properties ("nothing bad happens"). In most real systems however, resources and access mechanisms are regularly compromised, either maliciously by attackers, or inadvertently due to vulnerabilities caused by poor systems-engineering. I argue that the all-or-nothing nature of assurance provided by safety-engineering cannot describe or reason about systems that are secure and *survivable*—systems that can be engineered to proactively or reactively change their security policies and policy enforcement mechanisms, and thereby continue to provide assurance for critical resources, in spite of compromises and failures.

In this thesis, I present a framework that extends traditional state-transition models of access control security, to describe timing guarantees and stochastic behavior, and show how we can introduce notions of information *compromise*, subsequent *recovery* (whenever possible) and *flexible-response* in a modular fashion. Our framework is also capable of describing insider attacks. I show how we need to focus on liveness properties ("something good eventually happens") to explicitly capture the temporal and dynamic nature of enforceable guarantees required for survivability. I develop a new class of properties expressed as branching-time temporal logic formulas that focus on *secure availability* as a measure of survivability. For finite-state models, the validation of these formulas is decidable in polynomial time using automated model-checking techniques.

To showcase the expressive power of our framework, I apply it to study network Denial of Service (DoS) attacks, and model resilience to such attacks as a survivability property. I show how we can systematically analyze the relative impact of different anti-DoS strategies by changing policies and mechanisms during an attack. Using our automated verification methodology, we formally prove for the first time whether strategies such as selective filtering, strong-authentication, and client-puzzles reduce the vulnerability of an example network to DoS attacks.

Acknowledgments

I take this opportunity to thank all the people who gave me the freedom to indulge in this thesis-writing experience with their generous material and moral support. First and foremost, I thank my advisor Prof. Roy Campbell for his faith in my abilities and for always finding the financial resources to fund my research. Most of the ideas in this thesis have benefited from his insight, and have sharpened over time through our discussions. His critical comments and questions, as well as his boundless enthusiasm for new ideas, has improved the quality of this thesis greatly.

I also thank the members of my committee Prof Dennis Mickunas, Prof. Klara Nahrstedt, Prof. Robin Kravets, and Prof. Jose Meseguer, for their constructive feedback and suggestions that have added great value to my understanding of the subject. I would also like to express my gratitude to my academic advisor Prof. Geneva Belford, who put me on track and encouraged me to pursue my dreams, to Prof. Mike Faiman, for his useful advice during the first three years of my studies at the University of Illinois, and to Prof. Harandi for his support.

To my friends and colleagues at SRG, this thesis has benefited greatly from your support and feedback. I thank Apu Kapadia, Seung Yi, and Jalal Al-Muhtadi for always accommodating requests to read my papers, and for attending my practice talks time and again without complaining. I have also greatly enjoyed my interaction, at various points along the line with Manuel Roman, Fabio Kon, Dulcineia Carvalho, Chris Hess, Chris Andrews, Cristina Abad, Chetan Shivashankar, Cigdem Sengul, Suvda Myagmar, Geetanjali Sampemane, and Brian Ziebart.

To my friends in Champaign-Urbana, too numerous to name individually, I owe you my sanity, and for making life really pleasant and enjoyable in the middle of the cornfields, even in sub-zero temperatures.

I also express my gratitude to Anda Ohlsson, Andrea Whitesell, Bonnie Howard, Barb Cicone, Chuck Thompson, Pat Patterson, Lori Rogers, Erna Amerman, Mary Beth Kelley and the rest of

administrative staff at the Department of Computer Science for all your help and support, and for working behind the scenes and simplifying my life at the department.

To my mother Geetha, I cannot express in words how much I appreciate your unconditional love and support all through the years, and for the sacrifices you made to give me the best opportunities in life. To my sister Shubhashree, younger but infinitely wiser, for all the love and advice, and for being there for me always. To my extended family of innumerable aunts, uncles, cousins, deceased grandparents, my brother-in-law, and other friends and well-wishers, for all your good wishes and for all the caring and sharing.

Finally, to my father who passed away last year, I only wish I could turn back the clock, and share this milestone in my life with you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AL Access List

BLP Bell and La Padula

BST Basic Security Theorem

CL Capability List

CSL Continuous Stochastic Logic

CTL Computation Tree Logic

CTMC Continuous Time Markov Chain

DAC Discretionary Access Control

DoS Denial of Service

DDoS Distributed Denial of Service

DTMC Discrete Time Markov Chain

HRU Harrison, Ruzzo, and Ullman

MAC Mandatory Access Control

MDP Markov Decision Process

ORCON Originator Controlled

PCTL Probabilistic Computation Tree Logic

PNS Probabilistic-Nondeterministic System

RA Response Action

RBAC Role Based Access Control

SPM Schematic Protection Model

TAM Typed Access Matrix Model

TPNS Timed Probabilistic-Nondeterministic System

TSS Timed State Sequences

TTS Timed Transition Systems

In all affairs, it's a healthy thing now and then to hang a question mark on the things you have long taken for granted.

Bertrand Russell

Chapter 1

Introduction

Security engineering for information protection begins with the specification of security policies. An information protection system consists of a set of information resources that can be accessed by different sets of users. Policies are specified within the framework of an abstract system model that describes relevant system entities and their behavior. An information protection policy or a security **policy** specifies what is and what is not allowed by different entities in the system, in terms of their information access behavior and interaction [21, 42].

Policies are enforced by security **mechanisms** that form an integral part of the system implementation. Distinguishing between policy and mechanism allows one to model security requirements at a higher level of abstraction, independent of how they are actually implemented across different platforms.

A system is **secure** if the set of all possible actions allowed by the mechanisms is a subset of the authorized actions described by the policies. The system is **precise** if the set of all actions enabled by the mechanisms is exactly the same as the actions authorized by the policy specification.

The process of describing security requirements, such as confidentiality or integrity properties in an information protection system using appropriate terminology, and analyzing whether the system implements these requirements is called **assurance**.

In this thesis, I argue that existing models of information protection systems cannot model **survivability**—the ability of a system to continue to provide assurance under threat or actual attack. In particular, I investigate how **response actions**(RAs) by trusted users to change security policies and their corresponding enforcement mechanisms, can provide flexible-response against threats and attacks in an information protection system. For example, in the case of a buffer overflow attack,

restricting the access rights of a compromised process can preserve the confidentiality and integrity of sensitive information. I demonstrate how this ability to change policies and mechanisms in a controlled manner can extend the nature and scope of security guarantees for critical information resources.

Historically, security engineers have attempted to validate the design of information protection systems using an abstract mathematical model of dynamic system behavior. These formal models of information protection systems provide security engineers with a framework to analyze what security properties a system can guarantee as it evolves over time. The process of proving that a system model is secure with respect to a security policy (or property), also called policy engineering, can be divided into three inter-related tasks: system modeling, policy specification, and policy verification.

An important point to note here is that proving that an abstract system model can satisfy a security policy specification does not imply that the system is secure. In order to prove that the system is secure, it is necessary to show that the system correctly implements the model. In general, it is difficult to establish this fact for any sufficiently large or complex system, and is outside the scope of this thesis. I focus on the ability to specify security properties in the framework of an abstract model, and verify that this model satisfies the security property of interest. This methodology has exposed several critical design flaws and provided a terminology for describing provably secure systems, and as such plays an important role in security engineering.

In the next few sections of this chapter, I briefly summarize existing models of access control security (Section 1.1), explain what I think is a major limitation of modeling security as safety (Section 1.2), justify why I think it is important to think beyond safety, present a short overview of how I extend this formalism to model and describe survivability, and highlight the important contributions of this thesis (Section 1.3).

1.1 Policy Models and Properties

A system model is an abstract description of the users and resources of an information protection system, and actions that represent information access and modification behavior. Resource access is specified by an **access control** model. Access control is the most commonly used mechanism

to enforce information protection policies in operating systems and networks. Formal models of access control behavior are typically specified as state-transition graphs. A state, also known as a *protection* state, is a snapshot of the sets of subjects, objects, and a data-structure called the access matrix that represents relevant policy assertions in the system at a particular point in time. A transition changes the values of some or all of these sets, in response to an action issued by a subject, and captures the evolution of the system over time.

Security policies model **desirable** properties of this access behavior. The process of policy verification involves validating that the access control mechanisms conform to the behavior described by system policies. In terms of the state-transition graph, the goal is to ensure that all states reachable from a known initial state that is consistent with the policies, through authorized transitions that correspond to mechanisms, are also secure. This reachability condition is what is called a safety property on the graph.

A property is a set of finite or infinite sequences of states [3]. Safety properties specify "nothing bad happens". If a "bad thing" happens in an infinite sequence, then it must do so after a finite prefix and must be irremediable. Thus if a given model (or state-transition graph) does not satisfy a safety property, then there must be some prefix of states and transitions for which no extension to an infinite sequence will satisfy the property. Therefore a property is called a safety property [2] if and only if each execution violating the property has some finite prefix violating that property.

Traditionally, the relationship between policy and mechanism is expressed as the **access control safety property**. This property states that access should be allowed in a system if and only if the corresponding right-to-access is authorized by system policy. Enforcing this property is usually trivial if the protection state does not change over the lifetime of the system. In most system configurations, these access rights can change in authorized and unauthorized ways over time. For example, a user or administrator can choose to delegate or transfer the right to access an object (typically objects that they own) to other users in the system.

Harrison, Ruzzo, and Ullman [67] were the first to formalize this relationship between authorization and rights-transfer behavior for secure access control in their HRU access-matrix model. They discuss this in the framework of a state-transition model where the state represents a snapshot of entities, resources, and policies in the system, and the transitions are actions that can change

the protection state. They use this framework to describe how an access right can "leak", whereby a user can obtain permissions to access an object that he or she is not allowed to access according to the policy. Using this abstraction, they reformulate the access control safety question as follows: "Is there an algorithm that can decide if the transfer of a generic right violates the security policy of an access control system?"

Their results show how verifying this reformulated safety question for access control for a general access control model is undecidable. Denning et al. [40] further prove that the prospect of developing a comprehensive theory of assurance based on this question, or even a finite number of theories is unattainable. These results however do not rule out the existence of restricted models of access control behavior whose assurance is decidable. Following these results, a paradigm shift occurred in the formalism of access control security, and researchers [74, 98, 118] began to look at restricted notions of rights-transfer to ensure that answering the reformulated safety question is efficiently decidable.

In addition to this formalism of access control security, researchers have also looked at security policy models from the view-point of users in the system. The three standard user-oriented security properties are confidentiality, integrity, and availability. Confidentiality properties model information disclosure, and are enforced on read-access by access control mechanisms. Integrity policies model authorized modification of information and are enforced by controlling write-access in a typical system implementation. An availability policy, as the name indicates, models the ability to access information over time, and has not been as well-studied as the other two [42, 21].

The most popular model of user-oriented security policies is the Bell-LaPadula (BLP) model [14] of confidentiality. In this model, each subject or object in the system is assigned a security level. The set of levels forms a lattice-like hierarchy. The right to access an object depends on the security level of both the subject and object. The BLP model specifies rules that have to be enforced by access control mechanisms in order to guarantee security. If the mechanisms can prevent read-access to a higher level, or and write access to a lower-level, then the system can never enter an "unsafe" state. A system is said to be secure under the BLP model if all states reachable from a known safe state are also safe. Biba [20] proposed a dual of the Bell-LaPadula model for integrity, and other rule-based models that constrain access behavior by defining "authorized" transitions in

a similar fashion have been proposed [30, 62, 25].

The BLP model was further refined by McLean [87], who exposed a critical weakness of this formalism by arguing that it can be trivially satisfied by a system that downgrades the security levels of all users and objects, and subsequently allows all accesses. As a result, the "Principle of Tranquility" was introduced to the modeling process, where a user's level is allowed to change only in authorized ways before or after an access request.

An important point to note in all the models summarized so far, is the emphasis on modeling security as safety properties. This property is typically verified by performing a reachability analysis on a state-transition graph created by starting with a safe initial state and expanding it to include all states reachable by authorized transitions that apply at each state. In all these models, assurance is provided by modeling "good behavior" in terms of authorized states and transitions, and asserting "bad things" are **not allowed** to happen. In the next section, I argue that this is a major limitation of existing information assurance models.

1.2 The Need for Recovery-Oriented Security

We motivate the need for a new model of access control security by questioning the assumptions made by existing approaches. The HRU, BLP and other state-of-the-art formalisms model only desirable behavior. In all these models, the relationship between policy and mechanism is implicit. The access control decision is made based on whether the right-to-access can be found in the access matrix. The assumption is that since only authorized users can change this matrix, in accordance to the policy, the system is secure. No checks are made to ensure that the mechanisms are consistent with the system policies at runtime. The policies themselves are not available in the system in any other form.

In real systems, access control mechanisms are routinely compromised. I present two representative examples to illustrate this point: :

- The first example is the **privilege escalation** attack. In most standard operating systems, users are classified into two authorization or privilege classes: *superuser* and *regular*. A regular user can typically only change access rights to objects he or she owns. A super-user

on the other hand is allowed to change any entry in an access-rights matrix.

A privilege escalation attack is when the control of a process running as superuser is hijacked by a regular user, e.g., through a buffer-overflow attack. The attacker subsequently launches other processes, such as an interactive shell, which now run as processes with super-user privileges. The attacker can now assign itself any access-right in the system, thereby compromising the security of the system. However, the policy-enforcement mechanisms in the system are unable to detect this attack, since they only check if these newly updated access-rights are present in the access matrix, and allow the masquerading user to execute "unauthorized" actions. As a result of this attack, all confidentiality and integrity policies in the system may be compromised. The situation where an access matrix may contain insecure access rights is not factored into the design of these enforcement mechanisms, even though this behavior is clearly insecure. Privilege escalation is very common, especially in buffer-overflow attacks, which constitute up to 80% of all reported attacks on computer systems [36].

- Another example where a system mechanism may not always enforce a policy correctly is a resource-exhaustion or denial of service (DoS) attack, where a legitimate user is denied access to an authorized resource by the resource-access mechanisms, either because of system overload, or because of resource-hijacking by malicious users. This unavailability of a resource to a legitimate user, because of actions by unauthorized users, cannot be expressed adequately by existing models of access control.

In the first example presented above, the system enters an insecure state by executing an unauthorized transition. However, the policy enforcement mechanisms are unable to recognize this attack. If this behavior could be modeled, whenever an escalation of privilege is attempted, the system may be able to intervene and restrict the operations the process, or any processes it spawns, can execute in the future. If there is no attack, no restrictions are necessary.

Since runtime monitoring and intervention can be expensive, many Unix systems handle this problem by designing what is called a `chroot jail` [29] that constrains the address space visible to a process running with super-user privileges. The aim of these jails is to restrict the damage caused by a privilege escalation attack. However, an attacker could still succeed in violating some

system policies. Achieving this balance between functionality and choosing appropriate restrictions can become complicated.

In the second example, this behavior temporarily impacts the availability of an uncompromised resource to legitimate users in the system. It may or may not impact the confidentiality or integrity of the information [1]. In recent years, network DoS attacks have become extremely popular with attackers, who attempt to deny legitimate users access to networked information resources. These transient aberrations in policy enforcement behavior are largely ignored by current models of access control security, which lack the expressive power to include fine-grained representation of resource-consumption behavior.

Another motivation for a more expressive model is our claim that the ability to perform an RA and change access control policies to preserve the security of uncompromised entities, in response to a vulnerability exposure is a powerful attack-prevention and damage-containment mechanism [85, 26]. Moreover, it is frequently used in practice (e.g., changing firewall rules) to increase the survivability of the system.

Describing an attack, whether it is a violation of a safety property, or an availability requirement, and modeling the effectiveness of countermeasures (such as RAs) against such attacks cannot be expressed as simple safety properties. A safety property ceases to be true once an unauthorized transition occurs. We cannot describe the effect of a recovery strategy that can restore this property at a future state in the system as a safety property. In Section 1.3, we argue that we need to focus on what are called liveness properties ("something good eventually happens"), to describe survivable systems. A property is a liveness property if **no** partial execution (in terms of states and transitions) is **irremediable**. That is, every finite execution prefix in a model contains at least one continuation where the property can be satisfied eventually.

In the next section, I explain how to extend the traditional state transition access control model, specify policies that can describe survivability properties, and explore methodologies to verify this property within the framework of this model.

[1] If integrity encompasses the notion of freshness of information, this may be viewed as an integrity failure.

1.3 Modeling, Specifying and Verifying Survivability

In Section 1.2, we argue informally how the all-or-nothing nature of safety engineering cannot describe or reason about the ability of a system to pro-actively or reactively change its policies or mechanisms and survive a threat or an attack.

With this as our motivation, we develop a new policy engineering methodology, starting with a more expressive model in Section 1.3.1, and define a new class of policies in Section 1.3.2. We explore the use of automated verification and monitoring techniques for this class of survivability properties in Section 1.3.3, summarize the major contributions of this thesis, and present a road-map in Section 1.3.4.

1.3.1 Modeling Insecurity

In order to develop a theory of **recovery-oriented security** that focuses on the ability of a system to change its mechanisms to survive threats and attacks, I argue that we need to extend the standard state-transition model to include actions that model how an attacker or adversary can influence behavior at a given state in the access control model and cause an **insecure** transition to occur. In the context of the access control model, this undesirable behavior can take one of two forms:

1. An access-request that is *not* authorized by system policy is executed by an attacker, or

2. An access-request that is authorized by system policy cannot be serviced.

The first type of insecure transition models malicious behavior of an adversary who deliberately violates system policy. As a result, the system may evolve to a state where the integrity or confidentiality of information is compromised. The second type of transition models resource-engineering limitations that cause the policy enforcement mechanisms to fail (e.g., in a DoS attack). In both cases, it may be possible to recover from this insecure behavior if e.g., a system administrator intervenes and assumes control over future behavior. Note that it may not always be possible to recover from compromise. The confidentiality or integrity of information may be lost permanently, or the resource may become permanently unavailable. In this case, the system may operate under

weaker guarantees, or parts of the system that are unaffected by the compromise can continue to operate securely.

We formulate our notion of a **recovery strategy**, in the context of a state-transition model of access control augmented with policy assertions. These assertions capture the impact of executing both secure and insecure transitions on a protection state in our model. We use a standard model of state-transition graphs called Kripke structures [33], which are typically used to represent qualitative aspects of temporal system behavior. The need to model time is implicit in our notion of recovery. A state in a Kripke structure is captured by the set of atomic propositions, including policy assertions, that are true in that state. A state evolves when any of these atomic propositions change their value as a result of an action issued by an entity in our system.

Within this framework, a recovery strategy is specified as a state-transition subgraph that starts from an insecure state, evolves through a sequence of secure and insecure states through transitions that model both the attacker's and the system's interactive behavior.

We identify two types of users in this context: **adversaries** and **controllers**. An adversary issues a request that is capable of causing an insecure transition. A controller can change the behavior of different entities in the system using a **response action** (RA), including adversaries (possibly). A recovery strategy is **effective** if the controller can always force the execution of the system to choose a path that ends in a secure state. A recovery strategy is **property-preserving** if it is effective, and we can assert specific qualitative or quantitative properties along its paths.

In many cases, it may not be possible to model an adversary or a controller's behavior in a strategy deterministically. In the case of the buffer-overflow attack, the system cannot know a priori if an attack is being attempted. In the case of a DoS attack, the attacker's behavior can be modeled in terms of inter-arrival times between attack packets. Such probabilistic behavior can be studied by modeling different entities as stochastic processes. Motivated by these examples, we show how to extend our state-transition graphs to represent probabilistic and nondeterministic behavior using a Probabilistic Non-Deterministic System (PNS) [109, 110].

While qualitative and probabilistic statements about the effectiveness of a strategy are important, it is sometimes also useful to model time explicitly to make quantitative statements about the effectiveness of different recovery strategies. I show how we can extend our model carefully to

include timing guarantees using the abstraction of a timed-transition system (TTS) [5] to make the quantitative analysis of such systems tractable. An extension of this representation using a Timed PNS (TPNS) is useful for specifying real time in these models.

A PNS or TPNS can be reduced and analyzed as a purely stochastic discrete-time Markov chain (DTMC), or include continuous time as a continuous-time Markov chain (CTMC). In addition, they can also be analyzed as a Markov Decision Processes (MDPs) when nondeterminism is present. Analysis of such state-transition systems is sufficiently mature and we show how we can adapt these techniques for analysis of survivability properties.

1.3.2 Specifying Survivability Properties

A more expressive model is only the first step in this new theory of recovery-oriented security. In order to express and evaluate the effectiveness of RAs in recovery strategies, we define a new class of security properties called **survivability** properties that explicitly model the ability of a system to recover from information compromise. This formulation is explored in the context of whether the behavior of a critical system service can survive attacks and continue to provide useful service from the viewpoint of legitimate users in the system.

When comparing two different models of resource access and consumption, availability lets us contrast their ability to recover from policy failures or insecure behavior in quantitative terms. The longer the system can operate securely without being subject to integrity or confidentiality compromise or a DoS attack, the longer the resource is available, and therefore usable by legitimate users in the system. However, once a compromise occurs, the shorter the time it takes to restore the system to an authorized state using an RA, i.e., the shorter the recovery time or unavailability, the better the strategy. We can therefore measure survivability by bounding how long the system can remain secure, before "something bad happens," as well as how fast it can recover (if it can), when something bad happens in the system.

Our notion of survivability models the recovery behavior of a system under attack or threat of attack. We are interested in the situation when something bad happens, and want to evaluate if something good can eventually happen. This formulation of survivability is a special type of liveness property. We are specifically interested in the state-transition behavior of strategies that

capture the interaction between adversaries and controllers. Survivability properties specify qualitative and quantitative bounds on recovery. To specify such constraints on recovery strategies, we use well-known branching-time logics, especially CTL [33] (Computation Tree Logic), $PCTL$ or Probabilistic CTL for DTMCs and MDPs, and CSL (Continuous Stochastic Logic) for CTMCs. Survivability properties are expressed as bounded-response properties in this context.

We showcase the applicability of our framework by modeling the network DoS problem using our extended access control model, and show how we can define resilience to DoS as a survivability property. We also show how we can evaluate the effectiveness of different anti-DoS strategies using our model and its accompanying validation techniques.

1.3.3 Verifying Survivability

The next step in survivability engineering is the verification of these properties within these different system models. Specification and verification of different types of temporal logic formulas in a PNS is well-understood [19]. In particular, defining system behavior using the graph-based formalism of Kripke models, specifying safety and liveness properties using temporal logic over traces of computation in this model, and verifying if the model satisfies these properties are all sufficiently mature areas of model-checking research [33]. Therefore, we can leverage these techniques directly and automate the verification process of survivability assurance, relying on existing tools and methodologies. Furthermore, we can integrate future developments in this area into our models seamlessly.

Model checking is useful to evaluate the existence or non-existence of an effective recovery strategy and quantify bounds on recovery times. Automated model-checking is only decidable for finite-state models of concurrent behavior. For large finite-state models, it may become computationally expensive and suffer from what is called the state-space explosion problem. Our approach to describe and evaluate strategies is inherently modular, and reduces the size of the system we are modeling by design. Furthermore, standard techniques such as abstraction, symmetry and composition can reduce this overhead further. In this thesis, I show how we can model the network DoS problem using our formalism to showcase the expressive power of our framework and present a proof-of-concept application of these techniques to show that it is feasible.

A complementary technique that can be used to evaluate the properties of different strategies that may not be finite-state is run-time verification. Run-time verification techniques define monitors that can observe finite traces of system behavior and evaluate what properties are satisfied by these traces. Even if the behavior of a system cannot be described using finite-state semantics, the observable behavior of a system can be modeled as a finite set of traces. Evaluating temporal logic formulas over finite traces for safety properties is well studied, and is infeasible for generic liveness properties. Recent results [79] show how a restricted version of liveness called "bounded availability" can be monitored on execution traces.

An important point to note here is that given an abstract model of an existing system, we may not always be able to guarantee recovery. Our analysis may expose the weaknesses in the model that make it impossible to provide such assurances. In this case, a system-designer may be able to choose which resources to isolate from the rest of the system, in order to keep a smaller, but more critical subsystem immune to the threat of attack. Because of dependencies and trust relationships, large parts of the system may need to be disabled to keep it survivable. Our survivability engineering framework can help the designer make these choices.

1.3.4 Summary of Contributions

Exploring how access control mechanisms and resource-consumption behavior can be made survivable, extending the traditional state-transition graph with explicit insecure states and transitions, including time and stochastic behavior, modeling the notion of recovery from compromise, specifying and analyzing survivability as safety and liveness properties of subgraphs of system behavior using automated verification and run-time monitoring techniques, formalizing the network DoS problem as a survivability property, and analyzing the relative costs and benefits of different DoS prevention strategies, are all original contributions of this thesis.

The rest of the thesis is organized as follows: In Chapter 2 of this thesis I present a detailed discussion of existing models of access control security, as well as different types of security properties. In Chapter 3, I define my thesis problem, situate it in the context of background research, and describe my extended behavioral model of access control. With the help of this model, I describe how to specify survivability, recovery, and DoS-free behavior as availability properties and

study recovery using dynamic access control in Chapter 4. I explore the DoS problem and describe formalisms to specify and verify DoS resistance and resilience properties in Chapter 5. Chapter 6 presents related research, including work on formal models of DoS prevention, fault-tolerance, survivability, availability, dynamic access control, and DoS attacks and prevention strategies. I present my conclusions in Chapter 7, with a summary of contributions, lessons learned, and future work.

A theory has only the alternative of being right or wrong. A model has a third possibility: it may be right, but irrelevant.

Jagdish Mehra, The Physicist's Conception of Nature, 1973

Chapter 2

Background

In this chapter, I present a summary of background research in the context of security policy terminology, formal specification, and policy verification. This chapter is a description of the state-of-the-art with respect to modeling of information protection policies, and motivates the problem I describe in Chapter 3.

In the next few sections of this chapter, I describe how information protection systems are specified formally as state-transition models, and how the notion of information security is expressed as safety properties ("nothing bad happens") that can be verified within the framework of these models. I also highlight several fundamental results with respect to what types of properties can be verified within the framework of these models.

I begin this chapter with a description of access matrix models in Section 2.1 that forms the basis for the formalism of the policy engineering process.

2.1 Access Matrix Models

The principal mechanism for information protection in most systems is *access control*. If implemented correctly, access control can ensure that all accesses to resources are authorized by the system policy, thereby preventing inadvertent or malicious information exposure. An information protection system is typically described using an *access matrix* model. An access matrix describes the set of authorized access rights in the system. An access right is a relation between a subject, object and a privilege. For example, an access right of the form $\langle user_U, file_F, read \rangle$ specifies that subject $user_U$ has the right to *read* file object $file_F$.

The access matrix model was first formulated by Lampson [82] in the context of operating systems and refined by Graham and Denning [61, 43]. A similar abstraction, developed independently and called a security matrix, was introduced by Conway et al. [34] in the context of databases.

The effectiveness of access control depends on two important pre-conditions [42]: proper user identification using appropriate authentication mechanisms, and protection of access rights from unauthorized modification. The first condition, the ability to prevent impersonation of subjects is crucial to secure information access. In addition, the safety of any access control system rests on the ability of the system to restrict *who* can access and modify, add, or delete access rights.

In 1976, Harrison, Ruzzo, and Ullman developed a formal version of the access matrix model [67] as a state-transition graph that captures the dynamic behavior of protection systems. This model is commonly referred to as the HRU model, and defines access control as a safety analysis [21] problem. The states in an HRU model correspond to a snapshot of the active entities, information resources, and authorizations in the system represented by the access matrix, and the transitions are actions (such as add entity, delete right etc,) that can change the access matrix. Different security policies are defined as safety properties that can be validated by expanding the state-transition graph to model the dynamic behavior of the system. Using this model, Harrison et al. study the feasibility of proving properties about a high-level abstract model of a protection system, as the system evolves over time.

With the help of their model, they prove that in the most general abstract case, the security of computer systems, defined as safety properties in the framework of their model, is **undecidable**. They show that the prospect of developing a comprehensive theory of information protection that is general enough to provide automatic proofs or disproofs of safety is unattainable [42].

To put this result in perspective, it states that there are fundamental limitations on our ability to prove properties about a generic abstract model of a protection system. Systems without severe restrictions in their operations will have security questions that are too expensive to answer. As a consequence of this result, researchers have shifted their focus from looking for a general theory of safe systems, to the construction of specific protection system models that can be shown to be provably secure [74, 98, 118]. These models explore the nature and scope of practical restrictions that need to be imposed on the models to make safety questions tractable.

I describe the HRU model in Sections 2.1.1 and 2.1.2, and highlight the important results and their consequences with respect to the design of abstract models of secure systems in 2.1.3 and 2.1.4.

2.1.1 Protection State

In the HRU model, the protection state of a system is defined by a triple (S, O, A) where:

1. S is the set of subjects, or active entities in the model.
2. O is the set of objects or protected entities in the system. Each object has a unique name. Subjects are also considered to be objects; thus $S \subseteq O$.
3. A is an access matrix, with rows corresponding to subjects and columns to objects. An entry $A[s, o]$ lists the access rights of subject s over object o.

In an operating system, subjects are typically processes, users, or domains. Objects can be files, processes that represent services, or other resources. Access rights specify different kinds of accesses that may be performed on different objects, including *read*, *write*, *execute* and *append*. Special rights include the **own** right that represents ownership information, as well as the **copy** right that allows its possessor to grant rights to another subject or set of subjects.

In the next subsection, I describe the set of transitions of the HRU model that capture the evolution of the protection state over time.

2.1.2 State Transitions

As processes execute in the system, the protection state of the system may change over time, in response to actions initiated by subjects. This change in state is captured by **commands** in the HRU model. The HRU model assumes that there is a **reference monitor** associated with every object that implements the policy enforcement mechanisms and controls access to the object. This concept was introduced by Graham and Denning [61]. A monitor for object o prevents a subject s from accessing o if $A[s, o]$ does not contain the required right.

The HRU model identifies six **primitive operations** that can change the protection state of a system. These correspond to the number of ways subjects, objects, and rights may be added or

op	conditions	new state
enter r into $A[s, o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s, o] = A[s, o] \cup \{r\}$ $A'[s_1, o_1] = A[s_1, o_1]\ \forall (s_1, o_1) \neq (s, o)$
delete r from $A[s, o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s, o] = A[s, o] - \{r\}$ $A'[s_1, o_1] = A[s_1, o_1]\ \forall (s_1, o_1) \neq (s, o)$
create subject s'	$s' \notin O$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $A'[s, o] = A[s, o], s \in S, o \in O$ $A'[s', o] = \phi, o \in O'$ $A'[s, s'] = \phi, s \in S'$
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $A'[s, o] = A[s, o], s \in S, o \in O$ $A'[s, o'] = \phi, s \in S'$
destroy subject s'	$s' \in S$	$S' = S - \{s'\}$ $O' = O - \{s'\}$ $A'[s, o] = A[s, o], s \in S', o \in O'$
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O - \{o'\}$ $A'[s, o] = A[s, o], s \in S', o \in O'$

Table 2.1: Primitive Operations

deleted from the system. The conditions required to execute these commands and their effect on the protection state are summarized in Table 2.1.

Executing the primitive operator *op* in system state $Q = (S, O, A)$ causes a **transition** to state $Q' = (S', O', A')$, written as $Q \models_{op} Q'$ under the conditions shown in Table 2.1. A command in the HRU model consists of a possible condition followed by one or more primitive operations written as follows:

command $c(x_1, \cdots, x_k)$

if $r_1 \in A[x_{s_1}, x_{o_1}]$ and

$r_2 \in A[x_{s_2}, x_{o_2}]$ and

$\cdots$

$r_m \in A[x_{s_m}, x_{o_m}]$

then

op_1;

op_2;

$\cdots$

op_n

end.

Here $r_1, \cdots, r_m$ are rights ($m \geq 0$), $s_1, \cdots, s_m$ and $o_1, \cdots, o_m$ are integers between 1 and k.

The effect of a command $c(a_1, \cdots, a_k)$ with actual parameters $a_1, \cdots, a_k$, on a state Q, yields state $Q \models_{c(a_1,\cdots,a_k)} Q'$ as follows:

1. $Q' = Q$ if any one of the conditions of c is not satisfied, and

2. $Q' = Q_n$ otherwise, and there exist states $Q_0, Q_1, \cdots, Q_n$ such that:
   $(Q = Q_0) \models_{op_1} Q_1 \models_{op_2} \cdots \models_{op_n} Q_n$,
   which denotes the action of executing primitive operation op_i after substituting the formal parameters x_i with the actual parameters a_i.

The HRU model provides the basic abstractions, in terms of primitive operations, which can be combined together using conditional commands to describe the access control behavior of a protection system. For example (adapted from Bishop [21]), a process p creating a file f with own, read r, and write w permissions in a UNIX system can be specified as follows:

command $create\_file(p, f)$

create object f;

enter own **into** $A[p, f]$;

enter r **into** $A[p, f]$;

enter w **into** $A[p, f]$;

end

The next example shows how one can specify preconditions on primitive operations. In this specification, a process p can give another process q the right r to a file f only if it owns f. This is written as follows:

command $allow\_transfer\_right(p, q, f, r)$
if own **in** $A[p,\ f]$
then
  enter r **into** $A[q,\ f]$;
end.

The above examples demonstrate the expressive power of the HRU model.

In Section 2.1.3 next, I describe the safety question, i.e., under what conditions does the abstract model presented above implement a generic algorithm to automatically determine if the system is secure.

2.1.3 Safety Question

Given an abstract model of a particular access control system, the question that has interested researchers is the following [21], "Is there a generic algorithm that will allow us determine or prove if this model is secure?". The quest for the answer to this question has driven policy engineering research for the past two decades. In order to answer this question, one first needs to define what is meant by the term "secure" precisely.

The configuration of an access matrix describes what subjects can do, not necessarily what they are authorized to do. Within the framework of the operations in the HRU model, protection policies or security policies divide the states and transitions of a model into two types: **authorized** and **unauthorized**. Authorized states correspond to those states explicitly allowed by the specification. By default, all other states are unauthorized. Transitions are only allowed between authorized states.

Access control mechanisms can enforce the system's security policies by ensuring that the physical states of the system correspond to the authorized states of the abstract model. This is achieved by ensuring that a system is never allowed to initialize in an unauthorized state and unauthorized

actions from authorized states are restricted by resource access mechanisms (e.g., interceptors, monitors, etc.) that also enforce the policy. These mechanisms 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.

A policy in the context of a protection system specifies whether a particular set of subjects can access a specific set of objects, and what actions (rights) the entities are authorized to perform, either individually or as a group, on this set of resources.

I now present an example of a protection policy using the notation from the previous section. A simple Discretionary Access Control (DAC) Security Policy can be specified formally as follows [42]:

Definition 2.1.1 (DAC Policy). *Let $Q = (S, O, A)$ be an authorized state such that* **own** $\in A[p, f]$ *for subject p and file f, but $r \notin A[q, f]$ for subject q and right r. Let $Q' = (S', O', A')$ be a state such that $Q \models_c Q'$ and $r \in A'[q, f]$. Then Q' is authorized under the DAC policy if and only if $c = allow\_transfer\_right(p, q, f, r)$.*

This policy states that only owners of objects are allowed to transfer rights to other subjects.

For the access control model described in Sections 2.1.1 and 2.1.2, Harrison et al. studied the feasibility of proving properties about the security of an abstract system model. They define unauthorized behavior using the notion of a **leaked** right, and a **safe** state or authorized state as follows:

Definition 2.1.2 (Leaked Right). *A right r is* **leaked** *by a command c that when run in a state Q, executes a sequence of primitive operations and enters right r into some cell of A not previously containing r.*

Definition 2.1.3 (Safe State). *Given a system, an initial state Q_0, and a right r, we say that Q_0 is* **safe** *for r if there is no sequence of system requests that, when executed starting in state Q_0, will write r into a cell of the access matrix that did not already contain it.*

In many systems, leaking rights is allowed by the policies. For example, the DAC policy specified above intentionally allows the owner of an object transfer rights to other subjects. This type of transfer, specifically allowed by the policy is called **authorized transfer**. Therefore the system

may be implementing the policy correctly, but the initial state of a system may not be safe with respect to all rights r. The security verification question therefore is whether the transfer of a right r violates the protection policies of the system. This is typically expressed as follows:

Definition 2.1.4 (Safety Question). *Is there any algorithm that can decide if the transfer of any generic right r (or leaking) violates the security policies of a given protection system whose initial state is Q_0?*

In Section 2.1.4, I summarize the results of the safety question for the HRU model and show how it has impacted the design of safe systems.

2.1.4 Safety Analysis

Harrison et al. showed that access control safety, as defined in the previous subsection, is undecidable for an arbitrary protection system. They showed this by encoding the behavior of a Turing machine, such that the leakage of a right corresponds to the Turing machine entering its final state. If the safety question is decidable, then so is the halting problem. However, since the halting problem is undecidable, the safety problem is also undecidable.

Denning, Denning, Garland, Harrison, and Ruzzo [40] further proved that the prospect of developing a comprehensive theory of protection, or even a finite number of theories is unattainable, since this set is not recursively enumerable. Since the set of all safe theories is not recursively enumerable, it cannot be recursively axiomatizable, and therefore, systems for proving safety are necessarily **incomplete**.

While these fundamental results appear discouraging, they do not exclude the possibility of generating smaller classes of safe systems with more constrained behavior that are decidable.

In the same article, Harrison et al. show safety is decidable if no new subjects or objects can be created, and that it is PSPACE complete. They also explore the decidability question for a special class of systems called **mono-operational** systems. A mono-operational system is one where each command performs a single primitive operation. For this class of systems, once again, they show that safety is decidable. Mono-operational systems are too weak to express many policies of interest [94]. They cannot express policies that give subjects special rights to objects they create (for example creating a child process). Harrison et al. also proved that the safety problem is decidable

for systems that are both *monotonic* (no destroy and delete) and *monoconditional* (only one clause in the condition). Such systems are still very limited as far as expressing useful properties.

As an important consequence of these results, much of the research into theory of safe systems shifted from proving if arbitrary security systems are safe, to designing systems that were secure from first principles. To keep the analysis of leaking rights simple and tractable, most practical models (such as Unix) severely restrict a subject's right to grant privileges to other subjects, and forbid further delegation of these rights.

Examples of protection systems that have decidable theories for the safety question include the graph-based formalism of the Take-Grant model [74] introduced by Jones, Lipton and Snyder. Vertices in the graph correspond to subjects and objects. Edges are directed and encode whether rights can be taken or granted between the entities. A set of graph-rewriting rules codifies how the Take-Grant graph can evolve over time.

Jones et al. show that the safety question is decidable for the Take-Grant model, even if the number of subjects and objects that can be created is unbounded. In addition, it is decidable in time linear to the size of the initial state. An important point to note here is that the Take-Grant model only describes the transfer of authority in the system, and does not describe the protection state, thus abstracting only the information needed to answer the safety question. Lipton and Snyder have also shown that the safety question for systems [83] with a finite set of subjects is decidable, but computationally intractable.

The Schematic Protection Model (SPM) by Sandhu [118] is closer in spirit to the HRU model and uses the abstraction of security types. A type acts as a generic class label for an entity. Rights can be of two kinds: inert rights and control (such as take and grant) rights. Manipulation of rights is controlled by two relationships: link predicates and filter functions. Link predicates capture the relationships between subjects with regard to transfer of rights, and the filter functions impose conditions on when transfers can occur. Sandhu shows that this model has a decidable subset that is more expressive than the Take-Grant model. Amman and Sandhu [7] extend this work and describe a model that is formally equivalent to a monotonic HRU, but with decidable safety.

More recently, Sandhu describes the Typed Access Matrix model (TAM) [117], which introduces strong typing to the HRU model. The type of an entity is fixed when it is created and remains

fixed throughout the lifetime of the model. The TAM operations are the same as HRU operations, except that the **create** operations are augmented with types. Sandhu shows that a monotonic TAM model, like monotonic HRU, is undecidable. However if one avoids cyclic creates, safety is decidable. If all monotonic TAM commands are limited to three parameters, the resulting model becomes decidable in polynomial time. This model is the current state-of-the art for generalized access control policies [21, 94].

The HRU analysis and the results of safety analysis of other models discussed in this section show how it is hard to analyze propagation of access rights, even if we have complete knowledge of the mechanisms of propagation. However, the notion of policy in these generalized access control models does not directly capture the primary security concerns that are of interest to most users of an information protection system, in terms of what resources are visible to what users and how they can be modified. In the next section, I show how the three standard properties, viz., confidentiality, integrity, and availability, directly address how information access rights and mechanisms affect users in the system.

2.2 Information Protection Policies

Traditionally, information protection is defined in terms of *confidentiality* and *integrity* policies within the framework of an abstract state-transition model of a protection system. *Information Flow* security is another desirable property and is intrinsically related to confidentiality. *Availability* is also often mentioned as a desirable information security property but is not as rigorously studied as the other two and there are no formal definitions of availability policies in this framework. A major part of this thesis is dedicated to the study of availability policies, as a measure of survivability of confidentiality and integrity properties. Therefore, as background information, this section presents a brief overview of existing models of integrity and confidentiality policies, and I defer discussion of existing models of availability properties to Chapter 6.

In Section 2.2.1, I present standard definitions of confidentiality and integrity policies from the viewpoint of a set of subjects and their ability to access and modify information resources. In Section 2.2.2 and Section 2.2.4, I describe confidentiality and integrity policies, and follow it up with a discussion on information flow in Section 2.2.3.

2.2.1 Basic Definitions

Confidentiality deals with concealing information, and preventing unauthorized access to a resource. Integrity refers to preventing unauthorized modification. They are formally defined as follows (adapted from [21]):

- **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 $\overline{S_C}$. Subjects in $\overline{S_C}$ 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 $A[s', o], \forall s' \in \overline{S_C}$. 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 $I \subseteq O$ of objects partitions the set of subjects S into two sets S_I and $\overline{S_I}$. Subjects in $\overline{S_I}$ 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.

From the description of the access matrix state-transition model, and the definition of different security policies, one observes that confidentiality and integrity policies can be enforced by intercepting and validating actions against the policies, and denying the action if the corresponding policy cannot be found in the system. For example, confidentiality applies to read-permissions, and integrity to write-permissions in a filesystem. Therefore policies act as *guards* on the transitions between protection-state configurations.

The access matrix A should at all times only contain rights that are specifically authorized by these policies. Furthermore, access should be allowed if and only if the access right can be found in the system. This property is called the access control safety property. Let R be the set of rights in the system, and $P = \{\langle s, o, r\rangle | s \in S, o \in O, r \in R\}$ be the set of confidentiality and integrity policies. The operator $\Box$ is the standard "henceforth" operator from temporal logic and is useful to

describe invariants that have to be satisfied by states and transitions in the model. Access control safety in the framework of the access matrix model is specified as follows:

Definition 2.2.1 (Access Control Safety).

$\Box((r \in A[s,\, o] \leftrightarrow \langle s, o, r\rangle \in P) \wedge (allow\_access(s, o, r) \leftrightarrow r \in A[s,\, o]))$

In the next subsection, I describe the Bell-La Padula model for confidentiality. This model implements what is called the military-style classification based scheme for ensuring confidentiality in information protection systems.

2.2.2 Modeling Confidentiality

One of the primary security concerns of users in an information protection system is that they are often unaware of what a program acting on their behalf is doing [94]. Users have to trust that programs written by other users, but executing on their behalf, do not transfer or distribute their discretionary rights clandestinely, in addition to executing their legitimate functions. Such seemingly innocuous but malicious programs are known as *Trojan Horses*.

In high-assurance environments, such as military applications, the DAC policy model that allows users to pass rights to other users without constraints, is considered unsuitable. The Mandatory Access Control model (or the MAC) model was therefore proposed to introduce constraints on how access rights can be transferred. In a MAC system, transferring of rights is governed by system policy, administered by a security officer, and is no longer under control of users. The best known example of a MAC policy is the multi-level security model used in the military with its lattice of security levels that range from top-secret to unclassified. Rights to read a top-secret file, for example, cannot be transferred to any user in a lower level by any mechanism in the system.

In 1975, Bell and La Padula formalized the multilevel MAC security model using a notation similar to the HRU model. This model is popularly referred to as the BLP model [14]. Like the HRU model, it employs subjects, objects, rights, and an access matrix, but there are several important differences between the two models. The sets S and O do not change from state to state, and the set A contains only four rights: read, write, execute and append. For simplicity, we focus on only read and write, since the other two rights do not affect the discussion that follows. In addition to these abstractions, the BLP model also introduces a lattice of security levels defined by the

partial-order L of labels, and the set of categories C that add attributes to a security classification, making it easy to implement the "need-to-know" principle [42]. The function $F : S \cup O \rightarrow L \times C$ yields the security level of a given subject or object.

A state in $v \in V$ in the BLP model is the tuple (F, A). A system in the BLP model consists of an initial state v_0, a set of requests $\mathcal{R} = \{\langle s, o, r\rangle | s \in S, o \in O, r \in R\}$, and a transition function $T : V \times R \rightarrow V$. T transforms the system from one state to another when the request is executed (i.e., A or F change). The necessary and sufficient criteria for a system to be secure in the BLP are given next:

- **Simple Security Property**: A state v is secure with respect to the simple security property if and only if for every $s \in S$ and $o \in O$, $read \in A[s, o] \rightarrow F(s)\, dominates\, F(o)$. This is also called the "No-Read-Up" rule.

- ***-Property**: A state v is secure with respect the $*$-property if an only if for every $s \in S$ and $o \in O$, $write \in A[s, o] \rightarrow F(o)\, dominates\, F(s)$. This is also called the "No-Write-Down" rule.

The relation *dominates* is defined as follows: Security Level (L, C) dominates the security level (L', C') if and only if $L' \leq L$ and $C' \subseteq C$.

The Simple Security property prevents a low-level user from gaining read access to higher level objects (such as files). The *-property prevents a high-level user (or Trojan horse run by a high-level user) from copying contents of high-level objects to low-level objects so that low-level users can gain unauthorized access. Security in the BLP model is defined as follows:

Definition 2.2.2 (Bell-La Padula Security). *A state in the BLP model is* **state-secure** *if and only if it is secure with respect to the Simple Security Property and *-Property. A system (v_0, R, T) is secure if and only if (i) v_0 is state-secure, and (ii) every state* **reachable** *from v_0 by executing a finite sequence of one or more requests from $\mathcal{R}$ is state-secure.*

Bell and La Padula propose and prove the following theorem:

Theorem 1 (Basic Security Theorem(BST)). *: A system (v_0, R, T) is BLP secure according to Definition 2.2.2, for each $s \in S$ and $o \in O$, for all transitions between v and v', $T(v, r) = v'$ for $r \in R$, where $v = (F, A)$ and $v' = (F', A')$, if the following rules about transitions can be enforced:*

- *if* $read \in A'[s, o]$ *and* $read \notin A[s, o]$ *then* $F'(s)$ *dominates* $F'(o)$*;*
- *if* $read \in A[s, o]$ *and* $F'(s)$ *does* **not dominate** $F'(o)$*, then* $read \notin A'[s, o]$*;*
- *if* $write \in A'[s, o]$ *and* $write \notin A[s, o]$ *then* $F'(o)$ *dominates* $F'(s)$*;*
- *if* $write \in A[s, o]$ *and* $F'(o)$ *does* **not dominate** $F'(s)$*, then* $write \notin A'[s, o]$*;*

The proof follows from structural induction as these four rules capture what transitions are allowed for a system that is BLP secure. The class of systems described by the BLP security definition is the same as the class of systems that can be derived from expanding the states and transitions according to the rules specified in BST. The BLP model was the first to capture this notion of correspondence between policy and mechanism.

Subsequently, McLean [87] exposed a problem with proving a system secure using a BLP-like formalism, stating that the interpretation of a valid transition is transparent to the definition of a secure state. It is not enough to ensure that every state reachable from a secure state is secure. One needs to show that the manner in which this state is reached is also "secure". This is best explained with the argument presented next.

Consider a system Z whose initial state is state-secure and has only one type of transition. When a subject s requests any type of access to object o, all subjects and objects are downgraded to the lowest security level and access is granted. Now System Z still obeys all transition rules specified by BST. However, it is not "secure" in any meaningful sense [92].

The main problem with the BLP model is that does not have any restrictions on changing the protection state, especially the security levels of users and objects, before or after a transition. To rectify this problem, McLean defines a framework of security models, which places restrictions [92] on the transactions that prevent anomalies like System Z from passing as secure models. This restriction is called the "Principle of Tranquility". McLean's framework is a quadruple (S, O, A, L) where the elements are the same as those defined in the BLP model. A model within this framework is a set of state machines of the form (F, A), as in the case of the BLP model.

The framework however contains a new function $K : S \cup O \rightarrow 2^S$, which maps each subject or object in the system with the set of subjects that are allowed to change its security level. As before, the system consists of an initial state v_0, a set of requests $\mathcal{R}$ and a modified transition function

$T : (S \times V \times \mathcal{R}) \rightarrow V$, which yields a new state for a subject executing a request in the current state. The new definition of a secure system is given as follows:

Definition 2.2.3 (McLean's BST). *A transition function T between two states $v = (F, A)$ and $v' = (F', A')$ and $T(s, v, r) = v'$ is transition secure if and only if $\forall x \in S \cup O$ if $F(x) \neq F'(x)$ then $s \in K(x)$. A system (v_0, R, T) is secure only if (i) v_0 and all states reachable from v_0 by a finite sequence of one or more requests from R are (BLP) state-secure and (ii) T is transition-secure.*

This framework forms a boolean algebra of models whose most restrictive element is a BLP model where no security levels can change, and whose top element is a BLP model with no restrictions on security level changes whatsoever. The two frameworks presented in this section are the current state of the art for mandatory access control.

2.2.3 Information Flow

Access control policies and mechanisms (especially reference monitors) are convenient abstractions to model information security and can provide a high degree of assurance. The access matrix and the BLP models have played an important role in the design of secure systems. However, the models presented so far are not rich enough to capture what is known as the *covert channel* problem of access control.

This problem arises from the difficulty of mapping an access control model's primitives to individual objects, subjects, and mechanisms in a computer system implementation. For example, consider the response of a reference monitor for a low-level subject that is trying to write to a non-existent high-level file object. If the subject is notified about the mistake, a high-level Trojan Horse can use this channel to communicate one bit of information by creating the file whenever the bit it wants to transmit is one, and delete it when it wants to signal a zero. If however, the subject is not informed of this mistake, or if a dummy file is created, legitimate typing errors on behalf of the subject will be punished unnecessarily.

The problem exists because the BLP model or McLean's model do not treat the existence of a file as information that needs to be protected. Modeling and detecting covert channels is extremely difficult and among other things, involves tracing the information-flow paths of programs [41, 42], checking programs for shared resources [113], checking for clock asynchrony to prevent timing

channels [131], and using type-checking [128, 114] and other language based schemes to analyze information flow. Many of these techniques are intractable for large systems (especially distributed systems) and can only be applied after the systems are built, and making system changes at this stage may be prohibitively expensive [12].

Rather than analyze access control models post-mortem for information flow problems, researchers have explored the design of systems that are immune to this type of information exposure, from first principles. Interface models of confidentiality explore what restrictions on a system's input/output relation are sufficient for preventing undesirable flow of information. The most popular of these models is **Noninterference**, originally formulated by Goguen and Meseguer [59].

Goguen and Meseguer view the system as a state machine consisting of the set S of subjects, a set Σ of states, a set O of outputs, and set Z of commands. $C \subseteq S \times Z$ is the set of state transition commands, corresponding to the notion of a subject issuing a command. A state transition function $T : C \times \Sigma \rightarrow \Sigma$ describes the effect of executing state-transition command z when the system is in state σ, and an output function $P : C \times \Sigma \rightarrow O$ describes the output of executing command z in state σ. Initially the system is in state σ_0. As the system evolves, the outputs provide a record of the system's functioning.

Next, Goguen and Meseguer define two functions projection *proj* and purge (π) as follows:

Definition 2.2.4 (Projection Function). *Let $T^*(c_s, \sigma_i)$ be a sequence of state transitions for a system, and $P^*(c_s, \sigma_i)$ be the corresponding outputs. Then $proj(s, c_s, \sigma_i)$ is the set of outputs in $P^*(c_s, \sigma_i)$ that subject s is authorized to see, in the same order as these outputs appear in $P^*(c_s, \sigma_i)$.*

Definition 2.2.5 (Purge Function π). *Let $G \subseteq S$ be a group of subjects and let $A \subseteq Z$ be a set of commands. Then $\pi_G(c_s)$ is the subsequence of c_s obtained by deleting all elements (s, z) in c_s. $\pi_{G,A}(c_s)$ is the subsequence of c_s obtained by deleting all elements (s, z) in c_s such that both $s \in G$ and $z \in A$.*

The purge function captures the effect of making certain command executions invisible to some users. If the set of outputs any user can see in the system is the same as the set the user can see when the command history is purged of inputs that another user generated, the system is secure under noninterference. This is defined as follows:

Definition 2.2.6 (Noninterference). *Let $G, G' \subseteq S$ be distinct groups of subjects and let $A \subseteq Z$ be a set of commands. Users in G executing commands in A are noninterfering with users in G', written as $A, G : | G'$ if and only if, for all sequences c_s with elements in C^*, and for all $s \in G'$, $proj(s, c_s, \sigma_i) = proj(s, \pi_{G,A}(c_s), \sigma_i)$.*

In order to verify a system satisfies noninterference, Goguen and Meseguer develop a set of conditions (output consistent, transition consistent and locally respects) and prove that they are sufficient for establishing noninterference in state machines [60]. This result is called the Unwinding Theorem.

The Goguen-Meseguer model is defined in the framework of a deterministic state-machine model of the system. McLean [93] shows how one can relax this requirement and can prove noninterference directly using trace-based semantics and functional correctness.

Comparing BLP and noninterference is difficult, as the primitives of BLP lack a precise semantics [91]. However it is noted that in general BLP is weaker than noninterference as it allows covert channels. Surprisingly, noninterference is weaker than BLP, as it can allow a low-level user copy a high-level object to another high-level object, which would be disallowed as a read by the BLP model.

The interface approach for confidentiality presented so far, is relatively straightforward with respect to deterministic systems, but is difficult to analyze when it is extended to non-deterministic systems.

In 1986, Sutherland introduced a new property called nondeducibility [126], which states that a system is nondeducibility secure if users with low security levels cannot obtain information at a higher security level as a result of any activity on the part of a higher-level user. Low level users may still be able to observe high-level user behavior, but they cannot interpret the outputs. This property is weaker than noninterference and is noncomposable, though it does not assume determinism.

In order to overcome the limitations of nondeducibility, researchers have proposed various properties such as generalized noninterference [90]. In addition to nondeducibility, possibilistic and probabilistic models for nondeterministic [91, 63] systems, along with a notion of probabilistic noninterference (PNI) and a verification logic for this model have also been proposed [64].

Recently, researchers have explored notions of behavioral equivalence to model some of these issues for non-deterministic systems [115]. A complete discussion of this topic is beyond the scope of this thesis and not directly relevant to our work. There is a great debate in the community about the relevance and cost of general interface models for security, and whether access control models augmented with some covert channel analysis are sufficient for practical systems.

In the next subsection, I briefly describe different models for protecting integrity of information.

2.2.4 Modeling Integrity

Integrity policies are also called commercial security policies as they model accuracy of information, as opposed to confidentiality policies that are commonly referred to as military security policies and model disclosure. In 1977, Biba [20] proposed three policies to capture the notion of authorized modification of information resources as defined by an integrity policy.

A system in Biba's model consists of a set of subjects S, a set of objects O and a set of integrity levels I. The levels are partially-ordered and the function $min : I \times I \rightarrow I$ gives the lesser of the two integrity levels, under $\leq$. The function $i : S \cup O \rightarrow I$ returns the integrity level of a subject or object. Relations $r, w \subseteq S \times O$ and $x \subseteq S \times S$ defines the ability of a subject to read or write an object, and the ability of a subject process to execute another process respectively.

Integrity labels are not the same as confidentiality labels and are usually assigned differently. Biba's policies are defined in the context of an information transfer path.

Definition 2.2.7 (Information Transfer Path). *An information transfer path (or read-write path) is a sequence of objects $o_1, \cdots, o_{n+1}$ and a corresponding sequence of subjects $s_1, \cdots, s_n$, such that $s_i \,\mathbf{r}\, o_i$ and $s_i \,\mathbf{w}\, o_{i+1}$ for all $i, 1 \leq i \leq n$.*

The three policies defined by Biba are as follows:

1. **Low-Water-Mark Policy**: Whenever a subject accesses an object the policy changes the integrity level of the subject to the lower of the subject and the object. Specifically, a subject can only write to an object or execute a subject at a lower level, and must change its level to the lower level when it reads a low level object. Biba further showed that if there is an information transfer path from object $o_1 \in O$ to object $o_{n+1} \in O$, then enforcement of the low-water-mark policy requires that $i(o_{n+1}) \leq i(o_1)$ for all $n > 1$.

2. **Ring Policy**: Any subject may read any object, regardless of integrity levels. A subject can write to an object if and only if $i(o) \leq i(s)$. A subject s_1 can execute another subject s_2 if and only if $i(s_2) \leq i(s_1)$.

3. **Biba's Model (Strict Integrity Policy)**: Biba's model is the dual of the BLP model and its rules are as follows:

   - $s \in S$ can read $o \in O$ iff $i(s) \leq i(o)$
   - $s \in S$ can write to $o \in O$ iff $i(o) \leq i(s)$
   - $s_1 \in S$ can execute $s_2 \in S$ iff $i(s_2) \leq i(s_1)$

Within the framework of Biba's model, we can show that its enforcement preserves the property that if there is an information transfer path from object $o_1 \in O$ to object $o_{n+1} \in O$, then $i(o_{n+1}) \leq i(o_1)$ for all $n > 1$. This property prevents indirect as well as direct modification of entities without authorization.

Lipner combined the BLP model and Biba's model to obtain a combined system model capable of specifying both confidentiality and integrity policies. Lipner's model has both security levels and integrity levels.

In 1987, David Clark and David Wilson developed an integrity model different from the level-oriented BLP and Biba models. Their model is referred to as the Clark-Wilson [30](CW) model. The CW model introduces the notion of trust explicitly in the model through the concepts of certification and authentication. It is well suited to model "separation of duty" constraints, which prevents a subject from having the sole ability to inflict damage on the integrity of protected information. It is expressed in terms of a collection of nine rules designed to provide integrity protection. The rules define well formed *transactions* as a series of operations that change the system from one consistent state to another. A set of integrity constraints specifies the consistency requirements. Integrity verification procedures test whether data items conform to the integrity constraints. Transformation procedures implement well-formed transactions and change the state of data in the system from one valid form to another.

In the next subsection, I briefly describe hybrid models that are capable of specifying both integrity and confidentiality policies under one framework.

2.2.5 Hybrid Models and Role Based Access Control

While the models presented so far are capable of defining either integrity or confidentiality concerns adequately, most information protection systems require a combination of both types of policies. As a result, researchers have worked on hybrid policy models that are capable of addressing both integrity and confidentiality concerns under one framework. The most famous of these hybrid models is the Chinese-Wall Model [25]. This model develops the notion of conflict of interest (COI) classes and partitions subjects and objects into different classes accordingly. It also defines a framework to capture the notion of how past behavior (or history) should affect future information access within this framework. The formal Chinese-Wall model is similar in flavor to the BLP model, and specifies restrictions on read and write transitions between COI classes based on history of access.

Other hybrid models include the Clinical Information System Model [9], the Originator Controlled Access Control Model [62] (ORCON), and Role Based Access Control (RBAC).

The Clinical Information System Model was introduced by Anderson and focuses, not on COI, but on patient confidentiality, authentication before access, and assurance that records are not tampered (integrity) in the context of a hospital administrative system. The model is informal and describes a set of principles for access, creation, deletion, confinement, aggregation, and enforcement of policies for medical records. The ORCON model was developed by Gaubert [62], in which a subject can give another subject the rights to an object, only with the approval of the creator of the object.

The RBAC model[56, 119] is arguably the most popular hybrid model in the industry. The key concept in RBAC is a role, which is a placeholder for a set of users. Each role is associated with a set of permissions, which are its rights on objects. These roles may be organized into a hierarchy to reflect the organizational hierarchy among different users in a system. RBAC maintains two mappings: the User Role Assignment (URA) and the Role Permission Assignment (RPA). These two mappings can be updated independently and this flexibility provides administrators an efficient mechanism to manage and administer access control policies. In recent years, an RBAC NIST standard [55], and a graph-based formalism have also been proposed.

In Section 2.3, I end this chapter with a brief synopsis of the different models and techniques

we presented so far, and provide a preview of Chapter 3, which deals with the problem-scope and the solution-space of this thesis.

2.3 Looking Ahead

In order to keep the analysis of a system model tractable, many researchers focus on finite-state state-transition models of subjects, objects, and privileges and permissions and restrict interactive behavior. These models are mainly concerned about two important but slightly orthogonal concepts. The first set of models, which include the HRU model and its extensions, are concerned with the generalized notion of access control as a fundamental building block of a theory of safe systems. The transitions in these models capture the notion of how access rights change over the lifetime of a model. Security is defined in this framework as the ability of the model to prevent unauthorized transfer of access rights.

The second set of models, which include the BLP, CW, Biba, RBAC etc., capture the notion of security from the point of view of users of the system (or sets of users). They explicitly include a notion of assurance levels and describe the evolution of the system between levels, using the abstraction of valid transitions. We observe from these models that 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 for all possible reachable states through valid transitions to show that unauthorized state-transitions cannot occur.

The modeling and validation process presented in this chapter is an attractive choice for security engineers seeking to certify that a given model provides assurance guarantees. However, in this thesis I argue that formal modeling and validation of safety properties using a state-transition model as described above cannot describe the behavior of a system under attack. 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 addition, many of these models cannot capture non-determinism and concurrency, or express quantitative guarantees in terms of real-time values.

In the next chapter, I explore the need for a new policy specification formalism 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 ("if something bad happens, something good eventually happens") to model survivability or *recovery-oriented security*.

Thus, the task is, not so much to see what no one has yet seen; but to think what nobody has yet thought, about that which everybody sees.

Erwin Schrödinger

Chapter 3

Problem Statement

In this chapter I define my thesis problem, situate it in the broader context of existing formal models of access control systems, and develop a requirements-specification for survivability analysis.

In Section 3.1, I motivate the need for survivability modeling by examining the set of assumptions made by existing models of assurance, and highlighting how they do not represent realistic operating environments of access control systems. I explore how guarantees made by safety modeling are frequently compromised, and investigate how we can extend the scope of these guarantees by defining survivability properties to accommodate for recovery-oriented security.

I formalize this notion in Section 3.2 using a semantic framework general enough to accommodate existing access control abstractions. With this new model, I show how we can write system specifications and explicitly model how "bad" things can happen as the system evolves over time. I also explore how to model the ability of a system to recover, if possible, when the system is attacked maliciously, or compromised inadvertently.

I also show how we can redefine traditional security properties such as confidentiality and integrity as a special class of dynamic properties we call survivability properties. These properties describe how if bad things happen, whether we can guarantee that good things will eventually happen. We also define a metric for survivability, as the bound on computational resources required to restore the availability of such resources to legitimate users of the system over time. I show how we can use this representation to specify and evaluate different strategies for recovery, both qualitatively and quantitatively.

In Section 3.3, I present a high-level overview of my proposed framework, describe how to represent survivability properties within this context, and present an appropriate formalism for

this purpose. I summarize all these ideas as my thesis statement in Section 3.4 and discuss success criteria in Section 3.5.

In Chapter 4, I describe my model, and define the notion of strategies in greater detail. I show how we can incorporate timing guarantees and stochastic behavior, in both the model and property specifications, and discuss how to automate the verification of these properties. I also describe how dynamic access control can be used as an effective RA and discuss how to preserve certain safety properties and trust relationships in this context. In Chapter 5 of this thesis, I focus my attention on the network DoS problem and show how survivability modeling and analysis can specify and verify the effectiveness of DoS resilience strategies.

3.1 Context

Formal methods help a security engineer accomplish two important tasks [130]. Through the process of **specification**, it focuses a system designer's attention on the entities, their behavior, the nature of their interaction, on the assumptions about a system's environment, and on the nature of properties that must be satisfied, e.g., as invariants, in order to claim the system is secure. Through the process of **verification**, it provides additional assurance in terms of properties that can be preserved within the framework of a semantic or syntactic description of system behavior.

The process of proving a system is secure can be broken into three inter-related tasks [130]. The first task (not in any particular order) is to model the system, its entities, and their interaction using appropriate syntactic or semantic abstractions. Second, we express the security property of interest explicitly and formally. The third aspect of assurance is the proof, which may be automated and rely on structural induction, state-space exploration, or deductive reasoning, or may need to be produced manually, to show that the implication (System Model $\Rightarrow$ Security Specification) can be verified.

I develop my survivability properties in the context of an example model of access control behavior that I present in Section 3.1.1. This model is influenced by both the HRU and BLP models and appears in [101].

3.1.1 Semantic Model of Policy Enforcement

In this subsection, we present a formal description of a semantic state-transition model of access control, that acts both as a motivating example, as well as an introduction to the formal notation used in this thesis.

The basic formalism used in this thesis is a type of state-transition graph called a **Kripke** structure. Kripke structures are traditionally used to describe **reactive** systems and their behavior over time [33]. In addition to states and transitions, a Kripke structure also associates each state with a set of propositions that can be evaluated to truth-values depending on the values of boolean variables in that state. We show how we can use this feature to describe qualitative temporal properties of states and paths in a transition graph as the system it describes evolves over time.

Formally, a Kripke structure M over a set of atomic propositions AP, is a four tuple $M = (\Sigma, \Sigma_0, R, L)$, where Σ is a set of states, $\Sigma_0 \subseteq \Sigma$ the set of initial states, $R \subseteq \Sigma \times \Sigma$ is a total transition relation between states, and $L : \Sigma \rightarrow 2^{AP}$ is a function that labels each state with a set of atomic propositions that are true in that state.

We now show how to model the access control problem as a Kripke structure. Each state in Σ corresponds to the protection state of an information protection system, and consists of the set S of subjects, the set O of objects, the set $\mathcal{R}$ of rights, and an access matrix A as described in Section 2.1.1.

In traditional access control models, e.g., in the HRU and BLP models, a transition represents a change in the protection state. The six primitive commands in the HRU model, presented in Section 2.1.1, represent transitions that can change (add or remove elements from) each of the three finite sets S, O, and A (assuming $\mathcal{R}$ is fixed). In the BLP model, transitions are represented by *read*, *write* and *execute* actions. These actions usually change the values of the elements in O. For example a write action modifies an element of O. Reading an object does not change the original object, but can change the state of objects (e.g., a read buffer) belonging to the subject who has read the object. Allowing an execute method on an object may change the state of other information objects in the system. In the BLP model, the access permissions are defined by conditional rules, and the access matrix A is only conceptual. We include both the primitive commands and the standard filesystem commands from both models in the set $\mathcal{R}$ of our general access control model.

The states and transitions in the HRU and BLP models define a labeled-transition system (LTS), where the transitions are labeled by commands or actions. A Kripke model however does not include labels on transitions. A standard technique to convert an LTS to a Kripke structure is to include the initiation and termination of an action or command as propositions that hold in the start state and end state of a transition.

Let the set $\mathcal{R}$ be fixed. Let the set of requests $Q = \{\langle s, o, r\rangle | s \in S, o \in O, r \in \mathcal{R}\}$ be the set of all subject, object, and permission triples in our system. We represent requests in a state with the variable REQ in our model. The domain of REQ is $Q \cup \{\perp\}$. In a transition $\sigma_i \xrightarrow{q} \sigma_{i+1}$, in state σ_i, $(REQ = q)$ and in state σ_{i+1}, $(REQ = \perp)$.

These transitions are assumed to be atomic, and the state does not change until the action completes. A completed action or command results in a new state, where the values of one or more state-variables have changed, i.e., we assume that only one or no request(s) can be issued in each state.

The matrix A defines the current set of access rights in the system. These rights define the set Q_A of rights that are allowed by A. For each request $q \in Q$ we can now associate a boolean variable P_A whose value is true if $q \in Q_A$, and false otherwise. The state of our system now includes this variable P_A in addition to REQ and the sets S, O and A. This corresponds to the access permissions in the current access matrix A.

Definition 3.1.1 (Access Control System). *An access control system can be defined by the Kripke structure $M = (\Sigma, \Sigma_0, R, L)$ over AP where:*

1. $\Sigma = 2^S \times 2^O \times 2^A \times Q \cup \{\perp\} \times \{0, 1\}$
2. $\Sigma_0 = \{\langle S_0, O_0, A_0, (REQ = q), (P_A = 0) V (P_A = 1)\rangle\} \subseteq \Sigma$
3. $R(\langle S, O, A, REQ, P_A\rangle, \langle S', O', A', REQ', P_A{}'\rangle) \subseteq \Sigma \times \Sigma$ *that is total.*
4. $L(\langle S, O, A, REQ, P_A\rangle)$ *defines assertions corresponding to values of variables in S,O,A, REQ and P_A.*

Next, we show how to describe a traditional secure access control system using the abstractions presented so far, by restricting what transitions are allowed to occur between states. The syntax of the transitions T1-T3 between states are specified by the following rules as:

Definition 3.1.2 (T1). *A **secure transition** between two states is given by*

$$\langle S, O, A, (REQ = q), (P_A = 1)\rangle \rightarrow_{T1} \langle S', O', A', (REQ = \perp), ({P_A}' = 0)\rangle$$

Definition 3.1.3 (T2). *A **null transition** is given by*

$$\langle S, O, A, (REQ = q), (P_A = 0)\rangle \rightarrow_{T2} \langle S, O, A, (REQ = \perp), ({P_A}' = 0)\rangle$$

Definition 3.1.4 (T3). *An **unconditional transition** is given by*

$$\langle S, O, A, (REQ = \perp), (P_A = 0)\rangle \rightarrow_{T3} \langle S, O, A, (REQ = q), {P_A}'\rangle$$

Transition T1 specifies "good behavior". A request should be allowed if the corresponding right can be found in the access matrix. The precise meaning of the transition for different requests q needs to be defined clearly for any specific model, by carefully defining the changes allowed to occur in the new state $\langle S, O, A, (REQ = \perp), ({P_A}' = 0)\rangle$ as a result of the transition.

Each transition or action in our system changes the values of one or more elements of our system state. The primitive commands change the protection state, i.e., the sets of subjects, objects and the access matrix themselves. Other actions typically only change the values of information objects. Each command and action is associated with a precise semantics. The semantics for the primitive commands are given in Section 2.1.1.

If the resulting state of such a transition adheres to the semantics specified by the system designer, we can assert that the change was valid. Though we do not include these in our model, to keep the explanation simple, ideally we would augment each secure transition with a set of assertions that hold as **preconditions** in the start state σ_i of a T1—type transition labeled by action q, and a set of assertions that holds as **postconditions** in the end state σ_{i+1} of a transition $\sigma_i \xrightarrow{q} \sigma_{i+1}$.

The sets of propositions *Precondition*(q) and *Postcondition*(q) are evaluated in the start state σ_i and end state σ_{i+1} of a transition for a request $q \in Q$. The transition is allowed to occur if the set of assertions in the set *Precondition*(q) in state σ_i are consistent. After the transition occurs, the transition adheres to the semantics of the action if the set *Postcondition*(q) in σ_{i+1} is consistent. Note these sets may be viewed as Horn-clauses, and that the asserted postconditions in a given state can be used as preconditions when this state σ_{i+1} becomes the start state for the next transition. We can use standard program verification techniques to verify if a secure transition is semantically consistent.

Transition T2 asserts if the request is not authorized by the access matrix, the sets S, O, or A should not change in any way. Transitions T1 and T2 describe what is called the "policy membership" test of safety for access control models. Transition T3 makes the Kripke structure total, and states that our model can enter state where there is a request at any time, regardless of whether P_A is true or not.

This formulation gives us a methodology to validate that a system implements its policies by comparing the observed transitions for each subject with the permissions in the access matrix. If the transitions in the system are restricted to T1, T2 and T3, we observe that the system only exhibits authorized behavior.

Note that this definition of a secure access control system is the same as Definition 2.2.1 of access control safety in Chapter 2.

3.2 Problem

The semantic model of secure transitions with the policy membership test described in Section 3.1.1 is attractive because of its simplicity. To prove this model is secure, we can start with an initial state that is assumed to contain only rights that are authorized by the system policy specification. We can can rely on the definition of transitions T1–T3 to assert that any state reachable from this state is also secure.

The relationship between policies that are enforced by the implementation mechanisms and the policies actually authorized by some system policy needs to be explored in greater detail.

Let Q_{auth} be the (conceptual) set of authorized requests in the system, corresponding to rights that are allowed to particular users by the system policy. This set includes all the possible rights that can be installed in A by authorized users, not just the rights in the current state. In existing systems, the distinction between an access right and a policy-authorized right is *not* maintained. A request is allowed if and only if the corresponding right-to-access can be found in the access matrix A. We show how this is a weakness of existing access control models.

If the set of requests allowed in a system Q_A implementation is a subset of the set of policy-authorized request Q_{auth} then this system is *secure*. If these two sets are equal, this system is *precise*. In Section 3.1.1, we define transitions that correspond to authorized behavior, explicitly

allowed by system policies. This formulation assumes that the policy membership test is always done correctly. This behavior is depicted by the event-graph shown in Figure 3.1.

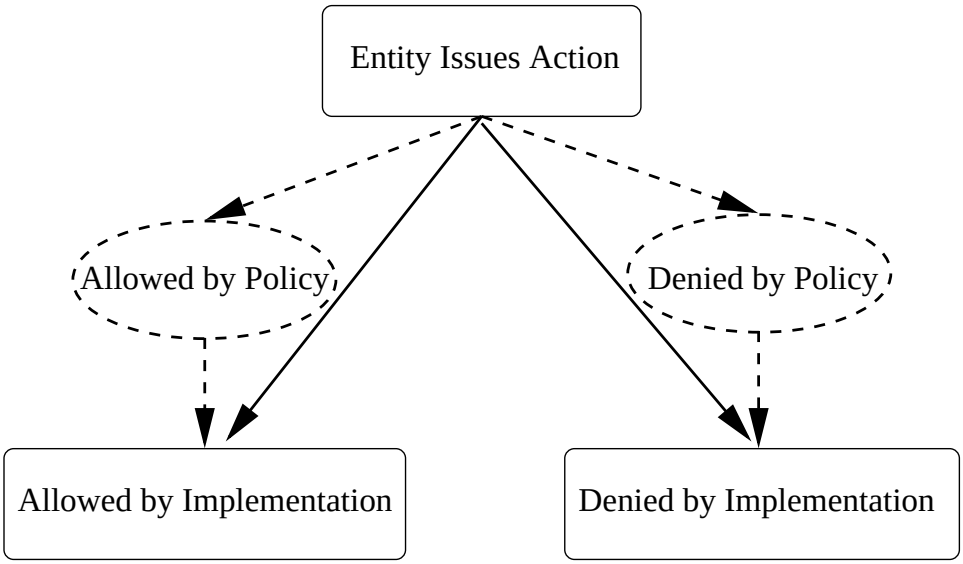

Figure 3.1: Event Graph for Ideal Access Control Decisions

This intermediate test for whether the action is allowed or denied by the policy, represented by the dotted oval, is not performed in most existing systems. These systems assume that the policy implementation mechanisms and the access-rights matrix **cannot be compromised** and the situational policies in the access matrix are consistent with the policies authorized by the system policies. In this thesis, we argue that this assumption is a major weakness of existing policy modeling and analysis techniques. Furthermore, safety guarantees provided by the model become worthless when a system enters a compromised state.

In order to describe the behavior of a system under attack, we argue that we need to explore how things can "go wrong". To make the model more expressive, I argue that we need to explicitly include a notion of *compromise* and, whenever possible, define strategies (as a sequence of states and transitions) that model *recovery*.

The set of policy-authorized requests Q_{auth} is a good starting point for this exercise. The challenge is to separate the policy from mechanism, and ensure that any violation or modification of the policy by attackers can be detected in the system.

In 3.2.1 next, I describe how we can extend the semantic model of Section 3.1.1 to explicitly accommodate for insecure behavior and subsequent recovery, to model survivability.

3.2.1 Modeling Insecurity

In real systems, the protection state of the system may be inconsistent with the behavior described by the authorized policies in the system. For example, an attacker may be able to alter the permissions in the access matrix and compromise the confidentiality or integrity of the information, as in the case of a privilege-escalation buffer overflow attack. Neither the model presented in Section 3.1.1, nor any of state-of-the-art models presented in Chapter 2 can describe this inconsistency as "undesirable" behavior.

Specifically, since the system does not make a distinction between what is authorized and what is installed in the access matrix, this attack cannot be detected by analyzing existing models. In order to differentiate between the actions authorized by actual policy and the actions enforced by the system's mechanisms, we argue that this distinction needs to be exposed at the model level. Therefore we augment the state of our model with a new boolean variable called P_{auth}. This variable is true when the current request q is in Q_{auth} and false otherwise.

Another issue that frequently arises in existing systems is when a legitimate subject makes an authorized request to access an object, and this action is denied. Note this unavailability may only be temporary. It may be because of poor-resource engineering, in scenarios where many legitimate users compete for the same resource. Another source of this type of insecurity is a denial of service (DoS) attack, when a malicious user or attacker denies access to legitimate users by competing for the same resource. Both these cases may be viewed as instances when the policies are not being enforced correctly temporarily. This insecurity may or may not result in an inconsistent state.

We show both these types of policy enforcement failures pictorially by augmenting the event-graph with faults to produce a fault-graph in Figure 3.2.

In particular, the dashed edges in Figure 3.2 correspond to the transitions that violate the policy authorization test. The first type of compromise can be viewed as a high-level description of *denial of service* (DoS), i.e., an authorized entity is prevented from completing a legitimate request. However, the information itself may not compromised and its effect on the system is usually benign. The second type of insecurity can lead to *information compromise*, i.e., irretrievable transfer or modification of information by unauthorized entities. Note the distinction between policy compromise and information compromise. Policy compromise may lead to information compromise.

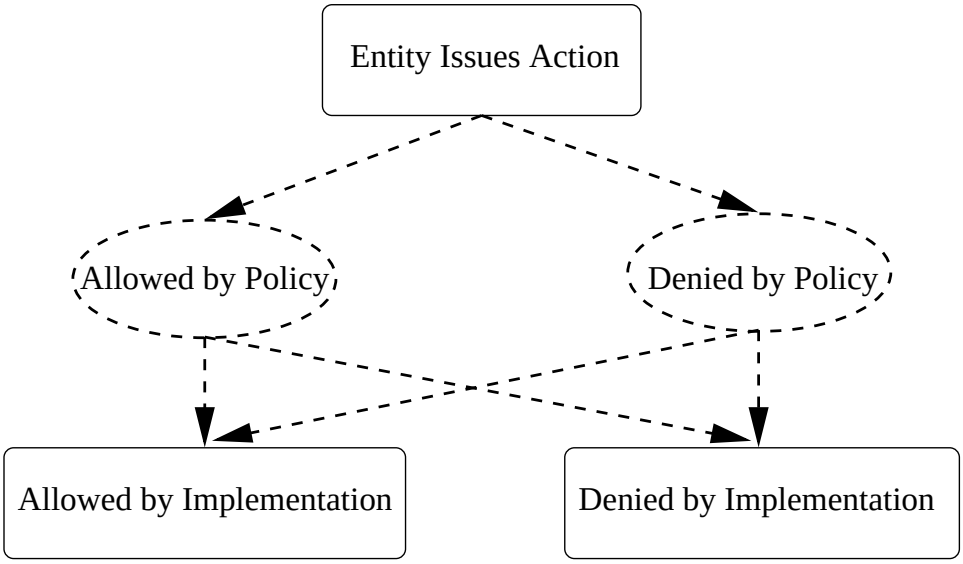

Figure 3.2: Fault graph for Access Control Decisions

Definition 3.2.1 (Augmented AC system). *The Augmented Access Control System which explicitly separates policy and mechanism is represented by the following Kripke structure*$M' = (\Sigma', \Sigma'_0, R', L')$ *over AP as:*

1. $\Sigma' = 2^S \times 2^O \times 2^A \times Q \cup \{\perp\} \times \{0,1\} \times \{0,1\}$
2. $\Sigma'_0 = \{\langle S_0, O_0, A_0, (REQ = q), (P_A = 0) \vee (P_A = 1), (P_{auth} = 0) \vee (P_{auth} = 1)\rangle\} \subseteq \Sigma'$
3. $R'(\langle S, O, A, REQ, P_A, P_{auth}\rangle, \langle S', O', A', REQ', {P_A}', {P_{auth}}'\rangle) \subseteq \Sigma' \times \Sigma'$ *that is total.*
4. $L'(\langle S, O, A, REQ, P_A, P_{auth}\rangle)$ *defines assertions corresponding to values of variables in S,O,A, REQ,* P_A*, and* P_{auth}*.*

We introduce these two types of insecure behavior into our model using the following transitions:

Definition 3.2.2 (T4). *A* ***benign insecure*** *transition is given by*

$\langle S, O, A, (REQ = q), (\mathbf{P_A = 1}), (\mathbf{P_{auth} = 1})\rangle \rightarrow_{T4} \langle S, O, A, (REQ = \perp), (\mathbf{{P_A}' = 0}), (\mathbf{{P_{auth}}' = 0})\rangle$

Definition 3.2.3 (T5). *A* ***malicious insecure*** *transition is given by*

$\langle S, O, A, (REQ = q), (\mathbf{P_A = 1})(\mathbf{P_{auth} = 0})\rangle \rightarrow_{T5} \langle S', O', A', (REQ = \perp), (\mathbf{{P_A}' = 0})(\mathbf{{P_{auth}}' = 0})\rangle$

An insecure but **benign** transition leaves the system unchanged, even if the action was authorized by the policy. Note an access right corresponding to this action is explicitly present in

the access matrix, as well in the set of authorized actions. This transition has no effect on the state variables (S, O, A). The **malicious** insecure policy changes the variables in $\langle S, O, A, q\rangle$, even though q it is not in Q_{auth}. The exact nature of this unauthorized change is model-specific.

The original transitions T1–T3 are presented next in their modified form. They are essentially the same as the transitions in Section 3.2, except for the addition of the variable P_{auth} in every state. In T1', $(P_{auth} = 1)$ indicates that the secure transition is also authorized by the policy. Similarly, null transition is specifically not authorized by the policy, and therefore $(P_{auth} = 0)$ in $T2'$ as next.

Definition 3.2.4 (T1'). *A **secure transition** between two states is given by*
$\langle S, O, A, (REQ = q), (\mathbf{P_A = 1})(\mathbf{P_{auth} = 1})\rangle \rightarrow_{T1} \langle S', O', A', (REQ = \perp), ({P_A}' = 0)({P_{auth}}' = 0)\rangle$.

Definition 3.2.5 (T2'). *A **null transition** is given by*
$\langle S, O, A, (REQ = q), (\mathbf{P_A = 0})(\mathbf{P_{auth} = 0})\rangle \rightarrow_{T2} \langle S, O, A, (REQ = \perp), ({P_A}' = 0), ({P_{auth}}' = 0)\rangle$

Definition 3.2.6 (T3'). *An **unconditional transition** is given by*
$\langle S, O, A, (REQ = \perp), (P_A = 0), (P_{auth} = 0)\rangle \rightarrow_{T3} \langle S, O, A, (REQ = q), {P_A}', {P_{auth}}'\rangle$.

For completeness, the state where $(P_A = 0) \wedge (P_{auth} = 1)$ indicates that the user is allowed to execute the action according to the system policy, but the right to execute this action is not available in A at this point. This may be because the user has not explicitly assigned themselves this permission. The system may prompt the user at this point with appropriate feedback, or the user may realize this behavior and assign themselves the appropriate right. Therefore this behavior imprecise but not insecure.

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, is denied this action by policy enforcement mechanisms. Information that could be exchanged freely has now become unavailable. However this compromise does not propagate confidential information.

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, 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, I explore what we mean by recovery from information compromise, specifically in the case of confidentiality and integrity policies. A complete analysis of compromise and recovery in the case of information-flow policies is beyond the scope of this thesis.

3.2.2 Recovery

Recovering from policy compromise may involve exercising administrative control and changing access control permissions dynamically, in response to an attack or vulnerability exposure, thereby preventing future spread of compromised information. For confidentiality compromise, if we can reliably constrain future interactive behavior of the entity that obtained the compromised information, we can argue that confidentiality from the viewpoint of the rest of the system is still preserved.

When a resource's integrity has been compromised by an unauthorized user, and is detected at a later point in time, the system may be able to rollback actions and restore values to a previously consistent state of the system, thereby enabling a weaker notion of integrity. Sometimes it may be necessary to restart the system in order to accomplish this. Other strategies such as storing multiple redundant copies, using error correcting codes, etc., can help restore integrity even under attack.

Modeling assurance requirements as safety properties cannot describe or quantify the impact of such strategies that explicitly recognize insecure states and transitions and adapt their behavior in response, by restricting future behavior of the system, and eventually bringing the system to an authorized state again. However, it is also important to realize that using these RAs in an ad hoc manner can create more vulnerabilities and opportunities for attack. The challenge therefore is to provide a framework that can represent different types of dynamic strategies such as the ones described above, and reason about the impact of these strategies on the security and survivability of a system model.

In Section 3.3, I explore how to specify survivability properties that can represent the effectiveness of different RAs, using our extended semantic model of access control.

3.3 Solution Space

In this section, I give a high-level overview of how to specify, quantify, and verify the ability of an access control model to survive policy compromise. I introduce a new class of survivability properties, modeled as branching-time properties on subgraphs of state-transition models. These properties describe the ability of (a part of) the system, comprising of specific subjects and objects, to continue to provide specific security guarantees even when other parts of system are compromised.

These properties specify the notion that even when certain entities in the system are compromised (i.e., bad things happen), the system is able to guarantee, for a specific set of users with respect to a particular property, that good things will eventually happen. We call these properties "dynamic policies" since they represent the ability of a model to survive different types of compromise, in both qualitative and quantitative terms

In Section 3.3.1, I define survivability as a property of execution traces of system behavior. I show example specifications that explicitly model compromise and recovery, and analyze them with respect to their ability to satisfy survivability properties. At the end of this section, I explore how we can use standard model-checking and other analysis techniques to validate these survivability properties in a given model.

3.3.1 Specifying Survivability

We define survivability as a special type of liveness property that captures the ability of an information protection system to continue providing certain security guarantees, even in the face of policy compromise. Policy compromise or information compromise may cause the unavailability of a resource to its authorized users. Recovery involves restoring this availability after compromise. Therefore, measuring this availability of the resource to legitimate users is an integral part of survivability modeling.

In traditional safety property validation, the analysis begins with a "secure" initial state that satisfies the property of interest, and the satisfaction of this property is tracked across all reachable authorized transitions from this state. For survivability, we are more interested in tracking the evolution of a system whenever we observe something "bad" occurs. Furthermore, like confidentiality and integrity properties, we are interested in the survivability from the viewpoint of an authorized user in the system. With this in mind, we relate survivability to traditional confidentiality and integrity properties as follows:

- **Survivability**: Given that subjects $s' \in S' \subseteq S$ can access a subset of objects $A \subseteq O$ using rights $r \in R$ according to either a confidentiality or integrity policy $p \in P_C \cup P_I$, a survivability policy $p_v \in P_V$ with respect to subject $s' \in S'$, partitions the set of subjects in S' into two disjoint sets S'_V and $\overline{S'_V}$. Subjects in S'_V are guaranteed that when a policy compromise occurs and the resources become unavailable, the system will restore their ability to access it *securely*, *eventually*, and *infinitely often*. Furthermore, this recovery behavior can be quantified in terms of timing or probabilistic guarantees. Subjects in $\overline{S'_A}$ are not provided any such guarantees.

We define the notion of a survivability strategy as follows. A survivability strategy can be viewed as the interactive behavior between two types of entities in our system: an **adversary** and a **controller**. An adversary corresponds to an attacker and the controller is a system administrator who is authorized to make changes to the protection state. A strategy is modeled as a special state-transition subgraph that starts with a secure state in our model and an insecure transition, initiated by an adversary, to indicate the behavior of the system under attack.

The system evolves through different states according to transitions specified by the strategy, representing the interactive behavior of both the adversary and controller, and ends in a secure state, indicating the system has recovered from the attack. Note that the strategy can be evaluated only in the context of the overall model of the system. The start state and end state of a strategy may be indistinguishable in which case the effect of the attack is neutralized, impacting only the availability of the resource when the recovery actions were in progress. On the other hand, the end state of a strategy may be only partially consistent with the policies, indicating that the attack was partially recoverable. We define strategies and the notion of an adversary and a controller in greater detail in Chapter 4.

We define survivability properties as temporal logic formulas that can be satisfied by traces of system behavior augmented with these survivability strategies.

We use CTL [33] and its probabilistic and real-time extensions to formulate these properties. CTL is the most widely used logic to specify properties of branching-time models, and we believe is an appropriate formalism in this context. Since CTL is widely used, we do not define its syntax and semantics in this chapter, but refer the reader to Appendix A.

Temporal logic formulas in CTL are interpreted in the context of Kripke structures. A *computation* in a Kripke structure is an infinite sequence of states where each state is obtained from the previous state by some valid transition in R. A *path* is a model of a specific computation of the system. A *computation tree* is formed by designating a state in the Kripke structure as an initial state and then unwinding the structure into an infinite tree with the designated state as root. This tree shows all possible executions starting from that state.

CTL formulas are interpreted over such computation trees of entity behavior. A model satisfies a CTL state formula $\mathbf{A}f$, if we can prove f is true "in all computation paths" from that state, it satisfies formula $\mathbf{E}f$ if it is true "in some computation path" from that state. Similarly the temporal operator $\mathbf{G}f$ is defined over paths and asserts that the formula f is true in all states along the current path in the future, and the formula $\mathbf{F}f$ over paths asserts that some state along the current path 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.

3.3.2 Example Specification and Verification of Survivability Properties

In this subsection, we illustrate how defining survivability as a special type of 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 in the context of the state-transition model described in Sections 3.2.1:

$S = \{John\}$

$O = \{foo, bar, temp\}$

$\mathcal{R} = \{r, w\}$

$A[John, foo] = r, A[John, temp] = r$

$Q_{auth} = \{\langle John, foo, r\rangle, \langle John, temp, r\rangle\langle John, temp, w\rangle\}$

We describe transitions T1 through T5 for this system in Table 3.1 next. Note we do not explicitly show the state variables that do not change in each state for space constraints:

	Start state	**End State**
T1	$\langle(REQ = \langle John, foo, r\rangle), (P_A = 1)(P_{auth} = 1)\rangle$	$\langle temp := foo, (REQ = \perp), ({P_A}' = 0), ({P_{auth}}' = 0)\rangle$
T2	$\langle(REQ = \langle John, bar, r, \rangle), (P_A = 0)(P_{auth} = 0)\rangle$	$\langle(REQ = \perp), ({P_A}' = 0)({P_{auth}}' = 0)\rangle$
T3	$\langle(REQ = \perp), (P_A = 0), (P_{auth} = 0)\rangle$	$\langle(REQ = q), {P_A}', {P_{auth}}'\rangle$
T4	$\langle(REQ = \langle John, foo, r\rangle), (P_A = 1)(P_{auth} = 1)\rangle$	$\langle(REQ = \perp), ({P_A}' = 0), ({P_{auth}}' = 0)\rangle$
T5	$\langle(REQ = \langle John, bar, r\rangle)(P_A = 1)(P_{auth} = 0)\rangle$	$\langle\{temp := bar\}, (REQ = \perp), ({P_A}' = 0), ({P_{auth}}' = 0)\rangle$

Table 3.1: Transitions for Example System

John's behavior can be specified using the behavior graph shown in Figure 3.3(i). The initial state of John's behavior can be the start state of any transition. Whenever John issues an action to read file *foo*, a transition of type T1 (secure) is enabled because the tuple $\langle John, foo, r\rangle$ is an authorized policy, and is present in the access matrix. The variable *temp* is assigned the value of foo as a result of this action. If John attempts to read file *bar*, since it is not explicitly allowed, a null transition (type T2) occurs as shown in Figure 3.3(ii). Note we represent it as two separate graphs for convenience, and the starting state of a computation tree of this model can be the start state of Figure 3.3(i) or (ii).

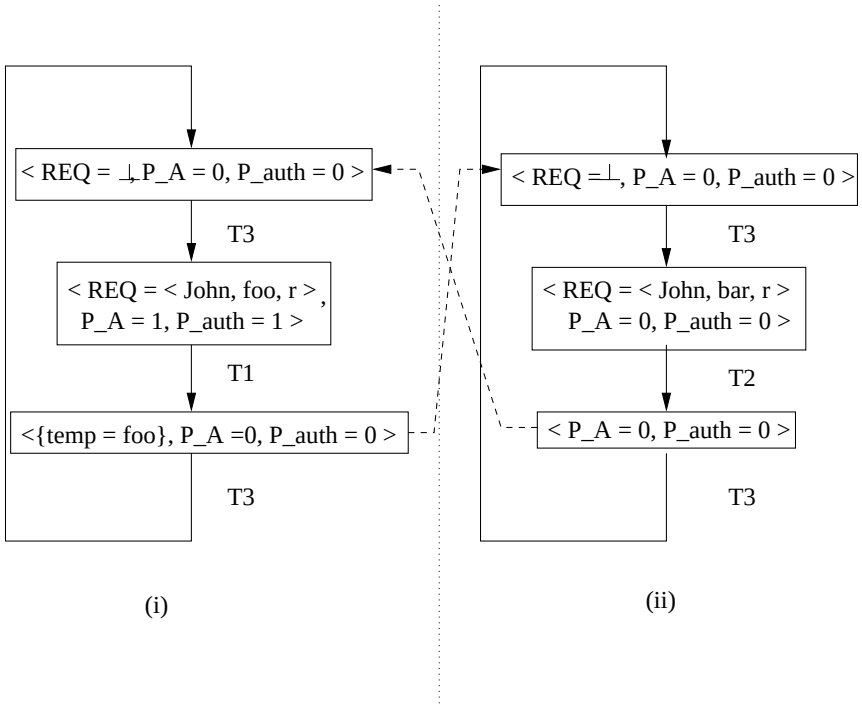

Figure 3.3: Behavioral Specification for Safety

The traditional definition of access control safety to specify John's desirable behavior, in a computation tree starting in state $\langle (S, O, A), (REQ = q), P_A, P_{auth} \rangle$ is given next. Let σ_i^{Tk} be the start state of a Tk transition and σ_{i+1}^{Tk} the end state. Traditional access control safety is given as ($\rightarrow$ is boolean implication):

Property 1 (Access Control Safety). AG$((\sigma_i^{T1} \rightarrow \sigma_{i+1}^{T1}) \vee (\sigma_i^{T2} \rightarrow \sigma_{i+1}^{T2}) \vee (\sigma_i^{T3} \rightarrow \sigma_{i+1}^{T3}))$

This property asserts that John is allowed to read file foo if the policy allows it (T1), and obtain an appropriate response. It also asserts that John cannot read file *bar* for which he does not have access according to the policy (T2). The only other transitions allowed in this system, if they are not T1 or T2, are of type T3. This property specification can be trivially verified against the example specification.

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 3.3, is not capable of describing this behav-

ior. The system may enable transitions of type T4 (benign insecurity) or of type T5 (malicious insecurity). The state-transition graph of Figure 3.3 is augmented with insecure states as shown in Figure 3.4.

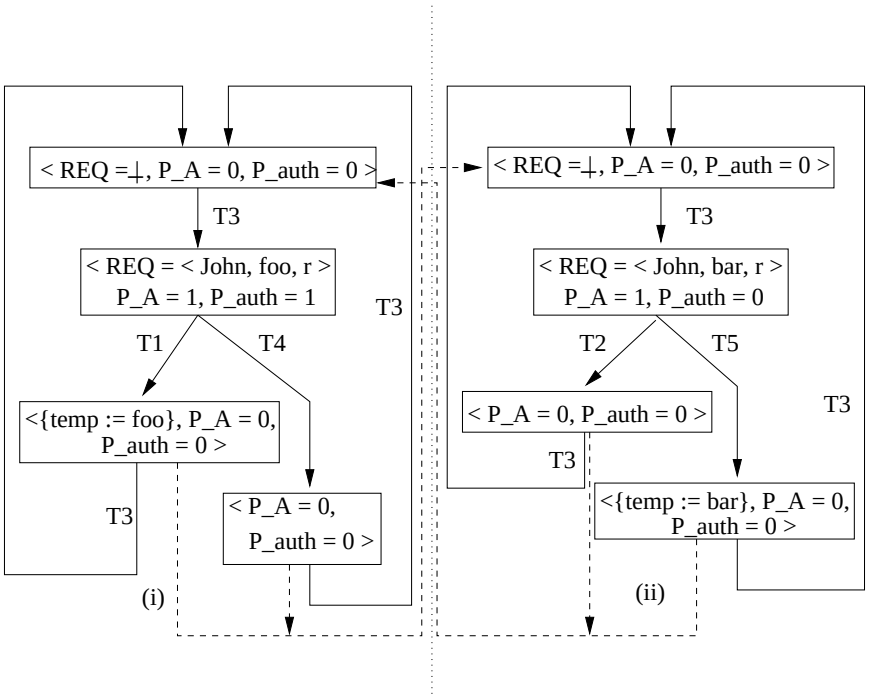

Figure 3.4: Behavioral Specification with Compromise

In Figure 3.4(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 3.4(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 2 (Access Control Survivability). $\mathbf{AG}((\sigma_i^{T4} \rightarrow \sigma_{i+1}^{T4}) \rightarrow \mathbf{EF}(\sigma_i^{T1} \rightarrow \sigma_{i+1}^{T1}))$

This property asserts that whenever John issues an authorized request to read file foo, and this is denied, at some point in the future, the system is capable of recovering from this action and able to satisfy another legitimate request for the same object at some time in the future. Verifying this

property by inspection in this model asserts that this model is indeed *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 3.4(ii) also satisfies:

Property 3 (Access Control Vulnerability). $\mathbf{AG}((\sigma_i^{T5} \rightarrow \sigma_{i+1}^{T5}) \rightarrow \mathbf{EF}(\sigma_i^{T5} \rightarrow \sigma_{i+1}^{T5}))$

Recovery

Next, we examine if we can augment Specification 3.4(ii) to include an explicit notion of recovery. If the information *bar* is compromised, i.e., John manages to read *bar* when he is not supposed to, we explore how we can change the specification and recover from this exposure, without impacting the rest of the system.

For this we define the *nil* as an empty file. We augment the specification of Figure 3.4(ii) with a new transition that resets the value of *temp* to *nil* after it is assigned to *bar*, before John can read *temp*. This hypothetical situation is shown in Figure 3.5.

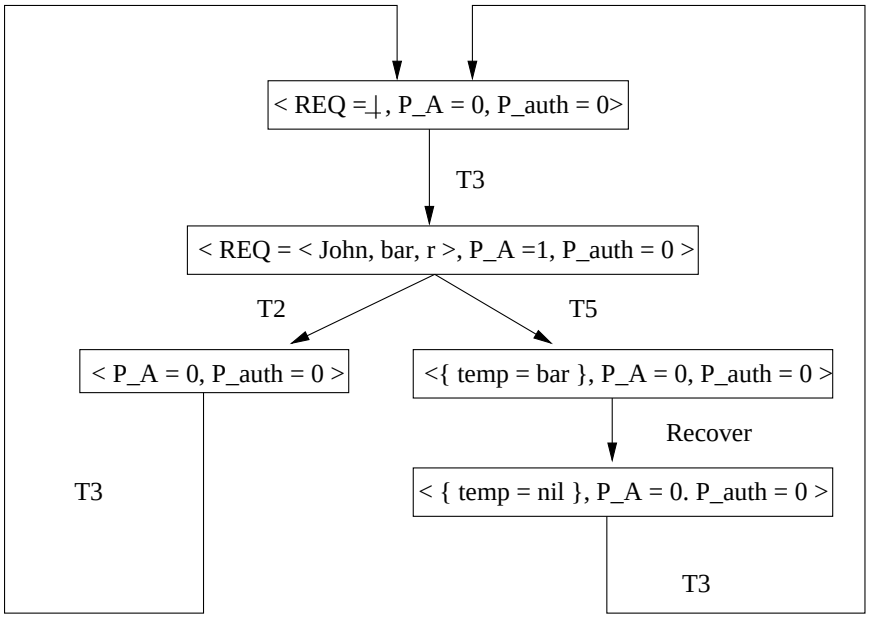

Figure 3.5: Behavioral Specification with Recovery

We define a new recovery action q_r that can be initiated by an administrator of the system. The transition T_r corresponding to this RA is given as follows:

Definition 3.3.1 (T_r). *A recovery action for the example model can be specified as follows:* $\langle\{temp := bar\}(REQ = q_r), (P_A = 1), (P_{auth} = 1)\rangle \rightarrow \langle\{temp := nil\}, (REQ' = \perp), (P_A{}' = 0), (P_{auth}{}' = 0)\rangle$.

This action resets the value of the file *temp* to *nil*. If this action can be executed by the system before any other action that attempts to read the value of *temp* is successful, then we can safely recover from this insecurity.

Property 4 (Access Control Recovery I). *The access control recovery property is specified as follows:* $\mathbf{AG}(\sigma_i^{T_r} \rightarrow \mathbf{A}(\langle(REQ \neq \langle John, temp, r/w\rangle)\rangle)\,\mathbf{U}\,\sigma_{i+1}^{T_r})$

This asserts that if a transition of type T5 occurs, the model should recover from this by setting John's file *temp* to *nil*, before John reads or writes to *temp*. If we can guarantee this modified behavior, then the system can recover from a confidentiality compromise.In a real system, an intruder detection system (IDS) may alert an administrator that the information is compromised. The administrator may observe the system at some later point in time, and analyze the trace of requests after the attack occurred and can decide whether the recover action is useful or not at this stage. In order to show the model is compromise-free, we have to also show how it can also prevent transitions of type T4 in Figure 3.4(i) as:

Property 5 (Access Control Recovery II). $\mathbf{AG}(\neg(\sigma_i^{T4} \rightarrow \sigma_{i+1}^{T4}))$

While all the properties shown so far can be verified by hand, we show how we can rely on model-checking techniques for CTL formulas to automate this process, if we can restrict our focus to finite-state or finite-state abstractions of access control models, in Chapter 4.

So far we have only modeled a system with one entity. When we have multiple entities and resources, the specification process can get complicated. To address this issue, in Chapter 4, we introduce the abstraction of a request-response trace that captures the progress of an access request across different entities in our system, and show how we can describe behavior across these different entities. In addition, we show how we can extend the model to incorporate timing guarantees and stochastic behavior to develop useful measures of survivability.

When an resource is compromised, recovery may involve temporarily suspending read and write access by other entities, making it unavailable in the process. In Chapter 4, 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 easily.

In addition to malicious behavior, 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, in Chapter 5, I showcase the expressive power of our framework by extending our simple semantic model to include resource consumption behavior on different entities in the system, and define useful properties to explore and analyze different strategies for recovery from DoS attacks.

3.4 Thesis Statement

From the description of the context, problem and solution space, I state my thesis as follows:

- Abstract models of information protection systems can be made more relevant by incorporating *insecure* states and transitions explicitly, and by refining traditional safety properties such as confidentiality and integrity as liveness properties in terms of their ability to survive attacks and be available to legitimate users over time. This new abstraction allows one to integrate notions of compromise, recovery, and denial of service into policy specification and verification methodologies, enables *recovery-oriented security*, and provides a framework to study different attack-recovery strategies and analyze their impact on improving survivability of critical information resources.

3.5 Success Criteria

I propose the following criteria to evaluate my thesis:

- Does the proposed thesis advance the state-of-the-art with respect to security property modeling and verification?
- Is the formulation of survivability properties in terms of their availability appropriate for the

context? Can it describe useful measures of recovery and resilience to attacks, and capture the cost-benefit issues of interest to security designers?

- Are the proposed extensions to access control models expressive enough to capture temporal and stochastic notions of compromise, recovery, and denial of service adequately, both quantitatively and qualitatively?

- Does the proposed framework for analysis of recovery strategies, based on the abstract model, address trust and consistency concerns adequately?

- Are these properties verifiable in the framework of the model? Is the verification methodology automatable? Can it leverage existing and incorporate proposed state-of-the-art with respect to automated verification of properties? Is the verification scalable? How can it be extended to models of large systems?

- Is the framework capable of describing a sufficiently complex model to be useful in practice, as highlighted by the modeling and analysis of quantitative and qualitative aspects of network DoS attacks and attack resilience?

Things alter for the worse spontaneously,
if they be not altered for the better
designedly.

Francis Bacon

Chapter 4

Modeling Recovery

In this chapter I show how we can extend the model and the qualitative semantic framework presented in Chapter 3 to include stochastic guarantees and timing behavior. In Section 4.1 we motivate the need for these extensions, and show how we can model an access control system as a PNS (a probabilistic non-deterministic system) that integrates both stochastic and nondeterministic behavior into state-transition semantic descriptions of reactive systems.

The Kripke model we present in Chapter 3 is equivalent to a non-deterministic finite-state machine. Modeling the access control problem with nondeterminism captures the behavior that any user can issue any request at any time. Extending this to describe stochastic behavior allows us to describe dynamic resource access scenarios, where requests issued by users, say accessing file-servers or web-servers, can be modeled as random variables. We also present an extension of PNSes, called the Timed PNS (TPNS) that can be used to model realtime timing guarantees. With these extensions, we can define quantitative and probabilistic survivability guarantees in the context of recovery-oriented security.

A PNS can be represented as a DTMC (Discrete Time Markov Chain), or as a CTMC (Continuous Time Markov Chain), its real-time variant, if we have a purely probabilistic finite-state system model. It is equivalent to an MDP (Markov Decision Processes) if we include nondeterminism. DTMCs, MDPs and CTMCs have been widely used in the past to describe dependability and reliability properties of networked systems. The correspondence between a PNS and these abstractions allow us to transplant related concepts and analysis techniques from these models to our recovery-oriented model of security. Survivability of a system under attack can be viewed as an application of these concepts to security models, with the added capability of describing qualitative

properties using temporal logics and their associated semantics.

In Section 4.2, in order to specify the recovery behavior of a system under attack, we develop the notion of a recovery **strategy**, in terms of the behavior of two types of users in our model: adversaries and controllers. We show how we can model the behavior of these types of users within the context of a PNS or TPNS, and describe attacks, countermeasures and recovery with the help of representative examples.

We also show how we can take advantage of state-of-the-art model checking techniques to verify if a particular system model can provide survivability guarantees of interest. We summarize how we can specify, analyze, and validate survivability properties against these models using $PCTL$ (Probabilistic Computation Tree Logic) for DTMCs and MDPs and CSL (Continuous Stochastic Logic) for CTMCs, and comment on the applicability of runtime verification techniques.

In Section 4.3 of this chapter, I examine how the survivability of a system degrades when a user's account in the system is compromised. Specifically, I examine the impact of this compromise with regard to confidentiality and integrity policies . I discuss how we can characterize the set of users who may be affected because of this compromise by analyzing explicit information flows. We show how to quantify the damage and discuss what properties a policy configuration should satisfy to minimize or control the impact of such attacks.

Finally, in Section 4.4, we examine the performance characteristics of dynamically changing access control policies and mechanisms as RAs within this framework, and investigate the nature of trust relationships and safety guarantees that need to be preserved in order to use them as RAs effectively.

4.1 Towards a More Expressive System Model

Our model presented in Chapter 3 can describe the behavior of a generic access control system, in terms of both actions that can change the protection state itself, as well as actions that can change the values of information objects in the system. It also provides a richer semantics that can differentiate between an action that is authorized by a policy, versus and action that is enabled by a mechanism. This distinction is important to recognize attacks and devise survivability mechanisms that can respond to these attacks.

From the point of view of a security engineer, our framework can be used to model access control mechanisms of a particular system implementation. Using the accompanying analysis framework, one can evaluate this model-instance by exploring how the system can be compromised. For each compromise, one can further specify and investigate different recovery strategies, and gauge their overall impact on the preservation of properties of interest in the system under attack.

Existing models of access control behavior are purely qualitative. By this we mean that the security properties in the model depend only on the semantics of the actions, in terms of changes in the system state that occur as a result of the actions. While such models of security systems are useful in their own right, they do not provide the ability to express quantitative properties that are integral to modeling survivability.

We identify two types of policy compromise that can lead to attacks in Chapter 3: information compromise attacks and DoS attacks. We argue that in order study the impact of different recovery strategies under these attacks, the model we have presented so far does not give us the necessary power of expression. In terms of response strategies for information compromise attacks, we are interested in measuring the relative benefits of using one strategy over another. One of the metrics to perform this comparison is how fast we can recover when a compromise occurs. Ideally, we would like to restrict our study of strategies to those that are able to provide an acceptable bound on when the model reaches a consistent state or when a resource becomes available again. This is called a "bounded availability" property.

In the case of DoS attacks, we need to model the usage patterns of the resource by both legitimate users as well as attackers over time. By changing the request behavior of different users (including attackers) over time, we can study the impact of their actions on the availability of the resource to legitimate users of the system. Furthermore, a strategy that is effective against a DoS attack should be able to demonstrate that the availability guarantees of a resource to its legitimate users improves over time, as a result of the strategy.

In order to represent both the behavior of a system under attack and the impact of a recovery strategy on the survivability of a system, we argue that we need to augment the state-transition graphs with explicit quantitative models of resource-request behavior, including such parameters as request sending rates, arrival rates and service rates. If we view the information objects in our

system as network resources, we can leverage the rich theory of stochastic processes and queuing theory to describe and model this behavior in terms of the request-response behavior of different users in our system. In Section 4.1.1, I discuss how we can extend the simple state-transition graphs to include descriptions of stochastic behavior. I also show how we can incorporate real-time into state-transition graph models and describe quantitative survivability properties within this framework.

4.1.1 Modeling Stochastic Behavior

We present the following abstractions defined in the theory of Probabilistic-Nondeterministic Systems (PNS) from [109, 110, 19], which can represent a system that displays both probabilistic and nondeterministic behavior. A PNS augments the state-transition function of a Kripke model with what is called the *next-state probability distribution* which is defined as follows:

Definition 4.1.1 (Next-state probability distribution). *The next-state probability distribution for a state space Σ is a function $p : \Sigma \to [0, 1]$ such that $\sum_{\sigma \in \Sigma} p(\sigma) = 1$. For each $\sigma \in \Sigma$, $p(\sigma)$ represents the probability of making a one-step transition from the current state to state σ.*

Using this function, we can now define a PNS over a set of atomic propositions AP as follows:

Definition 4.1.2 (PNS). *A PNS is a 4-tuple $\Pi = (\Sigma, \Sigma_0, \tau, L)$ where:*

1. *Σ is the denumerable or finite state space of the system*
2. *$\sigma_{in} \in \Sigma$ is an initial state*
3. *τ is a function that associates each $\sigma \in \Sigma$ with the set $\tau(\sigma) = \{p_1^\sigma, \cdots, p_{k_\sigma}^\sigma\}$ of the next-state probability distributions from σ. Cardinality of $|\tau(\sigma)|$ is k_σ.*
4. *L is the labeling function that associates each $\sigma \in \Sigma$ with the set $L(\sigma) \in 2^{AP}$ of propositions that are true in σ.*

Note the close correspondence between the definition of a PNS and a standard Kripke structure. The relation R is now replaced by the function V between two states. Finding the successor state of a given state $s \in S$ is a two-step process:

1. A next-state probability distribution p_i^σ may be selected nondeterministically among set $\tau(\sigma)$

2. A successor state $\sigma' \in \Sigma$ is chosen according to the distribution p_i^σ on S.

The reachability relation $\rho \subseteq \Sigma \times \Sigma$ is defined as follows:

Definition 4.1.3 (Reachability). $\rho = \{(\sigma, \sigma') \,|\, \exists p^\sigma \in \tau(\sigma) \wedge p^\sigma(\sigma') > 0\}$

With each state $\sigma \in \Sigma$, we represent the set of *legal* infinite sequences Ω_σ of states starting at $s = \sigma_0$ as the set:

$$\Omega_s = \{\sigma_0\sigma_1\sigma_2 \cdots | \sigma = \sigma_0 \wedge \forall n \in \mathbf{N}.\rho(\sigma_n, \sigma_{n+1})\}$$

The set of computations of this system Π is $\Omega_{\sigma_{in}}$. For $\omega \in \Omega_s$, $\omega \mid_n$ is the n-th state of ω, with $\omega \mid_0 = s$.

The model described here can be viewed as a simple encoding of the parallel composition of m Markov chains $A_1, \cdots, A_m$ [127]. In a PNS Π representing $A_1 \| A_2 \| \cdots \| A_m$, with each state $\sigma \in \Sigma$, we can associate the next state distributions ($\tau(\sigma) = \{p_1^\sigma, \cdots, p_m^\sigma\}$, where the distribution p_i^σ arises from a move taken by a chain A_i. The probabilistic behavior of each chain is preserved in Π, and the choice of the Markov chain that takes the transition is nondeterministic.

In order to analyze the properties of such a system, we need to define a probability measure on the set Ω_s of legal infinite sequences of states beginning at some state s. The standard technique is to define $\mathcal{B}_\sigma \subseteq 2^{\Omega_s}$ as the smallest algebra of the subsets of Ω_s that contain all the basic cylinder sets $\{\omega \in \Omega_s | \omega \mid_0 = \sigma_0 \wedge \cdots \omega \mid_n = \sigma_n\}$ for all $n \geq 0$ and $\sigma_0, \cdots, \sigma_n \in \Sigma$ that is closed under complement, and countable unions and intersections. This algebra is called the *Borel σ-algebra* of basic cylinder sets and its elements are measurable sets of sequences.

Due to the presence of nondeterminism, we cannot define a probability measure on $\mathcal{B}_\sigma$. However, for each set of sequences $\Delta \in \mathcal{B}_\sigma$, we can define its *maximal probability* $\mu_\sigma^+(\Delta)$ and its *minimal probability* $\mu_\sigma^-(\Delta)$.

Informally $\mu_\sigma^+(\Delta)$ represents the probability that the system follows a sequence in Δ given that the nondeterministic choices are as favorable as possible. Similarly $\mu_\sigma^+(\Delta)$ is when these choices are as unfavorable as possible.

Formally, these maximal and minimal probabilities are explained with the help of *strategies* or *schedules* that determine which next state probability distribution is chosen for each state. In order

to prevent confusion with the notion of a recovery strategy, we will qualify each use of the word in the other sections.

We are interested in strategies that maximize or minimize the probability that a system starting in state s, follows a sequence in Δ. We assume that a strategy does not depend on past history. A strategy is formally defined as follows:

Definition 4.1.4 (Strategy). *A strategy η is a set of conditional probabilities $\mathcal{Q}_\eta(i \mid \sigma_0, \sigma_1, \cdots, \sigma_n)$ such that $\sum_{i=1}^{k_{\sigma_n}} \mathcal{Q}_\eta(i \mid \sigma_0, \sigma_1, \cdots, \sigma_n) = 1$, for all $n \in \mathcal{N}$, $\sigma_0, \sigma_1, \cdots, \sigma_n \in \Sigma$ and $1 \leq k_{\sigma_n}$.*

A strategy η basically resolves the nondeterministic choices of a system that starts at σ_0 and reaches σ_n following the sequence $\sigma_0, \sigma_1, \cdots, \sigma_n$, by choosing the next-state distribution $p_i^{\sigma_n}$ with probability $\mathcal{Q}_\eta(i \mid \sigma_0, \sigma_1, \cdots, \sigma_n)$. The probability $Pr_\eta(t \mid \sigma_0, \cdots, \sigma_n)$ that a direct transition is taken to state t from state σ_n next is thus equal to $\sum_{i=1}^{k_{\sigma_n}} \mathcal{Q}_\eta(i \mid \sigma_0, \sigma_1, \cdots, \sigma_n).p_i^{\sigma_n}(t)$

Therefore with each finite sequence $\sigma_0, \sigma_1, \cdots, \sigma_n$ starting at the root of Ω_s, we can associate the probability $\prod_{i=0}^{n-1} Pr_\eta(\sigma_{n+1} \mid \sigma_0, \cdots, \sigma_n)$. These probabilities for finite sequences give a unique measure $\mu_{s,\eta}$ on $\mathcal{B}_\sigma$ that associates with each $\Delta \in \mathcal{B}_\sigma$ its probability $\mu_{s,\eta}(\Delta)$. The minimal and maximal probabilities can now be defined as follows:

Definition 4.1.5 (Minimal and Maximal Probability). *The minimal and maximal probabilities $\mu_\sigma^-(\Delta)$ and $\mu_\sigma^+(\Delta)$ of a set of sequences $\Delta \in \mathcal{B}_\sigma$ are defined by:*

$$\mu_s^-(\Delta) = \inf_\eta(\mu_{s,\eta}(\Delta)) \qquad \mu_s^+(\Delta) = \sup_\eta(\mu_{s,\eta}(\Delta))$$

Thus $\mu_s^-(\Delta)$ and $\mu_s^+(\Delta)$ are the minimal and maximal probabilities with which the system follows an evolution $s = \sigma_0\sigma_1, \sigma_2 \cdots$ when the nondeterministic choices are as unfavorable or favorable as possible.

The formalism presented here allows us to map a state-transition graph to a stochastic system model such as a DTMC, CTMC or MDP, allowing us to leverage analysis techniques for these representations. We also show how with the help of an appropriate temporal logic, we can specify qualitative as well as quantitative properties that represent survivability concerns adequately.

4.1.2 Modeling Real-Time

In order to describe quantitative performance properties of real-time systems, we present a standard augmentation of a PNS called a Timed PNS (TPNS), first introduced by de Alfaro [39]. This model introduces a special operator that expresses bounds on the average time between two events, and defines a framework for modeling and expressing performance, reliability, and correctness properties of discrete time probabilistic systems.

This probabilistic realtime system corresponds to an MDP with finite state space. In addition to the non-determinism and probabilistic behavior already described in a PNS, this formalism assigns each action from the set of actions associated with a particular state in the model to a *cost* which is interpreted as the amount of time elapsed during the action. The cost of an action in the model presented by de Alfaro can be either 0 , corresponding to immediate actions, or 1, corresponding to unitary time steps. We present the formal definition of a TPNS next:

Definition 4.1.6 (TPNS). *A TPNS over set* $\Lambda = (\Sigma, \sigma_{in}, Act, \kappa, p, c)$ *over AP where*

1. Σ *is a finite state space. Every state* $\sigma in \Sigma$ *assigns truth value* $\sigma[\![x]\!]$ *to every symbol* $x \in AP$.
2. σ_{in} *is an initial state in* Σ
3. A *is a finite set of actions*
4. κ *is a function that associates each* $\sigma \in \Sigma$ *with a nonempty set* $\kappa(\sigma) \subseteq Act$ *that can be taken at* σ.
5. p *is a probability distribution function such that for all* $\sigma, \sigma' \in \Sigma$, *and* $a \in \kappa(\sigma)\, p(\sigma'|\sigma, a)$ *is the probability of a transition from* σ *to* σ' *under action* a. *We require* $\Sigma_{\sigma' \in \Sigma}\, p(\sigma'|\sigma, a) = 1$ *for all* $\sigma \in \Sigma$ *and* $a \in \kappa(\sigma)$.
6. c *is a cost function such that* $c(\sigma, a) \in \{0, 1\}$ *for all* $\sigma \in \Sigma$ *and* $a \in \kappa(\sigma)$. *This is the cost of performing* a *at* σ, *equal to the elapsed time.*

Given a state $\sigma \in \Sigma$, the successor state of σ is chosen in a two step process, similar to the behavior of a PNS:

1. An action $a \in Act$ is selected nondeterministically

2. A successor state σ' is chosen according to probability $p(\sigma'|\sigma, a)$.

Iterating this process gives the set of behaviors of a TPNS, defined as follows:

Definition 4.1.7 (Behavior of a TPNS). *A behavior of a TPNS* Π *is an infinite sequence of states and actions* $\omega : \sigma_0 a_0 \sigma_1 a_1 \cdots$ *such that* $a_i \in \kappa(\sigma_i)$ *and* $p(\sigma_{i+1}|\sigma_i, a_i) > 0$ *for all* $i \geq 0$. *Given a behavior* $\omega : \sigma_0 a_0 \sigma_1 a_1 \cdots$, ω_i *denotes state* σ_i *and* ω_i^a *the action* a_i, *and* $\omega_{\geq i}$ *the behavior* $\sigma_i a_i \sigma_{i+1} a_{i+1} \cdots$

Let Ω_s be the set of of behaviors starting from any state $s \in \Sigma$. Let $\mathcal{B}_\sigma$ be the σ-algebra of measurable subsets of Ω_s. With each Δ in Ω_s, we would like to associate its probability measure $\mu(\Delta)$. This measure is not well defined, as the probability that a behavior $\omega \in \mathcal{B}_\sigma$ belongs to Δ depends on the criterion by which actions are chosen in each state.

Similar to the notion of strategies in PNSes, the concept of a *policy* is used to specify the criteria by which actions are chosen in a TPNS. A policy η is a set of conditional probabilities $Q_\eta(a|\sigma_0 a_0 \sigma_1 \cdots \sigma_n)$ where $a \in \kappa(\sigma_n)$. Starting from $s = \sigma_0$ of Ω_s, after a finite prefix $\sigma_0 a_0 \sigma_1 \cdots \sigma_n$, action $a \in \kappa(\sigma_n)$ is chosen with probability $Q_\eta(a|\sigma_0 a_0 \sigma_1 \cdots \sigma_n)$ according to policy η.

The probability of a direct transition to σ' after $\sigma_0 \cdots \sigma_n$ is given by:

$$Pr_\sigma^\eta(\sigma'|\sigma_0 a_0 \sigma_1 \cdots \sigma_n) = \textstyle\sum_{a \in \kappa(\sigma_n)} p(\sigma'|\sigma_n, a).Q_\eta(a|\sigma_0 a_0 \sigma_1 \cdots \sigma_n) \quad .$$

These transition probabilities now give rise to a unique probability measure μ_s^η on $\mathcal{B}_\sigma$. $Pr_s^\eta(A)$ is the probability of event A in Ω_s under policy *eta* and probability measure μ_s^η.

In this simple model, time is modeled by the values of a fictitious global clock by integers in $\mathbf{Z}$. In order to understand the operational semantics of this system, we introduce the reader to the abstractions of a timed transition system (TTS) [5]. At any point in the execution sequence ω of a TTS, either the system state changes or the clock value changes, usually by a unitary time-step or "tick", or neither. A timed state sequence $\rho = (\sigma, T)$ consists of an infinite sequence of states $\sigma_i \in \Sigma$, $i \geq 0$, and an infinite sequence T of corresponding time values $T_i \in \mathbf{Z}$, that satisfies the following conditions:

- **Bounded monotonicity**: For all $i \geq 0$, either $T_{i+1} = T_i$ or $T_{i+1} = T_i + 1$ and $\sigma_{i+1} = \sigma_i$. This property ensures that time never decreases.

- **Progress**: For all $i \geq 0$, there is some $j > i$ such that $T_i < T_j$. This is to ensure that time never stagnates. Thus there are infinitely many clock ticks in every timed sequence.

This progress property defines what is called a **Non-Zeno** system. Formally, a TPNS Π is non-zeno if a behavior from σ_{in} follows infinitely many time-steps under any policy, i.e., $Pr^{\eta}_{\sigma_{in}}(A) = (\sum_{i=0}^{\infty} c(\omega, \omega_i^a) = \infty) = 1$. A system designer must take care to ensure that their TPNS model is Non-Zeno.

To generate an execution sequence of a TPNS, at each step we have to either choose a transition with a certain probability without incrementing time, or increment time by 1 while taking an idle transition. The bounded monotonicity and progress properties are necessary, but not sufficient to show that the stepwise execution of a TTS cannot lead to a situation where time cannot advance ever. Henzinger et al [68] present additional conditions on the operationality of timed systems and prove how these conditions are sufficient to prevent descriptions of systems where time cannot advance ever.

The authors describe these operationality requirements using the concept of a *partial computation*. A finite prefix $\rho = (\sigma, T)$ of a timed state sequence is a partial computation if it satisfies what are called the initiality, consecution, lower bound, and finite upper bound conditions. The initiality condition asserts that the initial state of this sequence is an initial state of the system model. The consecution requirement asserts that the system can always take some transition from a given state σ_i to a new state σ_{i+1}. The lower bound and finite upper bound requirements on the transition specify how if we are at a given state and many transitions apply, then there is a lower bound on when at least one transition will be enabled. Once a transition is enabled, it will taken by the model within a finite time. If each partial computation of a timed transition system is a prefix of an initialized computation of the system, then we can guarantee that the a system that generates partial computations incrementally cannot arrive in a situation in which the progress condition on time cannot be satisfied.

The choice of $\mathcal{Z}$ as the time domain allows us to discretize time and implement scalable analysis techniques. However, Alur and Henzinger show how we can include non-negative real numbers, denoted by $\Re$ by using the notion of *intervals* to discretize time. These intervals define the duration of system states similar to the function of c presented in the TPNS abstraction. Each interval is

a convex subset of $\Re$. Every interval is of the form $[a,b], [a,b), [a,\infty), (a,b]$ or (a,∞), where $a \leq b$ and $a, b \in \Re$ for the left endpoint a and the right endpoint b. An interval I is *singular* iff it is of the form $[a,a]$, that is the interval is closed and its right endpoint is the same as its left endpoint. Two intervals I and I' are *adjacent* iff (1) either I is right-open and I' is left-closed, or I is right-closed and I' is left-open, and (2) the right endpoint of I is the same as the left endpoint of I'. An interval sequence $\overline{I} = I_0 I_1 \cdots$ is a finite or infinite sequence of intervals that partitions the real line such that:

1. Any two neighboring intervals are adjacent, and
2. For all $t \in \Re$, there is some interval I_i with $t \in I$.
3. I_0 is left closed and the left end point of I_0 is 0. The last interval of any finite interval sequence is unbounded.

Using this notion of intervals, we now define the operational semantics of such a system. The behavior of the system is defined by a set of timed state sequences that satisfies the finite variability property and are fusion-closed. We define these properties next. Each timed state sequence $\tau \in \mathcal{T}$ represents a system behavior by identifying a unique system state $\tau(t) \in \Sigma$ with every time instant $t \in \Re$. Formally a timed state sequence τ is a function from $\Re$ to Σ that satisfies the *finite variability* condition.

Definition 4.1.8 (Finite Variability). *There exists an interval sequence $\overline{I} = I_0 I_1 \cdots$ such that throughout each interval I_i, the values of the propositional variables in a state do not change.*

Thus the finite variability condition asserts that in any bounded interval of time, there can only be finitely many observable events or state changes.

The set $\mathcal{T}$ of timed state sequences is *fusion-closed* if each system state contains all the information necessary to determine the future evolution of the system.

Definition 4.1.9 (Fusion-closed). *For all timed state sequences $\tau_1, \tau_2 \in \mathcal{T}$ and time instants $t_1, t_2 \in \Re$, if $\tau_1(t_1) = \tau_2(t_2)$, then $\tau \in \mathcal{T}$ for the timed state sequence τ with $\tau(t) = \tau_1(t)$ for $t \leq t_1$ and $\tau(t) = \tau_2(t + t_2 - t_1)$ for $t > t_1$.*

Using the notion of intervals allows us to discretize continuous time and incorporate it into models of state-transition systems.

In the next subsection, we describe how we can use these models to extend the language of temporal logic to represent both probabilistic and real time properties of survivable systems.

4.1.3 Temporal Logics for PNSes and TPNSes

In this subsection, we present the syntax and semantics of different temporal logics that can be used to specify quantitative temporal and probabilistic survivability properties. These logics are all extensions of existing formalisms for specifying qualitative branching time properties using CTL or CTL^*. The syntax and semantics of CTL and CTL^* are presented in Appendix A. These logics include a special **probability** operator that can be used to specify the maximal or minimal probability that an event happens. In terms of semantics, the truth value of a formula ϕ at some state σ is a value $p_\sigma(\phi)$ in the interval $[0, 1]$. This can be interpreted as the probability that the formula ϕ holds when the system starts in state σ.

The logics $pCTL$ and $pCTL^*$ [19] or probabilistic CTL and CTL^* can describe branching time properties of **concurrent** Markov chains. Concurrent Markov chains allow a choice between probability distributions on successor states, modeling nondeterminism. This choice is understood to arise the context of a distributed computation and is made by a scheduler or an adversary. The logics $PCTL$ and $PCTL^*$ [65] or Probabilistic CTL and CTL^* describe branching time properties of **sequential** Markov chains, which are basically discrete time deterministic Markov chains. The logic CSL or Continuous Stochastic Logic describes properties of CTMCs. $TPCTL$ or Timed Probabilistic CTL [4], $PBTL$ [13] or Probabilistic Branching Time Logic, and pTL^* or probabilistic temporal logic [39], introduce a special operator **D** to express bounds on the average time between events and allow us to specify quantitative as well as probabilistic branching time properties.

We now present the syntax and semantics of $pCTL^*$, $pCTL$ and $PCTL$ and $PCTL^*$. Similar to CTL and CTL^*, we can specify either **path** formulas or **state** formulas. Path formulas can describe properties of execution paths in the computation tree of a model. State formulas are useful to specify branching time properties as well as various possibilities in a state. The probability operator **P** is only defined on state formulas and is used to express the **quantity** of paths that

satisfy a given formula from a given state. The **P** operator may be viewed as a quantitative version of **A** and **E** which express the universality or existence of (a) path(s) that satisfy a given formula. When the probability value is 1 is equivalent to **A**, and when it is nonzero it is equivalent to **E**.

Syntax: The syntax of $PCTL^*$ formulas is specified as shown next. In the following production rules, ϕ denotes a state formula and ψ denotes a path formula:

$$\begin{aligned} \phi &= \mathbf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{P}_{\bowtie_p}(\psi) \\ \psi &= \phi \mid \mathbf{X}\psi \mid \psi_1 \,\mathbf{U}\, \psi_2 \mid \psi_1 \,\mathbf{U}^{\leq k} \psi_2 \mid \psi \wedge \psi \mid \neg\psi \end{aligned} \tag{4.1}$$

The operators $\mathbf{X}, \mathbf{U}$ stand for the usual next and until temporal operators, and $a \in AP$. The probabilistic operator $\mathbf{P}_{\bowtie_p}(\psi)$ expresses the quantity of paths that satisfy formula ψ, where $\bowtie$ stands for $<, \leq, \geq, >$, and $p \in [0, 1]$. The bounded Until formula quantifies the number of states k that ψ_1 has to hold until ψ_2 holds.

The syntax of the $pCTL^*$ formulas is specified as:

$$\begin{aligned} \phi &= \mathbf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\psi \mid \mathbf{E}\psi \mid \mathbf{P}_{\bowtie_p}(\psi) \\ \psi &= \psi \,\mathbf{U}\, \psi \mid \mathbf{G}\psi \mid \mathbf{F}\psi \end{aligned} \tag{4.2}$$

The operators **A**,**E**,**G**, and **F** stand for all paths, there exists a path, globally and finally respectively. The logic $PBTL^*$ is shown to be essentially equal to $pCTL^*$. Logics $pCTL$ and $PCTL$ are restricted subsets of their starred version, with a limitation that the temporal operators **G**, **U**, and **X** have to be applied to every subformula.

Semantics: The formulas presented above defines a satisfaction relation $\Pi, \sigma \models \phi$, indicating that the state formula ϕ holds in state σ of PNS Π. This definition uses a probability space as described previously by $(\Omega_s, \mathcal{B}_\sigma)$. For a sequential Markov chain, in the case of $PCTL^*$, the formula $\mathbf{P}_{\bowtie_p}(\psi)$ means that the unique measure of the set of paths satisfying the formula $\psi \bowtie p$. For a

concurrent PNS and $pCTL^*$, we have minimal and maximal probabilities depending on the strategy of the adversary.

The syntax and semantics of CSL [11] is very similar to that of $PCTL$ with the addition of one new state formula operator $\mathbf{S}_{\bowtie p}(\phi)$ called the steady-state operator. Steady state probabilities refer to the system behavior in the long run, after the transients have died down. Also, the bounded until formula $\mathbf{U}^{\leq I}$ is now modified to specify a restriction on the bound using an interval I over reals instead of an integer.

Timed probabilistic logics include the operator $\mathbf{D}_{\bowtie d}$ in addition to $\mathbf{P}$ in the generation rules for state formulas. The intuitive meaning of $\sigma, \Lambda \models \mathbf{D}_{\bowtie d}$ in a TPNS Λ is that $\mathbf{D}_{\bowtie d}$ holds at $\sigma \in \Sigma$, regardless of the policy, if the TPNS reaches a ϕ state in average time $\bowtie d$. This definition relies on the fact that the TPNS is non-Zeno. For exact semantics of $TPCTL^*$ and $TPCTL$ for sequential Markov chains we refer the reader to [4], and to [39] for semantics of pTL and pTL^* for concurrent Markov chains.

In the next section, we describe how to specify a recovery strategy and present examples of survivability formulas using the logic presented here.

4.2 Recovery Strategies

We define recovery strategies formally and describe how we can include them in state-transition graphs of access control behavior. In Section 4.2.1, I show how to model an attack as the behavior of an adversary and describe how to differentiate this from authorized system behavior. Next, I describe how we can specify controllers who can counteract adversarial behavior. I also present how we can encode this interaction between a controller and an adversary with respect to a property that needs to be preserved, as a controller-synthesis problem in the context of an open reactive environments [80].

In Section 4.2.2 I show with the help of examples how we can use the formalism presented so far to describe useful survivability properties. I discuss how to compare and contrast different strategies in terms of their impact on the satisfaction of confidentiality and integrity policies. Using these metrics, I describe how to analyze different recovery strategies for information compromise. I focus on specification and analysis of recovery strategies for DoS attacks in Chapter 5.

I follow this with a discussion of how to validate or verify these properties within the framework of the models presented in this Chapter in Section 4.2.3. I show how there exist decidable algorithms to model-check different temporal logic-formalisms for finite-state models and briefly discuss whether run-time monitoring verification techniques can also be applied to analyze finite traces of an infinite-state system, providing weaker satisfaction semantics.

4.2.1 Adversaries and Controllers

We identify two types of users in the context of recovery strategies: adversaries and controllers. An **adversary** is a user who can cause an insecure transition to occur as a result of their actions. The impact of their action(s) on the security of the system may not be instantaneous. A **controller** is a user who can initiate response actions and attempt to counteract the behavior of an adversary. In the case of information compromise attacks, adversaries are users that deliberately insert access rights that contradict system policy. A controller, e.g., a superuser with the appropriate authorizations, can delete these rights from the access rights matrix if they are detected, before information can be compromised.

In the case of DoS attacks, adversaries can send (bogus) requests to networked servers and compete with legitimate users for the resource. A controller can install filter rules to drop these requests before they interfere with request-response behavior of legitimate users of the system. However the act of filtering itself may increase the response times for legitimate users in the system. We study this behavior in Chapter 5.

A recovery strategy is a state transition subgraph that augments an existing state-transition graph of system behavior. Typically, a recovery strategy refines the behavior of a particular edge in a state-transition graph that corresponds to insecure behavior.

Definition 4.2.1. *A recovery strategy* $T = (G_T, E_T)$ *augments a Kripke structure* $M = (\Sigma, \sigma_0, R, L)$, *a PNS* $\Pi = (\Sigma, \sigma_0, \tau, L)$, *or a TPNS* $\Lambda = (\Sigma, \sigma_{in}, Act, \kappa, p, c)$ *with a subgraph* $G_T = (V, E_V)$ *and* v_0, V_f *and a set of edges* E_T *where:*

1. V *is the finite set of states in the strategy.*

2. E_V *is the transition relation between these states.*

3. *v_0 is the first state in the strategy and corresponds to the state reachable from a state in Σ where the strategy is grafted. This edge to v_0 typically encodes the behavior of an adversary that executes an insecure transition to a potentially inconsistent state $\in V$.*

4. *$V_f \subseteq V$ is the set of final states in the strategy that connect the strategy back to the original state-transition graph. Edges from states in V_f lead back to states in Σ.*

5. *E_T includes the transitions between states in Σ to v_0 as well as transitions that connect back states in V_f to Σ.*

The new system model now has $\Sigma' = \Sigma \cup V$ and $R' = R \cup E_T \cup E_V$. From all states in the strategy, it must always be possible to reach a state in V_f with one or more transitions. We introduce two boolean variables to indicate whether the state corresponds to an adversary or a controller. If boolean variable ADV is true in a state, then the request issued from that state is by an adversary. If the variable CON is true in a state, then the action corresponds to a controller's behavior. Therefore states in the augmented graph that correspond to the strategy are now labeled with these variables to indicate whether the action is initiated by and adversary or a controller.

Given this model of an augmented state-transition graph, it is now useful to explore what types of properties can be preserved by strategies. Since a strategy models the behavior of the system under attack, it is useful to ask if it is always possible to reach a state where some security or consistency property can be asserted, by describing appropriate atomic propositions that need to hold in that state as a result of using the strategy over an unaugmented graph. Furthermore, this property can be refined by quantifying probability and realtime measures on paths that need to be satisfied.

We now present an example of how to specify survivability and recovery. During the process of analyzing a specification and identifying potential attacks, it is useful to validate that the model is capable of good behavior. An example of a formula in the context of the extended access control model in Chapter 3 is presented next. Consider the following survivability property that asserts that it is always possible for some legitimate access request to receive an appropriate response. This can be encoded for some model M using CTL as follows:

$$M, \sigma_0 \models \mathbf{AG}(\mathbf{EF}(\sigma_i^{T1} \rightarrow \sigma_{i+1}^{T1}))$$

This property asserts that it is always possible for some requests in the system to execute correctly. Note this measure of survivability is extremely weak, and only guarantees the existence of a path in the model where this semantically consistent request-response transition can occur.

With the introduction of a recovery strategy, we want to strengthen this guarantee as follows:

$$M, \sigma_0 \models \mathbf{AG}(\mathbf{AF}(\sigma_i^{T1} \rightsquigarrow \sigma_{i+k}^{T1}))$$

This specification asserts that it is always possible that if we issue a legitimate request in the system, it will produce an appropriate (semantically consistent) response through one or more intermediate transitions, modeled here by $\rightsquigarrow$. Note that this assertion is stronger than the **AGEF** assertion presented previously because of the universal quantification on the paths. It also allows branching behavior between the two endpoints of the transition. All paths through these states must lead to the desired state.

The existence of a path, or the fact that all paths now can reach a specified desirable state is a qualitative property of the survivability of an augmented state-transition graph. In Section 4.2.2, I show how we can specify quantitative properties to bound the amount of time before this desirable state is reached, and evaluate different strategies based on probabilistic and real-time behavioral characteristics described by the underlying transition graph.

So far, we have presented how to specify recovery strategies, assuming we know how to recover from a given security attack. However, we believe that it more useful to automate this process and investigate whether it is possible to synthesize a recovery strategy given an attack description. For this we need the description of a "recovered" state or a formula that encodes a property that needs to be preserved by the system augmented by the strategy. Using the set of all possible actions of adversaries and controllers, if we have a finite state system, for CTL (and CTL^*) Kupferman et al [80] show how if a property-preserving strategy exists we can always find it, or show that it cannot exist, and this is 2EXPTIME-complete (resp. 3EXPTIME-complete) in the size of the model. We propose to explore the automated generation of recovery strategies for specific problems as future work.

4.2.2 Defining Survivability Properties

In this section we present examples of quantitative survivability properties including DoS-free behavior.

The two survivability properties presented in Section 4.2.1 i.e., **AGEF** and **AGAF** represent two ends of the spectrum with respect to survivable models of system behavior. As shown in Chapter 3, the Until operator **U** is also useful to specify the propositions that should hold in a computation subtree or along a path before a recovery action restores the system to a desirable state, without compromising the security of the information. When we incorporate stochastic behavior and real-time into these models, the corresponding probability and timing guarantees replace the **A** and **E** path quantifiers wherever appropriate. These properties are summarized next:

- **Probabilistic Survivability Properties**: The operator $\mathbf{P}_{\bowtie p}$ where $\bowtie = <, \leq, >, \geq$ represents a quantity of paths in the model that satisfy the corresponding formula. Examples of probabilistic survivability properties include $\mathbf{AGP}_{\bowtie p}\mathbf{F}(\phi)$ that asserts that along every computation path, with probability $\bowtie p$ it is possible to get to a state where ψ holds, and $\mathbf{P}_{\bowtie p_1}\mathbf{GP}_{\bowtie p_2}\mathbf{F}(\phi)$ which asserts that on a probability $\bowtie p_1$ number of paths, it is possible to get to a state where ψ holds with probability $\bowtie p_2$, etc.

- **Real-Time Survivability Properties**: The operator $\mathbf{D}_{\bowtie d}$ represents a bound on real-time that can elapse before the property is satisfied. For example, the formula $\mathbf{AGD}_{\bowtie p}\mathbf{F}(\phi)$ asserts that along every computation path in the model, it is possible to get to a state where ψ holds within time $\bowtie d$. Similarly, the formula $\mathbf{D}_{\bowtie d_1}\mathbf{GD}_{\bowtie d_2}\mathbf{F}(\phi)$ asserts that along all paths, in time $\bowtie d_1$ it is possible to get to a state where ψ holds in time $\bowtie d_1$. P and D operators can also be combined to specify probabilistic timing guarantees.

These examples illustrate how the extended temporal logics can be useful in increasing expressive power of survivability properties. The examples presented so far are useful to describe information compromise attacks and recovery. We now present two properties that we claim are useful to model recovery from DoS attacks:

1. **DoS Resilience**: In order to specify the ability of a model to recover from DoS attacks, we propose the following formula:

$$M, \sigma_0 \models \mathbf{AG}(\sigma_i^{T1} \rightsquigarrow \mathbf{EF}(\sigma_{i+1}^{T1}))$$

which asserts along some path in the future, a request from a legitimate client will eventually get a response, after going through possibly many intermediate states and transitions (represented by $\rightsquigarrow$). This property does not put a bound on the waiting time and is equivalent to a Finite Waiting Time (FWT) property.

2. **DoS Resistance**: In order to resist DoS attacks all-together we need the following guarantee:

$$M, \sigma_s \models \mathbf{AG}(\sigma_i^{T1} \rightarrow \mathbf{AF}(\sigma_{i+1}^{T1}))$$

which asserts that along all paths in the future we can guarantee that every request will always receive a response in the next state. Therefore a model that can satisfy this property is DoS-free.

Once again, the existence of a DoS-free path is too weak to be useful in practice. The DoS-resistance property on the other hand suffers from being too strong. In Chapter 5 I show how we can model more interesting quantitative properties of recovery from DoS attacks.

4.2.3 Validating Survivability Properties

In this subsection, we explore what techniques are available to validate survivability properties within the framework of their corresponding system models. Formulas written in CTL for Kripke structures and in $PCTL$, $pCTL$ and $TPCTL$ for special types of PNSes and $TPNS$es all have efficient *model checking* algorithms.

Before we summarize known results with respect to the complexity of these techniques, we briefly introduce the reader to the differences between model checking and traditional theorem proving for property verification in system models. Traditionally, theorem provers were employed to facilitate the process of verifying whether a system's behavior can satisfy a property specification. A theorem prover annotates different stages (not necessarily each state) in a system's behavior with formulas that can be asserted at that stage, and uses an underlying deduction system with appropriate axioms to prove if the final property specification can be proved as a theorem in this system. The limitations of traditional theorem-proving systems are well-known. According to Halpern and Vardi [66], the first problem is finding an appropriate language to represent these assertions. The

need to use logic to model the behavior of the system necessitates the use of expressive logics. Theorem proving is difficult to automate for arbitrary logics, and undecidable for even first-order logic.

Model checking on the other hand works with a **semantic** model of the system directly. In contrast to proof-theoretic approaches, the problem of property validation is reduced to checking whether a given formula is true in the model. In this thesis we extend access control models with more expressive semantics. We believe that using model checking can be an appropriate choice for verification in this framework.

The paradigm of model checking was first explored in the context of finite-state program verification. The problem was to verify if a finite state program P satisfies a specification ψ in some temporal logic. One way of doing this is to completely characterize the program [89] by a temporal logic formula ϕ_P, i.e., ϕ_P describes all possible transitions of P in each possible global state. Checking whether P satisfies the property specification ψ can be accomplished by checking if the implication $\phi_P \Rightarrow \psi$ is valid. However this validity problem for temporal logic is known to be EXPTIME-complete [53].

Clarke, Emerson et al [32, 52, 112] proposed the following approach: Instead of representing P by a formula, they explored the use of a semantic state-transition model of P called the Kripke structure M_P, where the states represent the possible global states of P and the transitions the evolution of the model over time. The model-checking approach attempts to find the set of all states Σ in the model that satisfy ψ, i.e., $\{\sigma \in \Sigma | M_P, \sigma \models \psi\}$. The system satisfies the specification provided (all) the initial state(s) are in this set. Note that the size of the model is essentially the same as the length of ϕ_P, and a maximal model is exponential in size of the set of boolean variables in the system. The model checking problem can however be solved in time linear or polynomial, depending on the logic, in the size of M_P and ψ. Another added advantage of model checking is that it always produces a **counter-example** if a particular specification cannot be satisfied in the model, as a path consisting of states and transitions to a state that cannot satisfy the specification. Recent research has expanded the scope of model checking algorithms to probabilistic and real-time state models.

One limitation however is the fact that model-checking techniques only apply to finite-state

systems and this may seem to be severe limitation as most reasonably complicated systems cannot be described as finite-state models. However techniques such as abstraction, partial-order reduction, composition, exploiting symmetry, and structural induction [33] can all reduce the complexity of model-checking infinite-state systems by transforming them into finite-state models, with some loss of expression in the semantics.

We now summarize known results for the complexity of model checking branching time formulas ψ for specific temporal logics and models of reactive systems in Table 4.1:

Model	**Specification Logic**	**Complexity**
Kripke structure	CTL	$O(\|\psi\|.(\|\Sigma\| + \|R\|)$ [33]
Sequential PNS	$PCTL$	Linear in ψ, polynomial in Π [81]
Concurrent PNS	$pCTL$ or $PBTL$	Linear in ψ, polynomial in Π [19]
Non-Zeno Concurrent $TPNS$	pTL	Linear in ψ, polynomial in Λ [39]

Table 4.1: Model Checking Branching-Time formulas

Model checking PNSes and TPNSes is an active area of research. Many of these algorithms have efficient implementations based on symbolic manipulation of boolean formulas using what are called Ordered Binary Decision Diagrams (OBDDs). Model checking has been shown to be feasible for models with up to 10^{20} states.

An alternative to theorem proving or model checking for validating survivability properties is **runtime verification**. Both theorem proving and model checking aim at proving a model is correct or that satisfies a property specification before their execution. In order for these techniques to be automatable, they need to be finite state systems. Runtime verification on the other hand is the application of lightweight formal methods applied during the execution of a system. These techniques rely on observation and monitoring of finite execution traces, of either finite state or infinite state systems. Successful examples of the use of runtime verification include race-condition detection and deadlock-detection. Runtime verification techniques extend the scope of models that can be analyzed for different properties, but provide weaker guarantees.

However, there are various system properties that cannot be expressed as sets of runs, including possibilistic and general branching-time properties. As such, the application of runtime verification to liveness properties is infeasible. However, some recent results for Timed Linear Time Logics may

be useful for validating bounded liveness properties [79]. We propose to explore this in the future.

In the next section we discuss the impact of changing access control policies on confidentiality and integrity policies in the system, and discuss how we can extend our survivability analysis to address specific concerns in this context.

4.3 Analysis of Degradation of Access Control Survivability

In this section, we focus our attention to studying the impact of an attack that compromises an user account in the system. When a user's account is compromised, it is important to analyze what effect this has on the security of **other** users and objects in the system. We are interested in investigating how actions that can be initiated by a compromised user can affect existing integrity and confidentiality guarantees as the system evolves, and in exploring how this compromise can spread insecure or inconsistent information through the system.

We start with a snapshot of the consistent system state just before such an attack occurs. At this point in time, the system contains a finite number of subjects and objects. Let these sets be $S = \{s_1, s_2, \cdots, s_n\}$ and $O = \{o_1, o_2, \cdots, o_m\}$. The access control policies in the system are represented by access matrix entries of the form $A[s_i, o_j] = r$ that correspond to a confidentiality policies and entries of the form $A[s_i, o_j] = w$ that represent integrity policies.

For our analysis, these policies can be represented as a directed graph (digraph). A read permission is modeled as an edge that is directed between the object and the subject, and the write permission as an edge between the subject and the object. For example if s_1 can read o_1, then the digraph $G_P = (V_P, E_P)$ will contain edge (o_1, s_1). If s_1 can also write to o_1, then edge $(s_1, o_1) \in E_P$. We now define the notion of an information modification path, analogous the definition of an information transfer path presented by Biba [20], in this graph.

Definition 4.3.1 (Information Modification Path). *An information modification path is a sequence of nodes* $s_{i1}o_{j1}s_{i2}\cdots o_{ik-1}s_{ik} \in V_P$ *such that* $(s_{ij}, o_{ij}) \in E_P)$ *and* $(o_{ij}, s_{ij+1}) \in E_P$ *for* $1 \leq j < k$.

An information modification path or a write-read path encodes the explicit flow of modified information in the system. When a subject s_i is compromised in the system, a masquerading user

s' can:

- Read confidential information that s_i could read, or
- Write over information that s_i could write.

By reading information s_i was allowed to read, attacker s' has violated the confidentiality of all objects $O_C = \{o_j|(o_j, s_i) \in E_P, 1 \leq j \leq m\}$. Furthermore any subjects in S that can also read objects in O_C, i.e., $S_C = \{s_k|(o_j, s_k) \in E_P, o_j \in O_C, 1 \leq k \leq n\}$ also have their confidentiality compromised. A user who was not allowed to read the files these subjects were allowed to read, is now able to do so as a result of the attack. If an object o_p in the system cannot be read directly by s_i or if subject s_q in the system does not have any files that can also be read by s_i, then the confidentiality of this object or subject is not compromised.

However, attacker s' can cause further damage to the system by writing to objects that s_i has write permissions for. Once the integrity of the information has been compromised, this information can spread throughout the system by anybody who can read the tainted object. The initial set of objects whose integrity can be compromised include all objects in $O_I = \{o_j|(s_i, o_j) \in E_P, 1 \leq j \leq m\}$. The initial set of subjects who can be compromised include all subjects in $S_I = \{s_k|(s_k, o_j) \in E_P, o_j \in O_I, 1 \leq k \leq n\}$. This compromised information can affect the integrity of all write commands issued by subjects in S_I, and can further spread along the information modification paths from S_I.

Graph $G_P = (V_P, E_P)$ can now be explored for all information modification paths starting in set S_I. Each object and subject encountered along this path can be added to sets O_I and S_I to form the set of all subjects and objects whose integrity can be affected as a result of the compromise. Finding these paths reduces to the digraph **reachability** problem and the **transitive closure** problem for digraphs, which can be solved in size polynomial to the number of vertices ($O(V^3)$) in the graph and whose lower bound is given by the theorem that it is no easier than the matrix multiplication problem.

Given a user account that is compromised, we can use this transitive closure analysis to come up with the maximal sets S_C, O_C, S_I, O_I, and change the permissions of user s_i to reduce this damage. Given a policy configuration, we can apply this analysis by simulating the effect of

an attack for each user in the system and come up with a ranking of which users are better targets for attackers, depending on the cardinality of these maximal sets. Given a set of integrity and confidentiality policies, we are interested in finding minimal directed cuts (or dicuts) that increases the number of connected components in the graph. By removing these edges, explicit but unauthorized modification of information can be efficiently curtailed.

In the case of buffer overflow attacks, we can apply this analysis to policy configurations that attempt to minimize the impact of an attacker by restricting the address space and permissions associated with user-level processes running as root (e.g., `chroot` jails). In this situation, automated reachability analysis can help us identify the sets of users and resources that can be compromised as a result of a successful buffer overflow attack and validate our designs.

One of the issues with creating these `chroot` jails is that processes cannot be isolated completely from each other, especially because of the need to use many common system libraries. Administrators usually decide on what is shared and what is duplicated in an ad hoc manner. Given a digraph of read-write permissions, we can apply other graph-theoretic analysis techniques such as finding cutsets and cutpoints that increase the number of components in a graph and reduce dependencies among these user sets. We propose to investigate applications of the analysis framework presented here in the future.

In the next section, we discuss how to model trust into the abstractions presented so far and explore the implementation complexity of changing access control policies dynamically and maintaining consistency.

4.4 Dynamic Access Control

When an object (file or program) in the system becomes *vulnerable* to compromise as a result of the discovery of a new flaw, or when a subject cannot be trusted, changing the access control matrix is often a suitable course of action for system administrators as a preventive measure. Often, the required software updates may not be immediately available, and changing the authorizations for subjects and objects can prevent malicious users or software applications from accessing vulnerable (but not yet compromised) resources, and vice-versa, thereby reducing the threat of attack.

When an actual *attack* occurs (information exposure), the situation is less amenable to recovery

by dynamically changing the access control policies. If attacks can be detected, either by intrusion detection or other mechanisms, these techniques can still be used to isolate compromised parts of the system and contain damage.

In traditional access control models, restrictions are placed on the set of subjects that are allowed to add, update, or delete access rights from the access matrix, to implement different types of access control policies. Mandatory Access Control (MAC) policies restrict this privilege to trusted system administrators. In Discretionary Access Control (DAC), this privilege is extended to the owner of an object. Most systems support a combination of MAC (for public or system resources) and DAC (for privately-owned resources) policies.

In Sections 4.4.1 and 4.4.2, I examine the issue of changing access rights dynamically, with special attention to how they are implemented in practice, i.e., using access lists (ALs) and capability lists (CLs). Once a threat is mitigated, e.g., by expelling the users, or by installing updates, it is also useful to be able to restore the original access control policies, i.e., to rollback the system to its original operating environment. We also explore the costs of changing ALs and CLs for rollback and recovery. Most of the work presented here has appeared in [102].

Changing the access matrix in an ad hoc manner can have unexpected side effects in terms of safety and trust assumptions. In Section 4.4.3 we show that in order to preserve access control consistency during change, we need to implement an atomic broadcast and commitment protocol in a distributed setting. It is well known that achieving this is difficult without synchronization assumptions. However, partial consistency and recovery may be implementable with lesser effort in many cases, to keep the system running, even under threat of attack. In the same section, I describe how we can change trust assumptions safely during recovery, by augmenting policy specification and enforcement mechanisms with appropriate guards.

4.4.1 Implementing Access Controls

In a distributed system, shared resources (i.e., objects) can be on different physical machines connected over a network, and the access control enforcement mechanisms can also be distributed across the network. Each machine may have a reference monitor that intercepts both local and remote access requests to shared resources. A single centralized access control matrix to validate

accesses is rarely implemented in practice [21]. In order to reduce performance overheads, these access rights-sets are also spread across the system.

Two different representations for storing the rights across the system are commonly used: access lists (ALs) and capability lists (CLs). ALs are lists of $\langle subject, method \rangle$ pairs per *object*, and correspond to the list of subjects that are allowed to access specific methods for a given object. CLs are lists of $\langle object, method \rangle$ tuples, and correspond to the list of objects and methods that can be accessed per *subject*.

Traditionally, in stand-alone systems where the sets of subjects (usually classified into groups or alternatively as roles) or objects do not change very often, the relative benefits of choosing one representation over the other are comparable [42, 28]. In an AL-based system, invoking a subject's right to access an object is simple and is usually accomplished by deleting specific rights from the object's AL. To locate an access right one may have to search through the entire AL. CLs are usually generated by system administrators and stored in protected shared memory. Each access request can either carry the capability itself (with sufficient cryptographic protection to prevent modification) or a pointer to the location of the capability in the CL, which is only accessible by the decision logic. CLs simplify the lookup of rights and speed up the process of access control decisions. When capabilities are passed around or CLs replicated in different process address spaces, revocation of access rights becomes difficult.

It is not immediately clear which implementation mechanism is ideal to change the access rights in response to a perceived threat or vulnerability in a distributed setting, since there are no studies that compare the relative benefits of the two approaches. In [75] the authors speculate that ALs are more popular than CLs because they efficiently answer the question "who access a given object?", whereas CLs are useful to answer the question "what else can a subject access?" [21]. While the first question is useful to protect the confidentiality of information accessed and directly relates to access control, the second question is useful to evaluate the flow of information in the system.

In Section 4.4.2, our aim is to evaluate the performance overheads of changing access control rights in a distributed setting for both options.

4.4.2 Changing Access Control Rights

When a system is attacked or when a new vulnerability is discovered, administrators may receive notifications (either from other administrators, network monitoring software, software vendors, intrusion detection systems etc.) to disallow access to certain users, computers, software applications, or specific methods for applications (e.g., disable execute for email attachments). I examine the cost of changing these rights for both AL and CL-based distributed access control implementations in terms of the sizes of sets of subjects, objects, and methods. We note that these actions can guarantee that *future* compromise is prevented, as any future trace of behavior will be denied this action by the policy-enforcement mechanisms.

We assume that each subject s_i has an AL and object o_j has a CL associated with it for the same of comparison. Let $|\mathcal{S}|$ and $|\mathcal{O}|$ be the cardinalities of the sets of subjects and objects, equal to the number of CLs or ALs in the system respectively. Table 4.2 summarizes the maximum number of lists that have to be processed (ALs or CLs), in order to remove a user, a resource, or a specific access right tuple from a distributed access control system. The numbers also correspond to the maximum number of network connections that have to be initiated by the administrator in order to decide whether an AL or a CL needs to be updated. The actual amount of time required to process the list depends on the data structures used to implement the lists, and the number of access rights in the system.

Recovery Action	**AL model**	**CL model**
To remove user	$\|\mathcal{O}\|$	1
To remove object	1	$\|\mathcal{S}\|$
To remove a specific access right	1	1

Table 4.2: Maximum Number of Lists Processed

As shown in Table 4.2, to remove a user s_i in a distributed AL-based implementation, since the $\langle s_i, r_k \rangle$ tuples are distributed across multiple access lists corresponding to different objects, each list must be examined in turn and the tuples purged appropriately. In order to remove a user in a CL-based implementation, only the capability list of a single user has to be deleted. In both cases, the maximum number of entries that may need to removed is equal to the number of distinct

$\langle s_i, o_j, r_k \rangle$ tuples in the system. This is also the number of entries that need to be stored for rollback in order to restore the original access rights.

To remove an object o_j and all its associated rights, only that object's AL has to be removed from the system. In the case of CLs, all CLs corresponding to all subjects have to be examined, to find and delete the appropriate$\langle o_j, r_k \rangle$ tuples. In both cases the number of entries that need to be removed (or stored for rollback) is equal to the number of distinct access rights that have o_j in their tuples.

To remove an individual permission for a particular object (e.g., disable the auto-execute option in a web browser etc.), in both AL and CL-based implementations, only one table has to be updated, corresponding to either the s_i's CL or the o_i's AL.

To summarize, in order to remove an object's access rights from a distributed system, *there is a significant cost asymmetry in favor of ALs.* Similarly to remove a user from the system, using CLs is more efficient.

From this analysis, we observe that in terms of automating the process of changing access controls dynamically by removing entries from the ALs or CLs, if the sizes of the sets of users and objects are comparable, no implementation technique has a clear advantage. If we expect recovery-actions as primarily removing access to certain objects or disabling specific object rights, even temporarily, clearly ALs are better. However, in the case of buffer overflow attacks, curtailing a subjects' rights is more important and CLs are better.

Using RBAC

CLs are more efficient when system administrators discover that certain user accounts are compromised and want to sandbox a user, isolating their actions from the rest of the system. In contrast, in an AL based implementation, many lists may have to be located and updated. However, ALs can overcome this limitation if we use an aggregation mechanism such as Role Based Access Control (RBAC [56, 119]) to simplify administration of users and rights.

In RBAC, users can be associated with one or more roles. Each role is a placeholder for a set of permissions. The permissions consist of objects and methods that are authorized for that role. Users can be added and removed from roles, independent of the updates to the role-permission

assignments. This asynchrony simplifies the management of large sets of users and restricts their behavior according to their "role" in an organization.

RBAC is very flexible and can be implemented naturally using ALs. In addition, it can also be used to implement both MAC and DAC policies. Instead of specifying individual subjects and permissions in each AL for each resource, we can aggregate entries according to roles. This has the effect of reducing the size of the ALs, and reducing the search space for changing access controls. CLs can also be organized according to object-roles [35]. This reduces the total number of CLs in the system.

Instead of removing a user from a role-based AL, the user's access control permissions can be changed by revoking the user's current role and assigning the user to a special pre-defined role that has no access rights, or to a role that reduces the user's permissions to a restricted set of rights. This information can be relayed directly to the policy enforcement mechanisms (e.g., the reference monitors), who must use the user's new role and disallow any requests made by the user using their old role. While this eliminates the cost of changing the ALs, the communication cost of disseminating the user's new role has to be taken into account.

Another advantage of using pre-defined roles during response actions is the ability to formally analyze the access control behavior of the system a priori, even when the permissions of a user change dynamically. By restricting the permissions during dynamic state-changes in the system, it is possible to determine what guarantees can be made by the dynamic behavior of the system by formal analysis.

In the next subsection, I examine the possible side-effects of changing ALs and CLs as recovery actions. I also describe techniques to augment existing specifications to overcome these limitations.

4.4.3 Enforcing Safety and Preserving Trust

One of the problems with changing access controls dynamically is the need to synchronize operations. In distributed systems, an administrator may initiate a recovery-action to change access controls over the network. Depending on the type of changes requested, the administrator may have to update many ALs or CLs, distributed across different machines. As a result, some lists may get updated faster than others and the access control safety property many not be consistently

enforced across the system. As we show in the next subsection, maintaining consistency in this situation is not easy.

Changing access rights dynamically to alter the access control behavior of a system can have unexpected side-effects. In a stand-alone operating system, or in a homogeneous distributed system (e.g., Unix-based or Windows clusters), it may be possible to preserve existing trust assumptions by relying on the underlying protection model. Implementing protection domains is simplified by this support, and only network administrators can change access control lists or permissions. In a heterogeneous distributed system, the trust assumptions and trust validation have to be modeled explicitly into the system specifications, to prevent undesirable side-effects. We explore this in Section 4.4.3.

Enforcing Access Control Safety

In order to remove users, objects or particular methods from the system, a system administrator has to keep track of the different ALs or CLs stored in different parts of the system. Removing objects in a distributed AL-based system, or subjects in a CL-based system while maintaining safety is straightforward. The access rights in a particular object or user's AL or CL are unique, and are not replicated across the system. Therefore, no consistency issues arise.

However, the problem emerges when an administrator needs to modify multiple ALs to remove subjects (or CLs to remove objects) in a distributed setting, and keep the lists consistent. A non blocking atomic commitment protocol [100] (such as a modified two-phase commit) is required to ensure that updates are consistent. To guarantee timeliness, when multiple update requests are sent, maintaining safety is reduced to implementing an atomic broadcast protocol [100], which needs to be both reliable, as well as deliver the update messages in total-order.

It is well known that there are no deterministic atomic broadcast algorithms for asynchronous systems. This is because the distributed consensus problem can be reduced to atomic broadcasts. However, there are many schemes [100] that account for clock drifts and periodically send out synchronize messages that work under the assumption of bounded drifts. The two protocols viz., non-blocking atomic commitment and atomic broadcast, can guarantee that the access controls are consistently enforced across the system even when the ALs and CLs change dynamically. The

protocols may introduce a non-negligible overhead to change policies and their effectiveness as recovery actions must be evaluated in terms of these overheads.

In many cases, it may be desirable to implement a weaker notion of consistency. Partial consistency can still satisfy availability guarantees for particular users and objects in the system.

As mentioned earlier, RBAC can overcome the need to change access lists, but revoking a user's role and assigning the user to a new role requires similar consistency and synchronization guarantees.

Preserving Trust Assumptions

In addition to preserving consistency, trust assumptions related to changing ALs and CLs have to be examined carefully. Trust assumptions are incorporated into the access control model by including the concept of authorization as follows: allow access if and only if an authorized user added the access right to the system.

Enforcing this modified safety property is straightforward in a stand-alone system, or in a homogeneous distributed system (multiple machines running the same OS). Most existing distributed operating systems automatically provide support to enforce that the access control mechanisms can only be updated by authorized users (administrators in MAC, and owners in DAC). However, these mechanisms can be compromised by masquerading users, as in buffer-overflow attacks.

One way to circumvent an unauthorized user from changing these controls is by restricting the address space visible to a process running as superuser, on behalf of a subject with user privileges. Even if the process is compromised, the user should not have access to the authorizations required to change the access controls. However, we have to provide a legitimate user enough authorizations to execute this task normally. Balancing these requirements can be a significant challenge as highlighted in Section 4.3.

Another way of ensuring this condition is met is by requiring strong authentication. Furthermore, every time entries in the ALs and CLs need to be changed, to ensure that trust assumptions can be validated, users in the system can be challenged to produce a proof of authorization.

We describe a systematic technique to augment policy specifications with special clauses called guards that force users to present a proof of authorization, in the form of credentials, attesting that

they have the right to change the access rights. This mechanism is independent of the underlying protection model. If we can guarantee that these credentials cannot be obtained by masquerading users, then the trust assumptions are always preserved by any proper implementation of the specification. The problem now reduces to **separating** the authorizations from identity information in a request. A superuser may store these credentials in a smart-card, for example, and thereby prevent a masquerading user from changing the access rights. Ideally, we want to encode the "separation of duty" principle with these checks to ensure that a single malicious user cannot change these rights even if he or she were able to compromise the mechanisms.

We use the formalism of the HRU model, presented in Chapter 2, but augment the set of conditions from Table 2.1 with authorization proofs. Our protection state is still defined by the triple $\langle S, O, A \rangle$ where S is the set of subjects, O the set of objects and A is the access matrix. We include the set $\mathcal{R}$ of object methods that correspond to privileges in the HRU model. The set A is typically the union of different ALs or CLs in the system.

Note that the HRU model does not include any authorization checks in its definitions if primitive operations. If this system was implemented according to the specification, anybody is allowed to change the access rights matrix. As mentioned earlier, different sets of subjects are allowed (or authorized) to create and delete users, objects and methods. In a DAC system, users are allowed to create and own objects and add access rights to objects they own. For example, if $user1$ owns $file_{user1}$, then $user1$ can insert $\langle user2, file_{user1}, read \rangle$ into A. In an MAC system, users and objects can be added only by administrators.

Next, we present a method to automatically add guards to the primitive operations or protection state transitions and preserve trust during the modification of access control implementations.

To preserve the trust assumptions, we augment an access control specification with special proofs of authorization. Henceforth, if a user wants to change an entry in A, the user is required to produce a proof attesting that he or she is allowed, by some trusted authority, to actually execute the primitive operation. The policy enforcer (e.g., a reference monitor) has to be suitably modified to check this proof. The proof check can be verified non-interactively, and needs to be decidable. This proof-checker is a guard, similar to Dijkstra's guarded commands [45, 121], and these guards can be applied to both ALs and CLs without loss of generality.

One way of generating a proof of authorization is by using an attestation from a trusted administrator that gives the holder of the attestation the capability to change an AL or CL entry, i.e., the permission to call a method to change the entry. This type of capability (also called a license [129]) or credential is an attestation of trust. These attestations should be protected against modification by unauthorized entities. They can be made unforgeable by the issuer by attaching a cryptographic digital signature [96]. The signature should tie in the name of the issuer and the intended recipient to prevent modification and also provide non-repudiation of ownership.

Generating these credentials naïvely can cause management problems. Consider the set S of users who can issue signed capabilities, the set O of shared objects in the system and the set $\mathcal{R}$ of methods corresponding to access rights. The set of all licenses that can be presented to the policy manager in this system is exponential in size and is given by $C \subseteq S \times 2^{OX\mathcal{R}}$.

A user can have many different credentials authorizing some or all methods with respect to a particular object and may present any subset of these to the enforcer to change an access right. The enforcer needs to decide whether the decision is consistent with the trust management implications of these attestations and this may be non-trivial. For example, the monotonicity of the privileges available after revocation may have to be maintained [129] to prevent undesirable behavior.

Instead, we argue that in the case of MAC or DAC policies, two simple types of credentials are sufficient to attest the identity of the entities (primarily subjects) and the ownership of one entity by another. These credentials should not be delegated and should not available to any entity except the actual owner of these access rights. Satisfying these requirements may be a significant challenge.

Let $\mathcal{C}$ be the set of *typeof* and *owns* credentials. An examples of an identity credential is $typeof(Alice, administrator)$ that asserts that the identifier *Alice* is an administrator. The credential $owns(object, method)$ or $owns(user, object)$ attests that the method is "owned" or exported by the object or the object is owned by the user, respectively. We generate one credential per object and method. Therefore, the size of this credential set is equal to the number of unique access rights in the system. This simplifies the management of credentials and proof verification, though it may increase the number of credentials a user has to present.

In Table 4.3, I present how we can augment the primitive operations of the HRU model with

proofs of authorization to enforce DAC policies, where only the owner of an object can enter or delete access rights into the access matrix. The credential $owns(s, o)$ checks for ownership and the credential $owns(o, m)$ is an integrity check. Note we need to explicitly identify the subject s who issues the request to the specifications of the **create subject**, **create object**, **destroy subject** and **destroy object** operations. The effect of the operation, on the protection state of the system is the same as in Table 2.1.

op	conditions
enter r into $A[s, o]$	$s \in S$ $o \in O$ $\mathbf{owns}(\mathbf{s}, \mathbf{o}) \in \mathcal{C}$ $\mathbf{owns}(\mathbf{o}, \mathbf{r}) \in \mathcal{C}$
delete r from $A[s, o]$	$s \in S$ $o \in O$ $\mathbf{owns}(\mathbf{s}, \mathbf{o}) \in \mathcal{C}$ $\mathbf{owns}(\mathbf{o}, \mathbf{r}) \in \mathcal{C}$
create subject s' **issued by** s	$s' \notin O$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$
create object o' **issued by** s	$o' \notin O$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$
destroy subject s' **issued by** s	$s' \in O$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$
destroy object o' **issued by** s	$o' \in O$ $o' \notin S$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$

Table 4.3: Authorizations for DAC

For MAC policies, only administrators are allowed to change the protection state. In Table 4.4 we show how these authorizations are specified for adding and deleting rights, which are restricted to system administrators. The conditions for other primitive operations are the same as for DAC. This specification mechanism can also be easily extended for RBAC.

From the augmentations to the specifications, we claim that if the credentials are generated correctly and the administrators keys are not compromised, then the state transitions allowed in our modified system have the required authorization proofs necessary to guarantee the trust assumptions are preserved, even when the implementations are changed dynamically. Care must be taken to ensure that the mechanisms do not allow delegation or theft of these credentials. If *all* the guard conditions cannot be satisfied, the state of the system is unchanged. The protection

op	conditions
enter r into $A[s, o]$	$s \in S$ $o \in O$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$ $\mathbf{owns}(\mathbf{o}, \mathbf{r}) \in \mathcal{C}$
delete r from $A[s, o]$	$s \in S$ $o \in O$ $\mathbf{typeof}(\mathbf{s}, \mathbf{admin}) \in \mathcal{C}$ $\mathbf{owns}(\mathbf{o}, \mathbf{r}) \in \mathcal{C}$

Table 4.4: Authorizations for MAC

state is allowed to change only when the proof of authorization is verified correctly.

4.4.4 Applying Dynamic Access Control to Dynamic Environments

When the sets of subjects and objects do not change frequently, though the overheads of changing access controls in AL-based and CL-based systems seem equivalent, we observe that ALs are better when the changes involve updating rights for specific objects and object methods. In this section, we explore the ideas further in the context of dynamic environments where the sets of users and objects can change dynamically, and the access control matrix entries have short life times and may be updated frequently. In this situation, we find dynamic access controls are an important feature of the system design rather than just a mechanism to enable RAs. In such cases, the performance overheads for changing the implementations can have a significant impact on the design decisions. In this subsection, I describe briefly how we can use our analysis to justify the choices for contrasting AL-based and CL-based implementations of dynamic access controls.

Examples of dynamically changing environments include ad hoc networks, active networks, and smart spaces. In an ad hoc network, for example, where users bring in their own computers and connect together, it is often useful to associate ALs with the mobile resources themselves. Since different users enter and leave over a short period of time, it is not feasible for the participating subjects to carry CLs for all possible objects. Instead, the participants themselves can build dynamic trust relationships and assign permissions to each other by updating their own ALs.

In contrast, when the set of subjects can change over time, but the resources themselves remain more or less fixed, it is more efficient to use CLs. We have explored such a solution in the context of

active networks in our previous work [26, 103] and developed a CL-based architecture to enable the dynamic installation and update of policies in real-time, to accommodate different sets of subjects using the same resources for different active network protocols over time. Our security architecture for active networks also incorporated the access control safety and authorization techniques discussed in this paper, in the framework of a CL based implementation. With this architecture, we were able to demonstrate how we can change the access control policies on software routers dynamically, and deploy a host of reactive security countermeasures, including dynamic firewalls and vaccines, without sacrificing safety guarantees [85, 26].

Another area where we have explored the issues of changing access controls dynamically is in the context of smart spaces or active spaces [116]. An active space is a physical environment that is augmented with computing and communication resources and can be programmed and configured automatically to support different tasks and activities. An example of an active space is a smart room with many display and computation devices that can be configured as a meeting room, a lecture room, or a recreation room etc., only by changing the software in the room. In this context, we have developed an AL-based solution to address the access control issues in this room, where we have dynamically changing sets of users and to a lesser extent, objects. We chose an RBAC variation of ALs to scale to a large number of users.

4.5 Chapter Summary

We present our extended state-transition model of reactive system behavior that incorporates probabilistic and timing behavior in this Chapter. This model enriches the expressive power of traditional access control models and allows us to extend the nature of scope of security properties to model the survivability of a system under attack. We present examples of survivability properties as temporal logic formulas within the framework of this model, and show how we can leverage existing techniques such as model checking to validate these properties.

In the next chapter, we focus on the network DoS and related DDoS problem and show we can apply the techniques presented here to describe and analyze different DoS attacks and DoS prevention strategies.

"Can you do addition?" the White Queen asked. "What's one and one and one and one and one and one and one and one and one and one?" "I don't know," said Alice. "I lost count."

Lewis Carroll, Through the Looking Glass

Chapter 5

Denial of Service

In this chapter, I explore the Denial of Service (DoS) problem, and show how I can extend our modeling framework to describe DoS attacks and attack mitigation strategies. I focus on scenarios where resources become unavailable due to resource exhaustion by requests originating from both legitimate and compromised users.

Network Denial of Service (DoS) attacks are a classic example of such a problem, and have frustrated the efforts of network engineers in their quest to build resource-access and sharing mechanisms that are both efficient and adequate to enforce authorized usage. Many of the DoS vulnerabilities of network service access stem from sacrifices made by designers to reduce performance penalties, including coarse-grained accounting, which make the resources readily available to both legitimate and malicious users alike. The DoS problem arises when a malicious user, or set of users exploit these sharing mechanisms and monopolize access, or contend with legitimate users for shared network resources, thereby denying service to legitimate users.

In the context of bandwidth exhaustion attacks, in order to flood a multi-hop network between a single or a small set of DoS attackers and a network server, the attacker has to do almost as much work as the victim server, pumping packets on to the network continuously to keep the victim busy. It is usually easy to characterize such attacks by monitoring bandwidth utilization and attribute the attacks to the originators. Once the origin of the attack is detected, it can be effectively neutralized by installing filters and throttling these attackers at source or on intermediate routers. Since most servers have fat incoming pipes and most clients have thinner outgoing pipes, and since attackers could be identified easily, such attacks were not considered a major threat in the past.

In recent years, a new type of automated network DoS attack, called the Distributed DoS or

DDoS attack has surfaced. According to the 2002 CSI/FBI Computer Crime and Security survey, respondents to their survey who could quantify their losses reported a loss increase of 350% due to such attacks. DDoS attack scripts are widely available and can be downloaded and launched against any type of Internet server with minimum effort on the part of the attacker. These tools scan the Internet to identify vulnerabilities on network clients and install software attack daemons, called zombies, on compromised hosts, thereby distributing DoS attackers over several hundreds or thousands of hosts.

Each zombie sends a low-bandwidth flood, consisting of different packet types including ICMP, UDP, TCP SYN, or HTTP packets towards the victim [49, 48, 47, 46]. These floods are coordinated by a time-trigger to maximize their collective impact, and are typically issued by a mastermind daemon, over an encrypted channel. Many tools add multiple levels of indirection to both control and attack by creating layers of control daemons called masters. The cost of mounting these attacks *per attacker*, amortized over the hundreds of attackers, is negligible in comparison to the victim's cost in processing these packets. These low-bandwidth floods are virtually undetectable at source and in most intermediate networks. A DDoS victim is forced to process these packets at the expense of requests from legitimate clients of the service.

In terms of modeling survivability to DDoS attacks, the stateless nature of the IP-based Internet makes it difficult to characterize flows and reason about, or even prove the effectiveness of DDoS attack countermeasures. In recent years, researchers have developed a variety of strategies to tackle different aspects the network DDoS problem [99]. These include better authentication to prevent unauthorized users, or better accounting to traceback and locate attackers, as well as bandwidth regulation and filtering mechanisms to prevent malicious users from sending unauthorized requests to access resources in the first place. The difficulty of obtaining quantitative models of DDoS attacks and server behavior is seen as a significant challenge to understand better the impact of these strategies. To the best of our knowledge, no tools are available to validate the effectiveness of such strategies or even experiment with different strategies and compare their relative merits.

As mentioned in [69], one of the challenges with modeling or simulating DDoS attacks is the amount of computing resources required to observe, store, and collect statistics about the behavior of thousands of system components such as individual hosts, intermediate routers, and network

servers. While large-volume traces for DDoS attacks are available, the challenge is to analyze this information and come up with meaningful insights. Without such studies however, the effectiveness of anti-DoS strategies cannot be validated with confidence.

Formal methods, on the other hand, can prove to be a viable alternative and provide useful insights, at the cost of some abstraction and information loss. We believe that one of the shortcomings of existing quantitative models of network traffic with respect to DDoS attacks is the lack of a semantic framework to express and reason about temporal properties of attack behavior. We show how we can apply the framework from Chapter 4 to model DDoS attack behavior, analyze the impact of countermeasures, and provide useful insights. In Sections 5.2 and 5.3, I show with the help of examples how we can use the formalism of a PNS model, and specify DDoS survivability properties as temporal logic formulas and evaluate the impact of legitimate and attack traffic behavior on both the DDoS victim and the legitimate clients of a service.

Using our formalism, we can prove estimate the effectiveness of different DDoS prevention strategies. In Section 5.3, Our model exposes the cost-benefit issues of implementing strategies such as stronger authentication and filtering in reducing DoS vulnerabilities.

The techniques we present in this chapter complement quantitative models network behavior, which are often derived from measurement of dynamic performance characteristics. The data from such quantitative models can be used to provide the operating assumptions of our formal models and increase the confidence of the results obtained by our analysis. We show how we can characterize this interaction between qualitative and quantitative properties, and focus our attention on evaluating how attackers and legitimate users can influence each other's behavior with respect to their effect on a DDoS attack victim.

5.1 Modeling DoS Survivability

Shields [122] identifies network DoS attacks as attacks that use network services to disrupt network behavior. These attacks have the effect of causing either consumption or corruption of network resources, making them unusable by legitimate network users. As mentioned earlier, we only focus on resource consumption attacks in this thesis. Other types of DoS attacks include DoS by reservation (e.g., in QoS networks) and DoS by disruption (physical attacks). DoS by reservation

and physical attacks have proposed solutions (e.g., pricing models, physical security) that are not directly within the scope of our study.

Traditionally, DoS is measured by the **waiting time** between a service invocation request and its corresponding response. Unbounded waiting times correspond to absolute DoS. In order to make quantitative comparisons between different strategies in terms of increasing or decreasing a client's vulnerability to DoS, we need to model the response time for a service request explicitly.

Previous research on formal specification and verification of DoS properties [8, 134, 97] focus on showing how resource allocation models for operating system resources are resistant to DoS attacks. These specifications are concerned with modeling constraints on resource access mechanisms that are necessary (and whether they are sufficient) to **prevent** DoS. Yu and Gligor [134] specify different constraints on resource access and sharing mechanisms in terms of fairness, simultaneity, and resource allocation properties. Their work introduces a quantitative measure of DoS resilience in terms of waiting time (WT) policies. Two propose two types of waiting time policies viz., finite (FWT) and maximum WT (MWT). The FWT property is qualitative, whereas an MWT property can be viewed as the bounded availability property. An important result from their work is that in order to prevent DoS, **users** need to accept some additional restrictions on their request behavior in addition the constraints imposed on the sharing mechanisms at the server.

Millen [97] expands on this work and argues for a DoS Protection Base (DPB) similar to a Trusted Computing Base (TCB), with strong trust assumptions to guarantee that these constraints can be reliably enforced. This work also introduces a state-transition model of resource allocation and suggests as future work the use of probabilistic models of resource consumption to model consumption behavior. In effect, both models demonstrate that in order to prevent DoS, resource access mechanisms, as well as user behavior, have to be reliably constrained.

In the context of a DDoS attack, therefore, the property we are most interested in is the **waiting time**, measured as the end-to-end delay observed by a legitimate user of a network server. When this waiting time is unbounded, the server is potentially undergoing a DDoS attack. Traditionally, the **average** end-to-end delay on a network is modeled using queuing theory. Quantitative stochastic models of network behavior are commonly used, often requiring simplifying assumptions, and provide the foundation for delay approximations in networks. These techniques have shown useful

insights, even though it is impossible to obtain accurate quantitative delay predictions on the basis of these models alone.

Traditionally, the waiting time or the delay between a request and response within a communication network is typically modeled by four components [18]:

1. *Processing Delay*: This is the delay between the time a packet is received at a node in the network, to the time it is assigned to an outgoing queue for transmission.

2. *Queuing Delay*: This is the delay between the time a packet is assigned to a queue for transmission, to the time it starts being transmitted.

3. *Transmission Delay*: This is the delay between the time the first and last bits of a packet are transmitted.

4. *Propagation Delay*: This is the delay between the time the last bit of a packet is sent out on the link from the sending node, to the time this bit is received at the receiving node.

This accounting of the time spent by a packet on a network does not factor packet retransmissions. The propagation delay depends on link-characteristics and does not change per packet. The link itself is viewed as a bit-pipe over which a given number of bits per second, called the *capacity* of the link, can be transmitted.

The most commonly used model for allocation of capacity among multiple competing streams of traffic is statistical multiplexing. Under this scheme, packets of all streams are merged into a single queue and transmitted in a first-come first-serve (FCFS) order. A slight variation of this scheme is a system that maintains a separate queue for each traffic scheme and serves the queues in sequence one packet at a time. Statistical multiplexing provides the best performance characteristics for best-effort traffic.

The timing characteristics of the behavior of clients and servers are modeled as stochastic processes with probability distributions describing:

- **Inter-arrival time**: This corresponds to the arrival of a client request at a server, and is modeled as a random variable from a probability distribution of the time between two successive arrivals.

- **Service time**: The corresponds to the time required by the server to process a client's request and is represented by a random variable from a probability distribution of service times.

The goal of analyzing such systems is to estimate the average number of requests in the system that are either waiting in some queue or undergoing service, and use this to estimate the average delay per request. These parameters allow resource engineers estimate analytically what the average waiting time for a client in the system will be, and whether this is acceptable. The average number of requests in system N and the average delay T per request are related by a simple formula known as *Little's Theorem* and has the form $N = \lambda.T$ where λ is the average request arrival rate. This is given by (assuming this limit exists) $\lim_{t\to\infty}(\textit{Average number of arrivals in } [0,t]/t)$.

Many different stochastic models of inter-arrival times and service rates are popular. The simplest of these are based on what is called the "memoryless" property, where successive inter-arrival times and service times are assumed to be statistically independent of each other. The memoryless assumption is controversial. Many empirical studies in the 1990s [107] have shown that Internet traffic is not memoryless, but in fact bursty in nature with long range dependencies in packet inter-arrival and service times.

In recent years however, there is an increasing belief due to the changing characteristics of Internet traffic (no longer dominated by dialup lines and large file transfers or "mice" and "elephants") that the inter-arrival rates are indeed tending towards memoryless distributions, most notably the Poisson [27] distribution. This belief is validated by statistically analyzing large amounts of Internet traffic on stub and core networks. Assuming that the inter-arrival rates can be indeed modeled by the Poisson distribution, we already have a wealth of theoretical results to analyze such systems. We present this theory in Section 5.1.1 to show how Markov chains can be used to estimate average-end-to-end delay.

5.1.1 Background on Delay Analysis

Consider a single-server queue where packets arrive according to a Poisson process with rate λ. Traditionally, request service times are modeled as a memoryless exponential distribution with mean $1/\mu$ seconds. The standard technique to describe the dynamic behavior of such a system is by modeling it as a CTMC.

Each state in a Markov chain model corresponds to the number of requests in the server's queue waiting to receive service. When a new packet arrives, the state of the system changes with the queue size increasing by one. When a packet is serviced, the state changes again with the queue decreasing by one. Given independent identically distributed (IID) inter-arrival times, over a small time interval δ, the state of this chain changes with probability $\lambda.\delta$ that a new packet arrives, and does not change with probability $(1-\lambda).\delta$ initially. Given a packet is being serviced by the server, a packet leaves the server with probability $\mu.\delta$, a new packet may arrive with probability $\lambda.\delta$ and no change occurs with probability $1-\lambda.\delta-\mu.\delta$. These transition probabilities are specified with a small margin of error $o(\delta)$ [18].

In order to estimate the average delay for a request in this model, we are interested in calculating what are called the **steady-state** probabilities of a given Markov chain, that will help us estimate the average number of requests waiting to be serviced in the service queue when the system is in equilibrium. This value can be used to estimate the average waiting times from Little's Theorem. In steady-state, the probability that the system is in some state n (i.e., n elements in its queue) and makes a transition to $n+1$ at the next transition instant is the same as the probability that the system is in state $n+1$ and makes a transition to n, i.e.,

$$p_n.\lambda\delta = p_{n+1}.\mu\delta$$

This equation is called a global balance equation for this Markov chain. Since p_n is independent of δ, taking the limit of the equation as $\delta \to 0$, we obtain:

$$p_{n+1} = \rho.p_n,\ n = 0, 1, \cdots$$

where $\rho = \lambda/\mu$.

Using this equation, we can now find the average number of requests in the system in steady state as:

$$N = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$$

The average delay per request is given by the sum of waiting time in the queue and the service time can be calculated using Little's Theorem as: $T = 1/\mu - \lambda$.

General Service Time Models

When the request service times have a general distribution, not necessarily exponential, but the arrivals can be still modeled as a Poisson process with rate λ we can use what is called the *Pollaczek-Khinchin* formula [18]. This formula gives the expected waiting time for a request in the queue of a single server system. Suppose requests are served in the order they arrive and that X_i is the service time of the ith arrival. We assume these random variables $(X_1, X_2, \cdots)$ are identically distributed, mutually independent and independent of inter-arrival times. Let

$$\overline{X} = E\{X\} = 1/\mu = \text{average service time}$$

$$\overline{X^2} = E\{X^2\} = \text{second moment of service time}$$

The expected request waiting time in the queue W is given by the Pollaczek-Khinchin formula as:

$$W = \lambda.\overline{X^2}/2.(1-\rho)$$

$$T = \overline{X} + W$$

When service times are exponentially distributed, we have $\overline{X^2} = 2/\mu^2$ and reduces to the formula presented earlier. When service times are identical for all requests, $\overline{X^2} = 2/\mu^2$ and:

$$W = \rho/2.\mu.(1-\rho)$$

The techniques presented in this section summarize the theory behind estimation of waiting times for requests in a memoryless analytical model of network behavior. These formulas work well for a single server with multiple requests. However, analysis becomes really difficult when many transmission queues of this type interact in tandem, modeling the behavior of a request passing through heterogeneous intermediate routers where many traffic flows intersect before the request reaches a server. The description of arrival as a stochastic process in a downstream queue in such a network can get very complicated. When packet lengths and inter-arrival lengths are correlated, no analytical results are known for even a tandem queuing of *two* Poisson processes [18].

In the next section, we show how we can model the victim of a large-scale DDoS attack as a PNS using some of the theory presented in this section, and analyze it as a Markov chain. Our analysis is different from traditional models of queuing behavior in one important aspect. Specifically, our model of the state includes qualitative as well as quantitative attributes, which lets us define the notion of useful work, and use this as a measure of the survivability of the server. In section 5.3, I

extend this analysis to incorporate clients and intermediate routers, show how we can estimate the bound on the average delay characteristics, and use this as measure of the survivability of different DDoS prevention strategies.

5.2 Modeling A DDoS Victim

We focus on modeling a DDoS victim as a single server, which serves multiple clients, legitimate and otherwise. These requests arrive at the server via multiple interconnected routers. The model we present here only abstracts characteristics of server behavior that are relevant to resource consumption in the context of a DDoS attack. In the following subsections, I define our model(Section 5.2.1), specify survivability to DDoS attacks from the viewpoint of a server as temporal logic formulas that are meaningful within the context of the model, and analyze an example server specification by varying the parameters to correspond to different situational characteristics in Section 5.2.3.

5.2.1 Modeling a Network Server

We describe the behavior of the DDoS victim server as a sequential CTMC augmented with atomic propositions, as described in 4. Including atomic propositions in the state of CTMC does not impact its stochastic behavior. Subsequently, we show how can use CSL to analyze behavioral characteristics of the model under different adversarial behaviors. To build our model, we observe the following characteristics about a DDoS victim:

- There are two distinct aggregate classes of traffic: useful and unauthorized. We need some semantics in the model to differentiate between requests belonging to these two classes in order to describe the behavior of a server under attack.
- The service time for each packet in the queue depends on the class of the packet being serviced. In a webserver for example, the service rate for processing an HTTP request will be different from the service rate for an ICMP Echo request. This may vary further within a given protocol itself, corresponding to different options in the headers of these requests.

The definition of a DoS attack on a server depends on what constitutes "useful" work in this context. A typical web server serves HTTP requests to web clients. Requests arrive to the server

over the network and are stored in its input queue. In addition to HTTP requests, web servers also respond to other packet types. In particular most servers accept and respond to legitimate ICMP (including ping requests) as well as certain types of UDP datagrams [86]. Whenever the server finds an ICMP or UDP datagram in its input queue, it devotes some amount of time to serve it. In addition to these three types of packets, the server's CPU can also spend time discarding bad packets(incomplete headers, other unrecognizable packet types, unauthorized packets etc.). A simple state-transition graph that captures this CPU utilization behavior of a web server is shown in Figure 5.1.

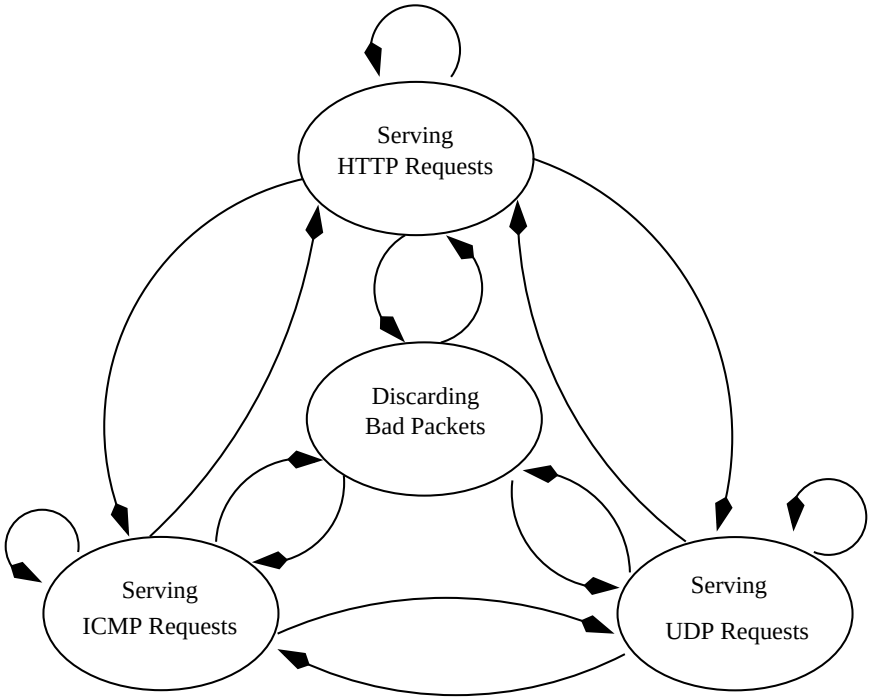

Figure 5.1: State-Transition Graph of Server

We use the PNS framework of Chapter 4 to differentiate between states in the model where a server is processing a useful packet or an attack packet, by modeling this knowledge as an atomic proposition that is true in that state. We first present a simple model of server behavior based on these characteristics with the assumption that the server is capable of classifying a packet as useful or unauthorized upon inspection. Within these two aggregate classes, individual packet flows may arrive at the server at different rates and may be processed at different service rates. Characterizing these rates can be extremely difficult and depend on client behavior, operational parameters of the

server, and the bandwidth of its incoming link.

In the case of a DDoS attack, we observe from empirical data [104, 69] that multi-source DDoS attacks typically saturate the pipe to a server or a subnetwork. A victim server is continuously processing packets from its input queues, with almost zero idle time. The data in [104] illustrates how low bandwidth requests from individual servers suffer from unbounded waiting times, typically larger than their voluntary timeout periods.

We make some simplifying assumptions about the characteristics of request arrival rates and service rates at a DDoS server, given that DDoS attacks are difficult to characterize and analytical models of DDoS traffic are not available. One thing we would like to emphasize however is that our modeling complements quantitative models of network traffic, and provides a framework to reason about the behavior of the system under attack. A better model of a specific server's network traffic can be plugged into our framework in the future and the analysis can follow the general procedure outlined here.

In our simplified model, the service rates at a DDoS server are defined by the memoryless exponential distribution, and are only dependent on the class of the packet and not its origin or its arrival rate, since a new request is always waiting for service. The probability that server moves to a state with a new request in the model is therefore determined by the service rate of its current request class.

The difference in the arrival rates of attack traffic and legitimate traffic determine the probability distribution on the next state of the model. To study the interaction between these arrival rates, we simulate the effect of how increasing amounts of attack traffic can affect the server's ability to do useful work. Later, in Section 5.3 I show how we can estimate some of these probabilities and coarsely approximate the average waiting time for different types of clients in a system, with respect to different DDoS prevention strategies. For now we focus on describing our server with four memoryless parameters: the probability that the next request is legitimate, the probability that it is an attack, and the probabilities that the server will finish processing its current request for both attack and legitimate packet types.

We claim that the server is doing useful work if its computation consists of states where CPU time is spent serving legitimate requests, HTTP or otherwise. For a given **attack profile**, described

by the relative arrival rates of attack packets and legitimate packets, we define this probabilistic behavior as CTMC and show how we can estimate the proportion of time spent by a server doing useful work. This value also depends on the service rates for each packet class. The steady-state analysis of this CTMC will give us a measure of the survivability of the system against an attack profile, as a proportion of total time spent by the process doing useful work versus processing all packets, including any unauthorized packets that it receives

Modeling behavior in this fashion does not address attacks where a group of clients get together and send legitimate HTTP requests to the server with the intention of competing with other legitimate HTTP requests from being served. If the attackers' requests are behaviorally indistinguishable from legitimate requests, this cannot be differentiated in our model. However, a server may be able to differentiate between packets generated by an automatic daemon vs. packets generated by realtime user traffic using spectral analysis as shown in [69].

We now present the CTMC model of a DDoS victim server:

Definition 5.2.1 (DDoS Victim Server). *The CTMC M_{Server} is a tuple $(\Sigma, \mathbf{R}, L)$ where:*

1. *Σ is a finite set of states*
2. *$\mathbf{R} : \Sigma \times \Sigma \rightarrow \Re_{\geq 0}$, the rate matrix*
3. *$L : \Sigma \rightarrow 2^{AP}$, which labels each state with the set of atomic propositions that are valid in that state*

The rate matrix $\mathbf{R}$ characterizes the transitions between the states of the CTMC. If $\mathbf{R}(\sigma, \sigma') > 0$ then a transition from state σ to state σ' can occur. If $\mathbf{R}(\sigma, \sigma') > 0$ for more than one state σ', and if we can assume that the service rates can be characterized as a memoryless distribution (e.g., the exponential distribution), then we can estimate the probability of moving from a state σ to the state σ' as the probability that the delay of going from σ to σ' finishes before the delays of other outgoing edges from σ . Let $\mathbf{E}(\sigma) = \sum_{\sigma' \in \Sigma} \mathbf{R}(\sigma, \sigma')$, the total rate at which any transition from state σ is taken. $\mathbf{P}(\sigma, \sigma') = \mathbf{R}(\sigma, \sigma')/\mathbf{E}(\sigma)$, except if σ is an absorbing state when $\mathbf{P}(\sigma, \sigma') = 0$.

To return to our example, we group the traffic to our webserver into two aggregate classes of packets, labeled as the HTTP class that corresponds to useful work, and a generic Attack class that corresponds to attack packets of any type.

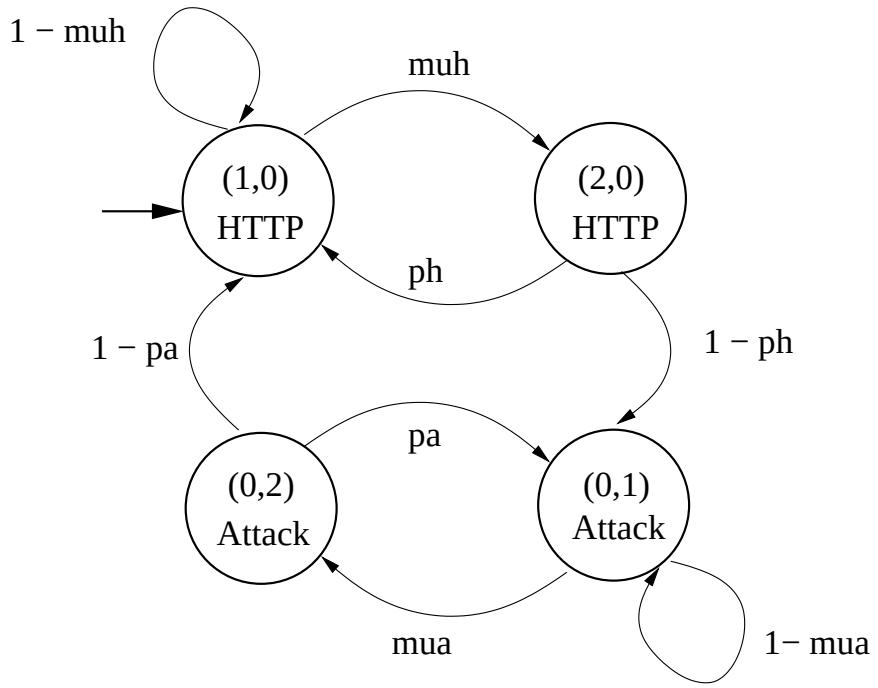

Figure 5.2: DDoS Victim Server as a PNS

Our model consists of two status variables `http` and `attack`. Each variable can take on one of three values $\{0, 1, 2\}$. Therefore the values of these variables define six atomic propositions of the form (`http` $= 0$), (`http` $= 1$), (`attack = 2`) etc. The probability that the next packet is an HTTP packet is given by `ph`, the probability that it is an attack is given by `pa`. The probability that the model stays in its current state processing a HTTP packet is given by `muh` and the probability that it stays in its current state processing an Attack packet is `mua`. These four parameters and the status variables define the PNS presented in Figure 5.2.

Here, state labeled $(1, 0)$ corresponds to the model executing an HTTP request (i.e., `http=1` and `attack=0`). It stays in state $(1, 0)$ with probability `(1 - muh)` and moves to state $(2, 0)$ with probability `muh`, indicating that it is finished processing the request. At this state it can again go to state $(1, 0)$ with probability `ph`, or to an attack state $(0, 1)$ with probability `pa = (1 - ph)`. From this state similarly, the model can move to a state $(0, 2)$ with probability `mua = (1-muh)` or stay in the same state with probability `(1 - mua)`. From state $(0, 2)$, the model can move to $(0, 1)$ with `pa` or to $(1, 0)$ with `ph`. No other transitions are allowed in the model.

A computation of the model starting at state s_{HTTP} can be represented as an infinite tree as shown in Figure 5.3. At each state, any of the two transitions can be enabled as shown, corre-

sponding to the probability of leaving a state in Figure 5.2. From this tree we observe that in addition to paths containing only useful states, there exist valid paths in the model that do not include any HTTP states. If the server only receives a large number of ICMP, UDP, or other badly constructed packets to fill up its input queues, then it may end up not doing any useful work, since HTTP requests do not reach the server's input queue. In fact, automated DoS and DDoS attack scripts generate exactly these types of packets in their attempt to flood [48, 47, 46] the input queue and deny service to HTTP requests. For a given attack profile, we can annotate the transitions with appropriate probabilities and estimate what states are more likely to occur in the model in steady-state.

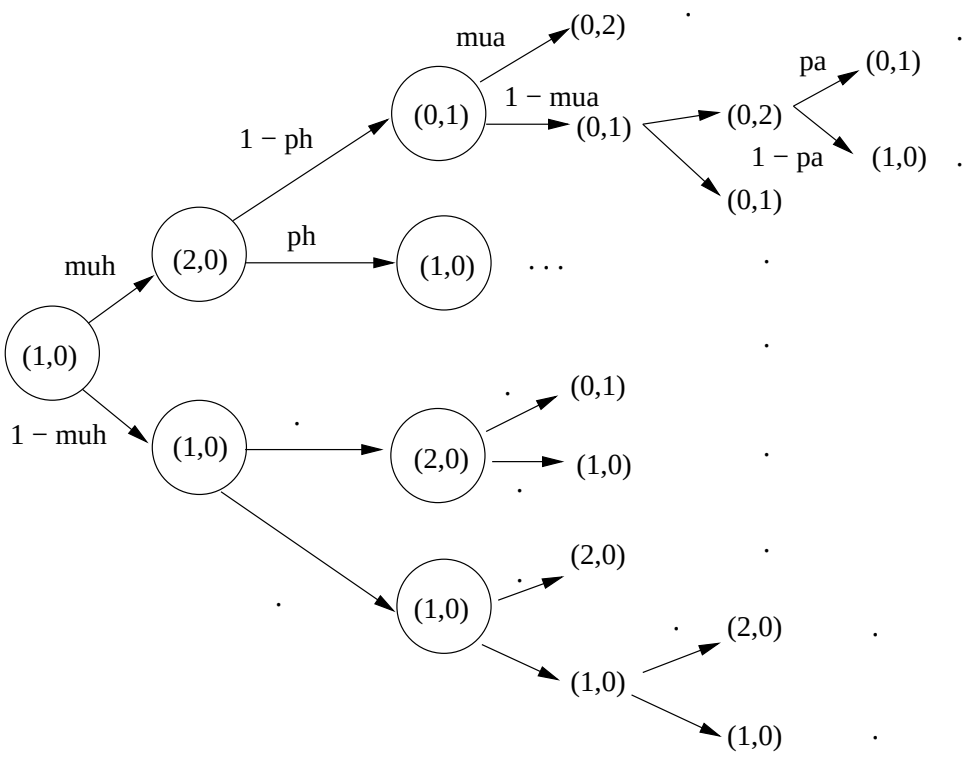

Figure 5.3: Server Computation Tree

In Section 5.2.2, we formalize the notion of a server's ability to always do useful work in terms of CSL properties and show how we can measure this in the CTMC specified here. In Section 5.2.3 we show how we can use our model to study the impact of different DDoS prevention strategies on the server.

5.2.2 Specifying and Verifying Server DoS Properties

The property we are most interested in, given a specific CTMC modeling a DDoS victim server is how the attack traffic rate affects the ability of the server to do useful work. In terms of the model, given an attack profile as `(ph, pa)`, and the service rates `(muh, mua)` we want to estimate how the probability mass flows through the CTMC as time passes.

If the system started in some state $\sigma \in \Sigma$, at time 0 (the probability of being in state σ at time 0 is 1), the vector $\pi^{\sigma}(t) = (\pi^{\sigma}_{\sigma'}(t))_{\sigma' \in \Sigma}$ denotes the probability distribution among the states σ' at time t where t is a non negative real number. The set $\{\pi^{\sigma_0}(t), \pi^{\sigma_1}(t), \cdots, \pi^{\sigma_n}(t)\}$ for an n state CTMC is called the **transient distribution** of the CTMC at time t.

The limiting probability distributions π^{σ} as $t \to \infty$ is called the **steady-state distribution** of σ and always exists for arbitrary finite CTMCs. Obtaining the steady state distribution for a given attack profile gives us the probability that the model will be each of the four different states in steady state. This can be used to measure the survivability of the model to a given attack strategy. We show how to model and compute this steady state probability for useful work with the help of an example.

We pick the probabilities in our example from the measurement data in [69]. This data corresponds to measurements made on attacks captured at Los Nettos, a moderate sized ISP near Los Angeles. The typical packet load on the Los Nettos network is 38K packets/s. During attack this increased to 100K packets/s. A total of 80 large scale DDoS attacks were logged. Approximately 85% of the packets were TCP during normal load, dominated by DNS and web traffic. During attacks, TCP reflection attacks against web servers and FTP servers were the most common packet types.

Since most networking characteristics are tailored for normal load, we pick 38K packets/s as the value for the service rate for useful packets. Most commercial servers come with benchmarks that give average and peak services rates which can be used to estimate this parameter. The peak traffic under attack was 100K packets/s, which we assign as the service rate for attack packets. Under this load, this gives us us `muh`$= 38/138 = 0.27$ and `mua`$= 100/138 = 0.73$. From this measurement data, we only have two types of information: the behavior of the system when no attacks are occurring, i.e., the probability of attack `pa`$= 0$, or the behavior of the system under attack when

the probability of attack is close to 1.

We specify the example as a PNS using a probabilistic model checking tool called PRISM [111]. This tool allows us to analyze systems that exhibit probabilistic behavior. The tool requires two inputs, a description of the system to be analyzed, and a set of properties to be checked against the system. The system to be modeled is described in the PRISM language, which is a simple state-based specification language. Appendix B presents the model we specified using the PRISM language. This program represents the CTMC of Figure 5.2.

The fundamental components of a PRISM program are *modules* and *variables*. A system is composed of a number of modules that can interact with each other. A module can contain a number of local integer-valued variables. The values of these variables at any given time constitute the state of the system. The global state of the system is given by the local state of all the modules. Within a module, commands describe its transition behavior. A command of the form:

```
[] g -> p1:u1 + ... + pn:un;
```

describes a transition with guard `g` as a predicate over the variables of the system. Each update `ui` describes a transition the module can make from its current state if the guard is true. Constants `pi` assign probabilistic information to the transition.

In our model, states $(1, 0)$ and $(2, 0)$ correspond to our notion of useful work. Therefore survivability in this model for different attack profiles can be evaluated by measuring the steady state probability, for a given attack profile, that the server will be in state $(1, 0)$ or $(2, 0)$. This is expressed as the CSL formula:

Definition 5.2.2 (DDoS Server Survivability). *DDoS Server Survivability is given by the following CSL formula :* $M_{server}, (1, 0) \models \mathcal{S}_{\bowtie p}((http = 1) \vee (http = 2))$

In PRISM syntax this is written as `S>p [ http=1 | http=2 ]`. From the measurement data in this section, we have two scenarios, where either `pa = 1.00` corresponding to the case where no useful work is done by the model or to the case where `pa = 0.00`. Validating these formulas for these two scenarios in the model is not very useful. However, in Section 5.2.3 I show how we can use the modeling and analysis framework from this section to analyze the impact of other attack profiles, and show how we can increase or decrease a server's survivability by implementing strategies that change the values of either `pa` or `mua`.

5.2.3 Effectiveness of Different Server DoS Prevention Strategies

In this section, we show how we can use the model from Section 5.2.1 to analyze the behavior of a DDoS server by simulating different attack profiles, changing the values of attack rates and service rates, and study the impact of different DDoS recovery strategies on a server's DDoS survivability.

The analysis presented here can be used by the owner of a server to determine if any DDoS prevention measures can be deployed and estimate the projected impact of deploying these measures. Some of the options for the owner in this case, as suggested in the past, include requiring strong authentication, proofs of authorization, or devising better attack recognition and filtering techniques to drop attack packets faster. Techniques such as strong authentication for DDoS prevention are controversial, since attackers can form bad authentication tokens, and force the server to spend more time discarding them, increasing the end-to-end delay for legitimate users of the system.

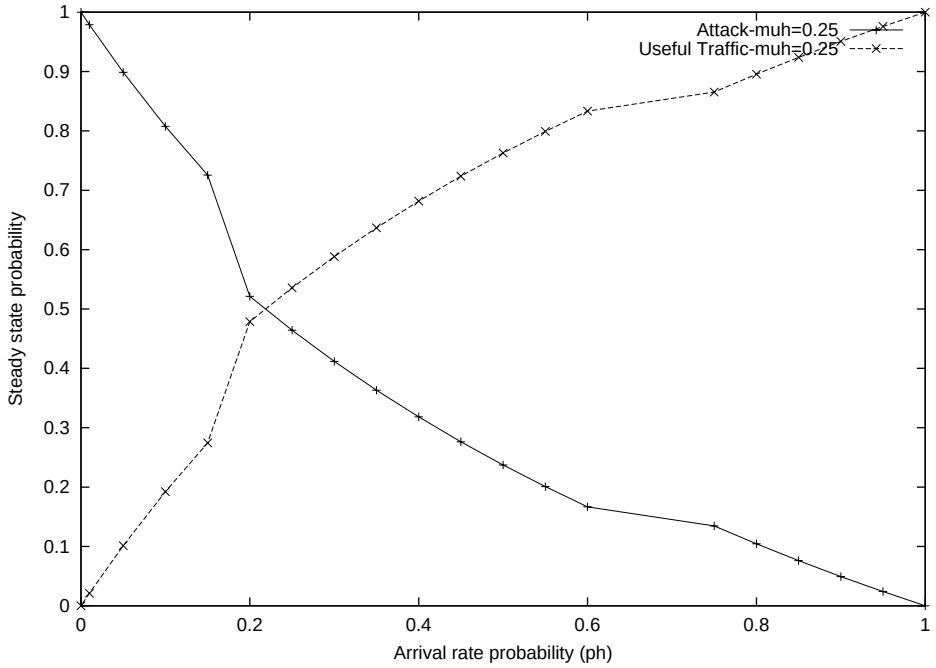

Figure 5.4: Impact of Changing Attack rates for a Fixed Service Rate

We focus on how we can use the analysis framework to estimate the survivability of the example server model presented in Section 5.2.2. In Figure 5.5, we plot the impact of changing the proportion of attack traffic to legitimate traffic on the survivability of the model, given that we have a fixed

configuration for the service rates. We pick `muh=0.25` and change `ph` from 0.00 to 1.00 in steps of 0.05 on the X-axis. On the Y-axis, we plot the computed steady-state probabilities of the server under different attack profiles from the PRISM model. This study highlights the what we term the **vulnerability** of the server to DDoS attacks.

As seen from the graph, the steady state probabilities that the model remains in state $(1, 0)$ or $(2, 0)$ that correspond to useful work, increases as the ratio of the attack packets to legitimate packets decreases. From measurement alone, we were able to obtain the steady-state distribution corresponding to the two endpoints of this graph. With the model however, we can analyze different aspects of the behavior without actually simulating different attack profiles. We observe that as the server's rate increases in proportion to the attack rate, the survivability of the system also increases, because the system spends more time doing useful work per request than in discarding attack packets.

While this result is expected, the graph also gives us the distribution of how this value changes over time. As the proportion of attack traffic drops from 1.0 to 0.8, the steady-state probability that the model spends its time in states corresponding to useful work increases sharply. Even with 80% attack packets, the server spends half its time in steady-state doing useful work. After this point on the graph where the two curves intersect, the slope of the gain in survivability is smaller. This implies that for the given configuration, an attacker has to send packets 80% faster than the legitimate servers to cause a serious degradation ($< 50\%$) in the server's ability to do useful work.

Using the results from this analysis, the owners of the ISP network can estimate the threshold of attack onset more accurately. Monitoring and filtering packets based on per-flow characteristics can be very expensive. In [69], the authors suggest that the threshold rate used to detect attacks is when the aggregate packet rate exceeds 40K packets/s. This measure was determined by observing the characteristics of attack traffic. Whenever the rate exceeded 40K packets/s in the network, it signaled the onset of an attack. Based on the measurement data, this is a reasonable assumption to make.

We contend however that such measurements provide only a coarse-grained view of the attack profile and its potential impact. With our analysis, we suggest that a better estimate on the onset of a damaging attack can be obtained by doing the steady-state analysis. With our coarse-grained

model, we can show that even if the ratio of the attack packets to the regular HTTP, regardless of its current traffic rate is 40%, we can still guarantee that the server will spend more than 80% of its time doing useful work. However, this may come at a cost of decreased throughput for legitimate clients in the system, which cannot be evaluated by modeling the server alone. Once the rate exceeds this threshold, the ability of the server to do useful work degrades according to the graph shown.

This analysis can also be used to focus on what flows to traceback in the case of an attack. Rather than focus on all flows, the server network can concentrate on tracing back and filtering packets on upstream on a small number of flows, depending on what their rate characteristics are, based on how reducing these flows can impact the survivability of the system under attack, according to the analysis from Figure 5.5.

Another option available to the owner of such a system to increase their survivability to DDoS attacks is to decrease the processing time required to discard bad packets, and require that legitimate packets produce an appropriate proof of authorization (e.g., using authentication tokens). This strategy has the impact of reducing the processing time for attackers, at the cost of increasing it for the legitimate users. We show how we can use our methodology to understand these tradeoffs better.

One important observation in order to use such techniques is that characterizing bad packets can be challenging. Many automated DDoS tools randomize packet headers making this problem even more difficult. However, assuming that we can characterize these packets, we can study the impact of changing the service rates for different request classes given a fixed ratio of attack packets to legitimate packets.

In Figure 5.5 we plot `muh` on the X-axis, changing it from 0 to 1 in steps of 0.05. The value of `muh` in the graph represents the relative processing rate of HTTP packets with respect to attack packets. For example when `muh=0.5`, these rates are equal and when `muh=0.25`, useful packets are processed three times slower than attack packets. We plot these curves for different attack to legitimate traffic ratios, viz., `ph=0.85, 0.7, 0.3`. The Y-axis plots the steady state probability that the model is doing useful work, a direct measure of its survivability.

From all three plots that the slower we process attack packets respect to useful packets, the

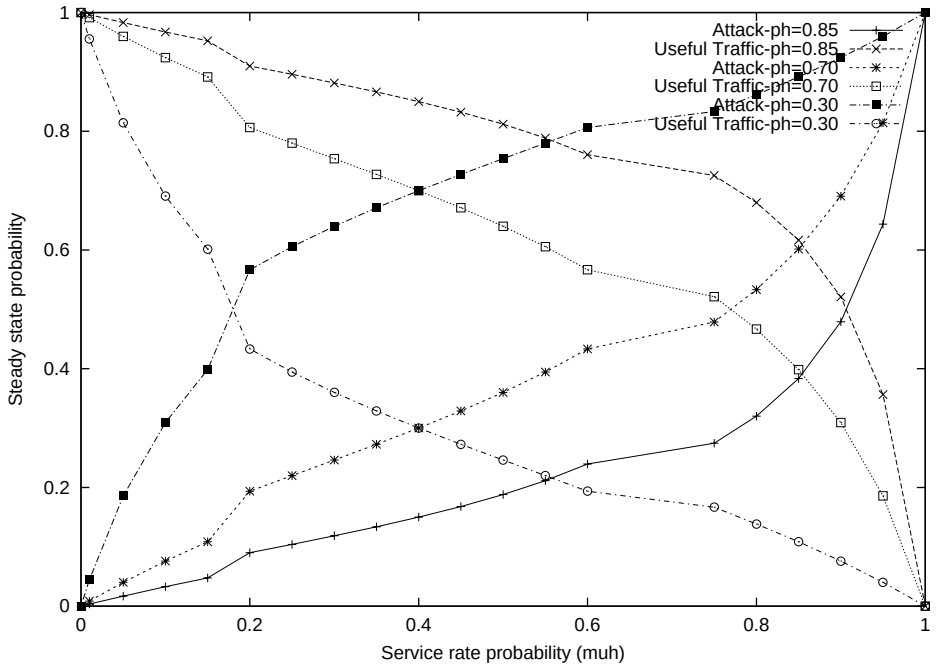

Figure 5.5: Impact of Changing Service rates for fixed Attack rates

survivability of the model *decreases*. In other words, if the server spends a comparable amount or larger amount of time discarding bad packets, in proportion to the amount of time it spends to process good packets, the overall survivability of the server degrades. However this degradation also depends on the attack rate. From the graph we observe that this degrade faster as the attack rate increases. For the three curves, the worst degradation occurs when `ph = 0.30` and the attack rate probability is 0.70. The crossover point for this curve occurs even when the proportional processing time for attack packets is four times as fast as legitimate packets (represented by `muh=0.2`. This suggests that packets need to be discarded much faster and techniques such as strong authentication which increase this gap are not suitable.

In general, we observe from the plots that discarding attack packets faster than useful packets always results in better survivability. However, for different attack rates, the crossover points occur at different proportions for different attack profiles. The more number of attack packets in proportion to useful packets, the earlier the crossover occurs, requiring that the packets be dropped faster under attack. This implies that characterizing attack packets quickly and discarding them is extremely important to ensure the survivability of the server.

In Section 5.3, we show how we can extend the server model presented to explicitly incorporate

clients and model their end-to-end delay characteristics under a DDoS attack. While we focused on how to make server's more survivable in this section, we extend our model to include the stochastic behavior of clients, and analyze a client's DDoS survivability properties.

5.3 Modeling Client DDoS Survivability

A DDoS or DoS attack is an attempt by unauthorized users in a network to deny access to legitimate users of a service. As a result of the attackers' behavior, authorized users' requests are subject to unbounded waiting times. The analysis of a server's DDoS survivability presented in Section 5.2 gives us a methodology to evaluate what strategies a server can adapt to decrease the impact of these attacks on its ability to do useful work. However, it does not provide us any insights into how attackers can influence the end-to-end delay observed by legitimate users of a network server.

In this section, we explore how to extend the PNS modeling and analysis framework to include clients and routers, and describe the delay characteristics of a client's request, using a traditional queuing theory model of network behavior. We formally specify client's survivability to a DDoS attack as a probabilistic delay guarantee expressed using a suitable temporal logic, and show how we can study the impact of different DDoS prevention strategies on a client's perceived waiting time.

In Section 5.3.1 we show with the help of a simple example how to model and analyze the delay characteristics of different classes of traffic. Using this example, we show how to specify client DDoS survivability properties in Section 5.3.2. We use the model to simulate analytically the behavioral characteristics of different DDoS prevention strategies including strong authentication (Section 5.3.3), and filtering (Section 5.3.4), with respect to the impact a client's waiting time. This analysis is in contrast to the analysis in the previous section that focused on the server's ability to do useful work.

5.3.1 Modeling End-to-End Delay for Clients

In this section, we show how we can estimate end-to-end per-packet delay for legitimate clients in our system when the server is undergoing a DDoS attack. Instead of focusing on individual packet flows, once again we perform a coarse-grained analysis by focusing on aggregate attack and

authorized-packet flows *during* an attack.

In a DDoS attack, only the networks closest to a DDoS victim is usually experiencing moderate to heavy load. Therefore we only model the aggregate traffic rates in these networks and use our analysis to infer end-to-end delay characteristics for legitimate clients. To keep the analysis tractable, we assume that the aggregate traffic rates on these networks are memoryless, especially if these flows comes from a large number of attack sources. When individual requests from legitimate servers intermingle with this traffic, using Kleinrock's Independence assumption, any correlations are eliminated.

Once again, we point out that we chose this model to illustrate our methodology and only claim that this model is a rough approximation to the actual attack and client request rates. Our contribution here is the methodology for analysis, which complements the underlying model of the network, and better models of aggregate traffic can only improve the confidence of our analysis.

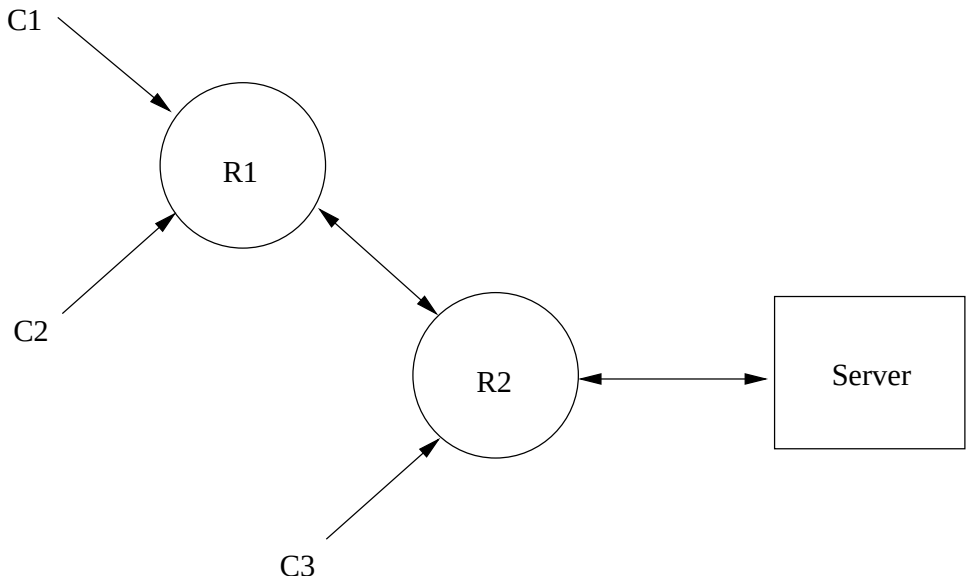

Figure 5.6: Example Client Network

We explain our modeling with the help of the example network shown in Figure 5.6. In this model, we have three clients, two routers, and one server. The clients in this picture represent aggregate traffic sources and may correspond traffic flows across ISP boundaries close to the victim server. The routers in these networks correspond to border routers on upstream ISPs from the client. Client streams 1 and 2 contain only legitimate requests, and Client stream 3 only attack

traffic. As more attack requests reach the server, as the server has finite computational resources, legitimate clients will start experiencing packet losses. If the legitimate clients' requests are sent using self-regulating protocol such as TCP, these losses will cause the clients to reduce their sending rates. The attackers however will not attempt to decrease their rates. We wish to study the impact of such behavior on the mean end-to-end network delay for the legitimate clients in steady-state.

We illustrate our modeling of DDoS survivability properties of authorized clients using the example from Section 5.2.2. Measurement data in the example network suggests that the average throughput under normal load is 38K packets/s. Since networks are engineered to provide optimal performance for normal load, and a small threshold to handle sudden bursts of traffic, we assume that the processing rate for authorized packets at the DDoS victim server is 40K packets/s. However under attack, a peak rate of $100K$ attack packets/s is observed at the server. This suggests that attack packets can be processed at rates approaching this value.

Let μ_h be the rate at which legitimate packets can be processed by the server, and μ_a the rate at which unauthorized packets are processed. We are now interested in the steady state behavior of this network for different attack profiles. Let the rate of attack be λ_a. Given the peak attack and authorized packet processing rates, under steady state, for a given value of λ_a, $\lambda/\mu < 1$, and no queues start to grow unboundedly we have:

$$\frac{\lambda_h}{mu_h} = 1 - \frac{\lambda_a}{\mu_a}$$

Using this equation, given a sustained attack rate λ_a, and fixed values for mu_h and mu_a, we can estimate the rate at which legitimate users can send packets as λ_h. The average end-to-end delay is given by $1/\lambda_h$ for authorized requests. This value is the reciprocal of the rate at which a legitimate user can send packets, for a given attack profile.

Note the formula above only models the expected rate at which legitimate clients can send packets, given an attack profile, and the network is stable, meaning that the queues do not grow unboundedly. This calculation does not include network transmission delays, or transient network congestion delays. Therefore our model of delay is conservative and the actual delays may be much higher. However, the trends we observe by modeling the steady state behavior of the network in this fashion are still useful from the point of view of resource engineering for stability.

In Section 5.3.2, I show how to express the DoS vulnerability of legitimate clients and show how

we can use the delay model to analyze the survivability of different DDoS prevention strategies, including strong authentication and packet filtering.

5.3.2 Specification and Analysis of Client DDoS Survivability Properties

We define a measure vulnerability of a client to DDoS attacks as follows:

Definition 5.3.1 (Client DDoS Vulnerability). *The vulnerability of an authorized DDoS client can be measured by the average end-to-end per-packet delay observed by the client for a given attack profile under steady-state. This corresponds to a probabilistic measure of average waiting time (PWT).*

As mentioned earlier, the reciprocal of the PWT value for a given attack profile represents the rate at which legitimate clients can send requests to the server and receive an appropriate response. Sending these requests any faster will likely increase the end-to-end delay for the request.

In order to understand the sensitivity of a network configuration to different attack profiles, it is useful to plot how the end-to-end delay per-packet changes as the attack rates increase in intensity. The rate of change of PWT values can give us important insights into the behavior of a network under attack.

Figure 5.7 plots the simulated attack rate vs. average end-to-end delay or PWT per-packet, under steady-state, for legitimate clients. For each attack profile, we calculate the steady-state packet rate that a legitimate client can send according to the formula presented in Section 5.3.1, varying the attack rates from 0 packets/s to 100K packets/s in steps of 1K. We only plot values of the delay observed by legitimate and unauthorized users attack rates from 6K to 88K packets/s since the delays increase unboundedly outside this range and the details at these levels are lost on the graph.

The curve labeled "No Attack" represents the average end-to-end delay given by the reciprocal of the normal workload for the example network (38K packets/s). The curve labeled "Authorized" represents the change in a legitimate user's PWT and the curve labeled "Unauthorized" represents how the attackers PWT changes. From the graph we observe that the PWT for legitimate clients degrades rapidly when the attack rate exceeds 60K packets/s. This suggests an appropriate threshold for detecting attack and deploying prevention strategies from the point of view of legitimate

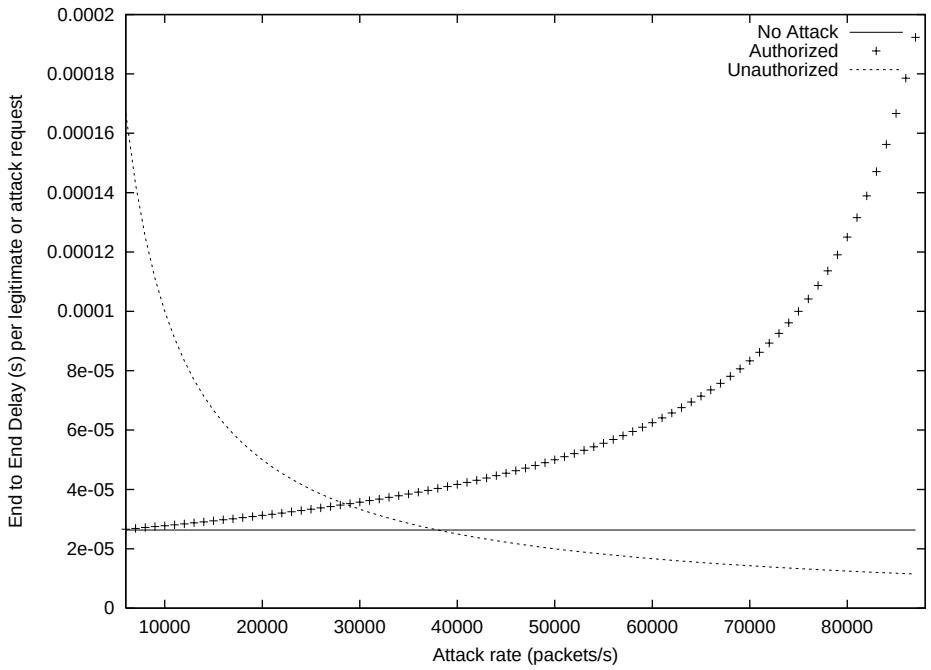

Figure 5.7: Baseline

client. Analysis results from Section 5.2.3 can be consulted to further refine this value and balance both server and client survivability needs.

As expected, the attackers' PWT decreases rapidly initially. At the crossover point of the two curves, around 28K attack packets/s, the PWT values for both traffic classes are equal. Beyond this point, it increases sharply for the legitimate clients, and falls less sharply for unauthorized packets. This crossover point can be used as a direct measure of the vulnerability of the network. If an attacker needs to send a greater amount of packets/s to cause this crossover, then the system is more resilient to DDoS attacks.

In the next subsection, we model the impact of different DDoS survivability strategies, including strong authentication and filtering on the PWT of legitimate clients.

5.3.3 DDoS Prevention Strategies

The first strategy we examine is strong authentication on the server. This is one strategy the server can implement even when the other network entities (clients and routers) cannot be reliably constrained (e.g.,, when they belong to different administrative domains). Each legitimate request can carry an authentication credential that authorizes the client making the request. The server

processes all requests and looks for the credential. If the credential can be found and validated, the appropriate response is sent. Otherwise, the request is dropped.

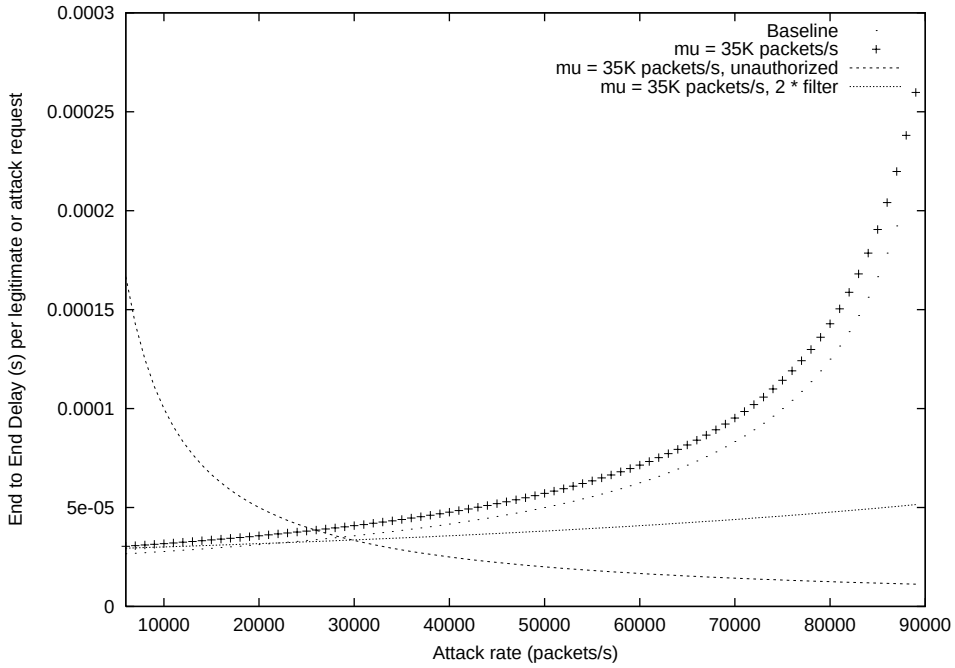

Figure 5.8: Strong Authentication

Adding a credential check to each request increases the time required to process each request. When the system reaches a steady state, this has the effect of decreasing the packet arrival rates. In Figure 5.8, we plot the impact of naïevely changing the server to include strong authentication, without changing the behavior of the server to attack packets. As shown by the curve marked with "mu = 35k packets/s", the PWT of legitimate clients increases with respect to the baseline. Furthermore the crossover occurs earlier at 25K packets/s, making it more vulnerable to DDoS attacks.

We also show how if we change the behavior of the server to filter unauthorized packets, the PWT for legitimate clients changes less slowly as shown by the curve labeled "mu = 35K packets/s, 2 * filter". This indicates that a combination of strong authentication and filtering can be useful in increasing the survivability of a DDoS client.

5.3.4 Filtering

We now plot how filtering attack packets faster at the server can impact the DDoS survivability from the viewpoint of a client in the system. Figure 5.9 shows how filtering attack packets faster at the server (without changing any other parameters) can improve the DDoS survivability of the clients in terms of their PWT policies. As the curves illustrate, dropping attack packets faster directly impacts the observed waiting times for authorized users in the system.

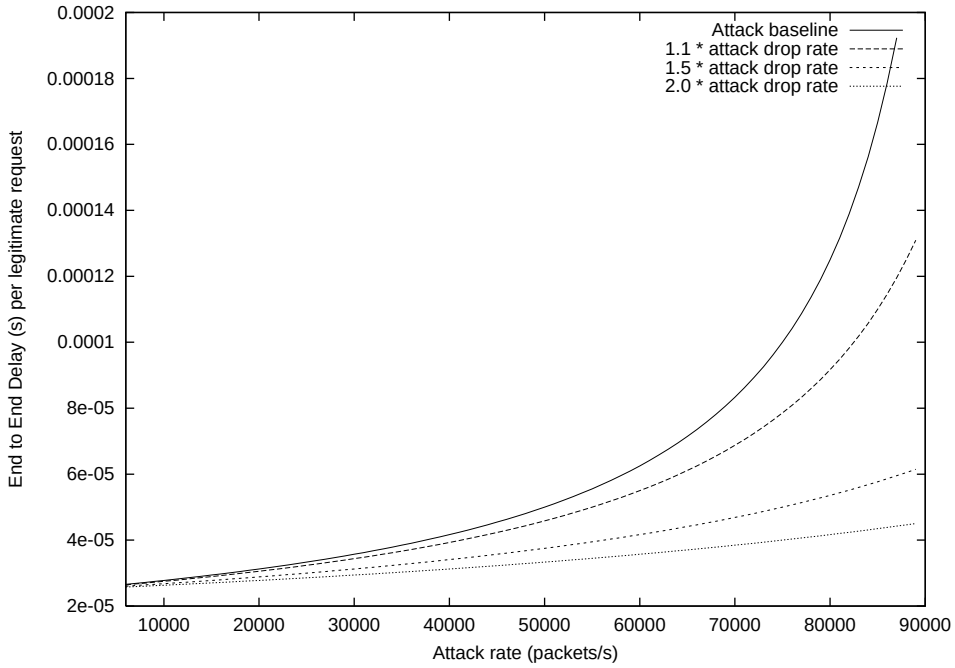

Figure 5.9: Filtering

5.4 Chapter Summary

In this Chapter, I present a formal model of network resource consumption and define network DDoS survivability as probabilistic temporal properties can be verified within this framework. Our modeling and analysis complements existing stochastic network models, and allows us to specify and measure both qualitative and quantitative aspects of DDoS survivability. To the best of our knowledge, this the first systematic treatment and analysis of survivability to DDoS attacks.

While researchers have developed many different techniques to tackle DoS and DDoS attacks, in the absence of a formal model it is difficult to reason about the relative impact of these different

techniques in terms of their survivability. Using our model, we can quantify the costs of deploying different DoS prevention strategies such as strong authentication, and filtering on network servers and observe their impact on qualitative and quantitative survivability properties. We evaluate the impact of different DoS prevention strategies using both standard automated model checking and stochastic analysis techniques.

In the next chapter I summarize related research, in the context of recovery-oriented security, availability policies, dynamic access controls and denial of service policies to differentiate how the work I present here in this thesis differs from the existing body of research.

Some have relied on what they knew;
Others on simply being true. What
worked for them might work for you.

Robert Frost, "Provide, Provide"

Chapter 6

Related Work

In this chapter, I present a summary of existing research and show how my thesis complements related topics. In Section 6.1 I show how concepts such as fault-tolerance, reliability and dependability relate to the model of survivability presented in this thesis. In Section 6.2, I summarize results in the context of analysis of access control models. In Section 6.3 I present a brief summary of different modeling techniques for state-transition behavior, especially for networks, including Petri-nets and Stochastic Activity networks (SAN) and show how our analysis extends to these models. Finally, in Section 6.4, I summarize existing research with respect formal models of DoS behavior, and strategies that tackle different aspects of DDoS attacks.

6.1 Modeling Dependability

The notion of specifying and analyzing the dependability of a system is not new. Research into the fault-tolerance and reliability of both hardware and software components in a distributed system is quite mature. For most part, this research has focused on developing robust techniques to handle a specific class of failures. In terms modeling robust protocols for dependability, the most common failure model for components is the independent and arbitrary (or Byzantine failure) model. Techniques to handle such failures include replication of components, load-balancing and sharing, checkpointing for recovery, probing, feedback, leader-election, and voting for consistency etc. The robustness of such models is measured purely quantitatively in terms of [100] the mean-time to failure (MTTF) and the mean-time to recover (MTTR) parameters.

Increasingly, researchers have started to include security as an attribute for dependability anal-

ysis. Avizienis et al. [10] have defined different attributes of dependability that include availability, reliability, safety, confidentiality, integrity and maintainability. The term survivability is used in this community to focus on two related problems : intrusion tolerance and database survivability.

Intrusion tolerance in distributed systems [44] is a well researched area. The area evolved from the study of the relationship between fault-tolerance and security techniques. More recently, researchers have focused on different abstractions and mechanisms for specifying intrusions, as an analogy to the fault classification and analysis. Researchers have worked on design of intrusion tolerant applications [132], communication protocols [50] and software infrastructure. The ITUA [37] 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. These techniques do not directly address the integrity and confidentiality concerns of specific users and information objects in an information protection system. The focus is mainly on availability of resources and maintaining consistency in the face of arbitrary failures.

Database survivability is also a well researched topic. In [84, 6] the authors illustrate the principles of trusted recovery in defensive information warfare [105] with respect to military databases. This work defines the notion of malicious and benign transactions and their dependence. They also present techniques for recovering databases from inconsistencies after an attack, by relying on redundancy and fault-tolerance. However the treatment of the material is almost exclusively geared towards maintaining transactional *integrity*. Their research does not address confidentiality properties or DoS protection directly.

More recently, Jha and Wing [73] 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. This work also extends the scope of traditional dependability models. They propose a general framework to specify different aspects of survivability as general safety and liveness properties, and suggest techniques for reliability and cost-benefit analysis.

The notion of survivability of security models is defined as the ability of a system to continue operation, especially in the presence of accidental failures or malicious attacks [51]. Our goal in this thesis is more modest. We refine the scope of this definition and explore how survivability can be specified and analyzed in the context of systems security. Instead of modeling for accidental or

arbitrary failures, or for general system models, we focus on specific attack descriptions within the framework of a more expressive access control model. Furthermore, our analysis is restricted to survivable notions of confidentiality, integrity, and availability properties from the point of view of resource access behavior. We believe that this is our unique contribution.

6.2 Access Control Analysis

To the best of our knowledge, we are not aware of any systematic studies that evaluate different access control policy implementations with regard to changing implementation mechanisms in response to a perceived threat or vulnerability. However, our work on analysis of access control models, with respect to RAs that can change access permissions and preserve consistency, is influenced by several recent research efforts.

The model for incorporating a proof of authorization as a guard that is evaluated before executing an RA, directly follows from the formalism presented by Schneider in the context of policies that can be enforced by execution monitoring [121] that can be enforced by execution monitoring. Schneider proves that if we can rely on the guards for consistency, then any system that correctly implements these mechanisms also correctly enforces the policy. This forms the basis for our proof of trust validation.

While we rely on a simple state-based specification language in this thesis, many researchers have focused on high-level languages and frameworks to describe access control policies and models. These languages typically abstract the logical properties of interest to capture delegation and interoperability between different access control specifications.

For example, Bertino et al. [17] describe a logical framework for reasoning about the expressive power of different access control models. Specifically they address the task of evaluating different flavors of extensions to database authorization models (e.g., negative authorizations, multi-policy models, role based authorizations etc). Koch et al. [78], analyze the interaction between different policy models using a theory of graph transformations. Jajodia et al. [72] propose a language for expressing authorizations that enables the enforcement of multiple access control policies and show how programs written in this language effectively capture the abstractions necessary to define different types of authorizations encountered in access control models.

In our work, we have focused on low-level mechanisms that implement access control policies. We recognize that our simplified semantics may not exploit the power of expression available by using these higher-level languages to define access control models, especially in database systems. A full treatment of the survivability of these models by describing their semantics is beyond the scope of this thesis.

In order to evaluate the flow of trust in an access control , many different formal analysis techniques are available. Weeks [129] provides a formal semantics for expressing trust management systems via a fixpoint lattice model for monotonic assertions. This model is useful to understand the trust management of capability-based assertions. Chander et al. [28] provide a state transition approach to model the interaction of trust management and access control. The interaction of access control and trust management presented in our thesis, including the use of unforgeable credentials to provide authorization proofs, and the equivalence of ALs and CLs can be validated in their framework.

The capability-based KeyNote system of Blaze et al. [23], provides a single language for both policies and credentials, based on predicates that describe the trusted actions permitted by holders of specific public keys (or other cryptographic identifiers). Our model integrates access control with a simple trust management mechanism. The main purpose of KeyNote is to express and evaluate policies and trust delegations that occur in PKI applications. KeyNote can be integrated into our framework for trust management for other types of dynamic policies that require more require credentials.

We believe that the model of access control behavior we used in this thesis abstracts the relevant entities and their interactive behavior adequately to describe survivability properties of interest. The analysis techniques summarized here enchance a designer's understanding of the flow of trust and the power of expression of more complicated or composite models of access control behavior. These techniques can be used to complement our survivability modeling for integrity, confidentiality and availability properties.

6.3 Models of Access Control Behavior

In Chapter 5 of this thesis, we used Markov chains to describe network access behavior. Markov chains are finite-state models of probabilistic phenomena. Using DTMCs and CTMCs to model queuing systems and estimate steady state probabilities is fairly standard [77]. In general, these models are useful to specify and analyze quantitative aspects of probabilistic system behavior.

Formal verification techniques such as model checking on the other hand try to answer questions related to the functional correctness of reactive system behavior, in a qualitative manner. Fortunately, the model of a PNS presented in this thesis allows us to integrate both quantitative and qualitative analysis techniques in one framework. The state-transition based techniques used to represent quantitative network behavior fits in well with qualitative state-transition formalisms of finite-state systems that exhibit probabilistic phenomena. This integrated approach to system specifications provides a best of both worlds framework for analysis of survivability properties.

In addition to augmented Markov chains as presented here, various other techniques are available to specify and analyze qualitative and quantitative aspects of model or network behavior. These include stochastic Petri nets, and process algebras. Eventually, for analysis purposes, these models are converted into finite state non-deterministic automata and the underlying theory behind these techniques is the same. However, some of these models provide richer abstract semantics, and are easier to use.

Petri Nets [108] are a formal and graphical language appropriate for modeling systems with concurrency. The language of Petri nets is a generalization of automata theory and allows one to express concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastically occurring events (as Stochastic Petri Nets). A Petri net is described in terms of places, transitions, and arcs. Some places are marked as input and others as output. Places can contain tokens; the current state of the modeled system called a marking, is given by the number of tokens in each place, and their type if the tokens are distinguishable Transitions are active components. The tokens model activities which can occur, causing a transition to fire, thus changing the state of the system. Transitions are enabled, or allowed to fire, when all the preconditions for the activity are fulfilled, i.e., there are enough tokens available in an input place. Transitions fire at different times, or concurrently, based on the system being modeled, and the tokens are moved to output places.

The number of tokens removed or added depends on the cardinality of each arc. The interactive firing of transitions in subsequent markings is called token game. Timing characteristics can be included by specifying firing delays.

A Process Algebras [16] is formal description technique used to specify reactive systems involving communicating and concurrently executing components. The fundamental abstraction is a process graph, which corresponds to a labeled transition system (LTS). A node corresponds to an agent in the system, and the labels to atomic actions. Process graphs are represented algebraically in the form of terms. Process algebra focuses on the specification and manipulation of process terms as induced by a collection of operator symbols. Process algebra imposes an equational logic on process terms, such that two terms can be equated only if they are behaviorally equivalent. Process Algebras are attractive because they can describe composition naturally, and for their ability to compare behaviors. Probabilistic process algebras are the closest equivalent to our Markov chain formalism and adhere to Markov chain semantics.

Stochastic Petri nets (SPNs) and Stochastic Process Algebras (SPAs) can be viewed as high level description languages for Markov processes. Many comparisions of SPNs and SPAs are available, concentrating on their different representations of causality, concurrency, compositionality, and the ability to recognize equivalent behaviors. SPNs have a very clear notion of states, whereas SPAs focus on actions. At the specification level, SPNs are graphical, whereas SPAs are a textual language. Any Markov chain can be expressed, although as a degenerate form, using either formalism. The choice of an appropriate specification language can impact the usability of our analysis framework, but does not detract from the general techniques presented.

6.4 DOS Related Work

In this section, we present related research, both in the context of formal modeling and analysis of DoS, as well as different DoS and DDoS prevention strategies. We also highlight how other service models such as QoS networks and ATM networks address the DoS problem.

6.4.1 Formal Modeling

Shields [122] presents informal, but comprehensive definitions of network DoS. However, the DoS problem itself has received some attention in the past. One of the first attempts to formalize the notion of DoS resistance was by Gligor [58] who defined a quantitative metric called MWT (Maximum Waiting Time). Amoroso [8] highlights the need for specifying a service model in terms of policies as predicates involving subjects (users) and objects (resources) and resource consumption operations. DoS policies are specified in terms of predicates that specify conditions, using priorities, under which a subject can deny other authorized subjects access to a critical objects.

Subsequently, Yu and Gligor [134] extended the MWT notion to include qualitative aspects of resource access and sharing behavior and devised a specification and verification proof methodology (manual) to guarantee finite waiting times (FWT) and prevent DoS. As mentioned earlier,

Millen [97] analyzes trust assumptions, specifies a state-transition model of resource consumption behavior, and suggests the use of probabilistic waiting time policies. In Millen's resource allocation model [97] provides a framework for expressing detailed time and space constraints to specify denial of service rules and policies. Service denials occur when the space and time allocations for some process does not meet its requirements. FWT and MWT policies can be easily specified in this context. PWT is suggested as future work in the model.

In [95] Meadows defines a formal framework to analyze network authentication protocols. With the help of a cost-benefit analysis framework, this work shows how cryptographic protocols can be made more resistant to DoS by trading the costs incurred by a defender against the costs to the attacker. By making the attackers (or unauthorized clients of a service) do more work than the defenders, the benefit of mounting a DoS attack can be made less attractive.

We are not aware of any formal characterizations of network DoS behavior that explicitly include qualitative and quantitative definitions of DoS and DDoS survivability as presented in this thesis.

6.4.2 DDoS Attacks

Many innovative solutions have been suggested to tackle different aspects of DDoS attacks. We observe that most anti-DDoS solutions rely on filtering attack packets as a basic mechanism to prevent DoS. The key to applying filtering is the ability to distinguish between attack and non-

attack packets. DDoS tools spoof source addresses to prevent administrators from tracing back the flow to the zombies. However, these spurious addresses can be detected by vigilant border routers. Filtering these datagrams by border routers (called ingress or egress filtering [54, 70]) can prevent these packets from entering intermediate networks and contending with legitimate packets there.

In addition, most attack packets are ICMP or UDP floods and most ISPs filter out non-HTTP traffic to HTTP servers. However, many DDoS attack scripts have started randomizing the fields in a packet header to defy classification [86]. Therefore, identifying such attack packets may be difficult and may require a large amount of per-flow state on routers deep in the network.

We examine some of the proposed anti-DDoS solutions to characterize attack traffic. The D-Ward Project [38] explores the use traffic monitoring to detect "abnormal" flows on routers in source networks (or client networks) to detect DDoS attacks and use appropriate filtering to tackle these attacks at the source. In CenterTrack [125], network monitoring is used to detect abnormal levels of attack traffic using special diagnostics-only routers that are interspersed with regular routers, but also form a separate physical network. Malicious flows are directed to these routers, and they can be isolated from the main routing paths in the network. In SOS [76], a secure overlay network is formed over existing networks to route useful traffic to servers. Instead of burdening edge routers with anti-DoS filtering, SOS pushes the filtering of traffic deep into the network where routers can easily process and filter large amounts of traffic.

Many schemes to traceback and filter IP datagram attack "streams" to their source interfaces, in spite of spoofed addresses, have been proposed in the recent years. We present a brief summary here to highlight the importance of such schemes to help identify the source of attack floods.

In ICMP Traceback [15], a router generates an ICMP traceback message with a certain probability and sends this packet along with the rest of the flow to the destination. Over time, the destination receives enough traceback messages to determine all the traffic sources, including the zombies. IP Traceback [120] is an alternative to ICMP traceback. Here routers probabilistically mark packets (in the IP header field) with partial path information during packet forwarding. The victim can reconstruct the complete paths after receiving a modest number of packets. The performance overheads of IP Traceback are improved in Advanced and Authenticated Marking [124], which basically authenticates the packets markings in IP Traceback. The analytical properties of

probabilistic traceback have also been studied [106]. Hash based IP traceback [123] uses an innovative technique to encode the IP address of intermediate routers concisely in the packet itself. In [133], the authors propose a new packet marking approach called Pi, for path identifiers, which are embedded as fingerprints in IP datagrams and can be reconstructed by victims to identify attackers in spite of spoofing.

IP Pushback [88, 57, 71] is a proposed IP router mechanism that enables routers to characterize floods of suspicious attack traffic and deploy aggregate congestion control (ACC) mechanisms to stem their flow to the victim. IP Pushback enables the mechanisms to implement filtering on chosen intermediate routers and can be used with Traceback. A variety of different heuristics to characterize attack packets have been proposed. However, since the defining characterizations of DDoS attack packets are becoming harder to detect, coming up with reasonable heuristics to use Pushback effectively is a significant challenge.

We observe that many of the solutions presented here advocate filtering as a mechanism to prevent DDoS attacks. Some of the solutions enable filtering at the source, some on intermediate routers and still others on servers. We believe our modeling and analysis will help quantify the costs and benefits of implementing different filtering mechanisms and help administrators decide which solutions are most effective in their particular context.

In the next subsection, we discuss how a QoS network addresses the DoS or DDoS problem.

6.4.3 QoS

The Internet offers very simple quality of service or QoS model, i.e., point-to-point, best-effort data delivery. In order to get better support for real time applications and to control the ability to share bandwidth on a particular link among different traffic classes (controlled link sharing) various modifications to the basic service model have been proposed. Two design philosophies have emerged in recent years : IntServ and DiffServ. The IntServ model includes best-effort service, real-time service and controlled link sharing. It uses resource reservation[24] and admission control as its basic building blocks.

Diffserv uses packet marking and advocates the use of special field in the IP packet[22]. Packets are classified and marked to receive a particular per-hop forwarding behavior on nodes along their

path. Sophisticated classification, marking, policing, and shaping operations need only be implemented at network boundaries or hosts. The mechanism of interest to us is the "In-Out" packet marking scheme [31]. This scheme maintains a threshold value per user called the traffic meter and marks packets in excess of a traffic meter, as "out", but they are not dropped by the sender. These packets may be preferentially dropped by downstream receivers. The traffic meter specifies the end-to-end QoS parameters required by the application, but does not take bandwidth thresholds on links into account.

A technology that provides end-to-end QoS using hop-to-hop admission control is ATM. ATM functionality corresponds to the MAC layer and physical layer in the IP protocol stack. ATM[1] is a switch-based connection-oriented protocol with fixed packet sizes. The fixed packet size allows better prediction of QoS parameters. It defines a protocol for end-to-end service parameters negotiation.

ATM provides five service categories: constant bit rate (CBR), real-time variable bit rate (rt-VBR), non real-time variable bit rate nrt-VBR), available bit rate (ABR), and unspecified bit rate (UBR). Traffic is classified into one of these five categories, based on an application's end-to-end QoS requirements, and packets from different applications in different classes will receive different QoS. The protocol reserves chunks of bandwidth to different QoS classes, and implements an admission control mechanism per class. In RSVP or some existing DiffServ schemes, unused bandwidth within a reserved class is wasted if it is not used by the application. ATM defines a class called ABR that uses an adaptive admission control strategy to probe and utilize unused bandwidth belonging to other classes.

QoS networks can reduce the vulnerability of server to denial by consumption attacks by virtue of their traffic shaping and traffic policing mechanisms. However, end-to-end resource reservation does not prevent a malicious sender from sending excess packets that violate the service level agreement. For this reason both admission control and per-flow statistics are necessary to detect and punish misbehaving users in QoS networks. If the amount of state that needs to be maintained in order to accomplish this is very large, scalability could be a issue.

To summarize our discussion on related work with respect to solutions to the DDoS problem, we observe that a variety of techniques to address different aspects of the problem are available.

Our framework presents a general framework evaluating the efficacy of these techniques.

We present our conclusions, catalog lessons learned, and suggest directions for future work in Chapter 7.

When one admits that nothing is certain one must, I think, also admit that some things are much more nearly certain than others.

Bertrand Russell

Chapter 7

Conclusions

This chapter highlights the lessons learned from my study of modeling and analysis of survivability properties for recovery-oriented security in access-control models, and summarizes solutions I explored to address relevant problems. In Section 7.1 I revisit the questions I posed in Section 3.5 and justify how I address these concerns. This is followed by a summary of contributions in Section 7.2. Finally, I discuss how the concepts and models I propose in this thesis can be extended in the future in Section 7.3.

7.1 Conclusions

In this section, I present an evaluation of my thesis in terms of the success criteria I defined in Chapter 3. Each numbered paragraph in this section is a justification of the corresponding evaluation question from Section 3.5.

1. In my thesis, I claim that existing models of access control security make strong assumptions about the nature of security guarantees that can be enforced by access control mechanisms in real systems. In particular, modeling access control security as safety properties provides a very restricted notion of information assurance. Existing state-of-the-art models cannot capture the behavior of an access control system under threat of exposure or attack, and fail to provide useful abstractions to model notions of recovery, from compromise or policy enforcement failure. I believe that the modeling, specification, and verification methodology I present in this thesis addresses these issues directly, and highlights the importance of including these notions to build survivable security solutions. The importance of survivable security

can only be understated.

2. I develop my theory of recovery-oriented security using a general temporal logic framework. I extend the state-transition formalism of access control models to explicitly incorporate both qualitative and quantitative measures of system behavior. Qualitative temporal logic formalisms are widely used to specify concurrent system behavior, because they can describe the ordering of events in time, without the need to model time explicitly. The meaning of a temporal logic formula is defined in the context of labeled state-transition graphs of execution semantics called Kripke structures. Both synchronous and asynchronous models of execution can be specified using these abstractions. My choice of branching time temporal logic for specification of survivability properties is well suited to capture the nature of user actions in the context of non-deterministic and stochastic behavioral models of access control systems. Furthermore, I show how we can use standard augmentations of temporal logic frameworks to model probabilistic and real-time guarantees in these models. Therefore I claim that choosing this framework to model recovery-oriented security is well-suited to describe temporal properties of such systems.

3. One of the major contributions of my thesis is the ability to capture the temporal nature of security (or insecurity) guarantees in models, using availability as a measure of survivability as well as recovery. The ability of a system to resist compromise can be measured in terms of how long it can provide authorized access to uncompromised users and resources. The ability to recover quickly, whenever possible, can also be captured by measuring how long the system was unavailable, or by how fast it can become available again. Making a model resilient to denial of service can be formulated easily as making it more available. I show how we can specify survivability, recovery, DoS resilience and DoS resistance as availability properties. Therefore, we can use the same concepts to describe different guarantees in the model and use the same verification methodology uniformly. Our model provides a cost-benefit analysis framework in terms of availability, and can be used to study the usefulness of different design choices for recovery.

4. In my study of recovery strategies, I show how we can extend the lifetime of critical resources

in the system by changing access control policies and mechanisms on the fly, in a controlled manner, without sacrificing consistency guarantees. I describe how to augment a policy specification automatically, to preserve trust-relationships of interest. I also demonstrate how implementation mechanisms can affect the overheads of changing access controls. The formal study of dynamic access controls for recovery is also an original contribution of this thesis.

5. I show how we can directly employ standard model-checking techniques to verify survivability policies. Model checking is widely used for verifying finite state concurrent systems. For branching time formulas expressed in CTL, $PCTL$ and CSL, model-checking is decidable and efficient. The models presented in this thesis are small, yet rich enough to prove different properties about the behavior of systems under information exposure or DoS attacks.

   While the need to abstract system behavior using finite-state semantics may seem restrictive, symbolic model checking algorithms can increase the efficiency of this process. Techniques such as abstraction, exploiting symmetry and reduction can further expand the size of models that can be verified within a given set of memory and processing constraints. Systems that are not finite state can be verified using model checking in combination with abstraction, induction, or even deductive reasoning and theorem proving.

6. I claim that this thesis work is the only formalism of DDoS attacks, to the best of our knowledge, which models the unique characteristics of network resource-consumption behavior. By integrating stochastic models of arrival rates, service rates and traffic congestion behavior into our state-transition model of access request behavior, security engineers can analyze the impact of different network operating characteristics and their effect on a model's vulnerability to DoS and DDoS attacks. As shown by our examples, the modeling and analysis framework can be used to study and prove survivability properties of different proposed DoS-prevention strategies.

7.2 Summary of Contributions

In this section, I itemize the major contributions of my thesis as follows:

1. I describe a new methodology for information assurance that challenges the "all-or-nothing" nature of guarantees provided by existing access control models. In particular I claim that access control systems are regularly compromised, either maliciously or inadvertently and safety guarantees are invalidated periodically. I argue that we need a new abstraction I call *recovery-oriented* security to explore the nature and scope of guarantees in realistic system models that are under threat of attack. I define survivability in this context as the ability of a system to provide flexible response and recover from policy compromise.

2. I show how to extend existing access control behavior models to explicitly capture the effect of information compromise as well as DoS behavior. I incorporate nondeterministic, probabilistic, as well as real-time behavioral attributes to extend the power of expression of traditional models. I claim these extension are the key to define as well as analyze the temporal nature of security guarantees in real systems. I show how we can specify appropriate notions of survivability and recoverability, as well as resilience to DoS and DDoS attacks in this extended behavioral model, using appropriate flavors of temporal logic. One of the benefits of this methodology is the ability to use standard automated model-checking techniques for verification of temporal security guarantees. I show how we can define models of different

3. I also explore how changing access control policies in response to vulnerabilities and threats of exposure can preserve temporal security guarantees and increase survivability of a system. I study the overheads of implementing flexible response using dynamic access control, and explore how we can preserve safety and trust assumptions in this context.

4. Finally, I show how my framework can be used to formalize of the network DoS and DDoS problem, and specify and verify both qualitative and quantitative DoS survivability properties. This formulation allows us to study different DoS-prevention strategies and compare their relative effectiveness in reducing DoS vulnerabilities. I also show how we can integrate stochastic modeling and analysis with behavioral models of network access control to study the DoS problem in the context of specific network models and validate the usefulness of different response strategies.

To summarize, I claim my thesis provides a rich semantic framework to specify temporal notions of recovery-oriented security and explore the nature and scope of flexible response strategies to improve the survivability of access control models.

7.3 Future Research

Using the specification and verification methodology from this thesis, I would like to focus on automating the process of generating strategies to augment system models and improve the survivability of an information protection system under attack.

The ability to respond to security threats and recover from attacks on operating systems and networks is viewed as a particularly challenging problem in information assurance. While the administrators of a system or a network's security-configuration have a variety of options with respect to automated tools for intrusion detection and configuration management, it is becoming increasingly difficult to shrink the window of opportunity for an attacker using existing tools and mechanisms, between when a vulnerability is detected to when it is exploited. In this context, automatic response is seen as an interesting if not controversial solution, with significant technical and legal challenges before it can be adopted widely.

This problem can be investigated at different levels, with the final goal of developing a framework to implement automated response actions. However, solution need to have a strong theoretical basis, in terms of providing a methodology to prove and analyze the impact of these response actions, with regard to their ability to recover or restore security guarantees.

To accomplish this, we propose to leverage my thesis work, and study specific instances of automatable responses in the context of a particular installation, and design an administrative interface that can pro-actively apply these actions in response to alerts from system logs and intrusion detection systems.

Specifically I would like to incorporate risk analysis to study the cause and consequences of threats and exposures on an abstract model of an information protection system. Identifying potential sources of exposure can greatly influence a security engineer's ability to design policies for robustness and understand the tradeoffs between safety and availability. However, subjecting a real system to attacks is dangerous and impractical. I propose to build a vulnerability analysis tool

that can take different abstract system models and associate risk values, as well as different attack models, and analyze and quantify the costs and benefits of choosing different response actions.

To automate this process, I would like to explore techniques to build abstract models automatically from a system installation. The tool will also include fault-trees and event trees, as well as options to specify different risk models. Fault trees are used to capture the consequence of information compromise and propagation of confidentiality and integrity failures. Fault trees are generated by intrusion detection and anomaly detection systems and can also be compiled from vulnerability reporting sources such as Bugtraq and CERT. A fault tree database can be automatically constructed for a given system model by collating the fault trees for different entities. Event-trees capture the dynamic transitions of our abstract model and can be gathered from system logs or specified by the user using an appropriate specification tool.

This proposed tool will also include different risk models as well as risk analysis techniques to perform a cost-based cause-consequence analysis. Using availability as one metric, for example, we can explore the consequence of different attacks on our model and identify critical paths and entities that are high-risk and therefore may need stronger protection. This process will also give us a mechanisms to study insider attacks, which have so far been notoriously difficult to model, and use the specification and analysis techniques described above to capture insider threats and observe their consequence. The modeling and analysis should ideally output a list of strategies to reduce the vulnerability of attack, and a cost-benefit analysis of these strategies.

Appendix A

Semantics of CTL^*

Two flavors of temporal logic are popular. They differ in terms of whether they represent flow of time implicitly in in terms of states and transitions, or whether they also include real-time in their formalisms explicitly. In models where real-time values are not included explicitly, the two standard temporal operators are *henceforth* and *eventually*. The henceforth operator is written symbolically as $\Box$ or as **G** (for "globally"), and asserts that the property is true in all states of the model in the future. The eventually operator is written usually as $\Diamond$ or as **F**(for "finally"), and denotes that some state that can satisfy the property will be reached eventually. In addition to these two temporal operators, the regular boolean connectives can also be used to construct temporal logic formulas.

Temporal logics are interpreted in the context of Kripke structures, which are essentially state-transition graphs.

We describe availability using the operators in CTL^*, which is a powerful logic for describing properties of computation trees. Note that our abstraction of request-response trace is essentially a computation tree. CTL^* formulas are composed of *path quantifiers* and *temporal operators*. Path quantifiers are used to describe the branching structure of computation trees. There are two path quantifiers in CTL^*: **A**, which stands for "in all computation paths" and **E** for "in some computation path". In addition, we also have **G** and **F** as described previously, as well as **X**, which requires the property hold in the second state of the path, and binary operator **U** for two properties that holds if there is a state on the path where the second property holds, and at every preceding state on the path, the first holds. Operators $\vee$, $\neg$, **X**, **U**, and **E** are sufficient to represent any CTL^* formula. There are two types of formulas in CTL^*: *state formulas* and *path formulas*. State

formulas are true in specific states and path formulas along specific paths. They are defined as follows:

- If $p \in AP$, then p is a state formula.
- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a path formula, $\mathbf{E}f$ and $\mathbf{A}f$ are state formulas.
- If f is a state formula, then it is also a path formula.
- If f and g are path formulas, $\neg f$, $f \vee g$, $\mathbf{G}f$, $f\mathbf{U}g$, $\mathbf{X}f$, and $\mathbf{F}f$ are path formulas.

For the Kripke structure M, a path $\pi \in M$ is an infinite sequence of states $\pi = s_0, s_1, \cdots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$. π^i is the suffix of π starting at s_i. If f is a state formula, the notation $M, s \models f$ means f holds at state s in M. If f is a path formula $M, \pi \models f$ means f holds along π in M. The $\models$ relation is defined inductively as follows:

1. $M, s \models p \leftrightarrow p \in L(s)$
2. $M, s \models \neg f \leftrightarrow M, s \not\models f$
3. $M, s \models f \vee g \leftrightarrow M, s \models f$ or $M, s \models g$
4. $M, s \models \mathbf{E}f \leftrightarrow$ there is a path π from s such that $M, \pi \models f$
5. $M, s \models \mathbf{A}f \leftrightarrow$ for every path π from s, $M, \pi \models f$
6. $M, \pi \models f \leftrightarrow s$ is the first state of π and $M, s \models f$
7. $M, \pi \models \neg f \leftrightarrow M, \pi \not\models f$
8. $M, \pi \models f \vee g \leftrightarrow M, \pi \models f$ or $M, \pi \models g$
9. $M, \pi \models \mathbf{X}f \leftrightarrow M, \pi^1 \models f$
10. $M, \pi \models \mathbf{F}f \leftrightarrow$ there exists $k \geq 0$ such that $M, \pi^k \models f$
11. $M, \pi \models \mathbf{G}f \leftrightarrow$ for all $k \geq 0$, $M, \pi^k \models f$

12. $M, \pi \models f\mathbf{U}g \leftrightarrow$ there exists $k \geq 0$ such that $M, \pi^k \models g$ and for all $0 \leq j < k$, $M, \pi^j \models f$

Note **X**, **F**,**G**, and **U**, are defined only for paths.

There are two useful sub-logics of CTL^*, one for branching time and one for linear time called CTL (Computation Tree Logic) and LTL (Linear Temporal logic respectively. In CTL, each of the temporal operators **X**, **F**, **G**, and **U** *must* be immediately preceded by a path quantifier (either **A** or **G**). Therefore in CTL temporal operators quantify over paths that are possible from a given state. LTL is useful for describing events along a single computation path. LTL formulas are of the form $\mathbf{A}f$ where f is a path formula whose only state sub-formulas can be atomic propositions.

Appendix B

PRISM Code for PNS of DDoS Victim Server

```
// DDoS Server Model as a CTMC
// Model as one process with two state variables http and attack
// and four probability values muh mua, ph, pa

stochastic

const double muh;
const double nmuh = (1.0-muh);
const double mua = nmuh;
const double nmua = muh;
const double ph;
const double nph = (1.0-ph);
const double pa = nph;
const double npa = ph;

module DDOS
http : [0..2] init 1;
attack : [0..2] init 0;
```

```
[] (http=1) & (attack=0) -> nmuh:(http'=1)&(attack'=0) + muh:(http'=2)&(attack'=0);
[] (http=2) & (attack=0) -> ph:(http'=1)&(attack'=0) + nph:(http'=0)&(attack'=1);
[] (http=0) & (attack=1) -> nmua:(http'=0)&(attack'=1) + mua:(http'=0)&(attack'=2);
[] (http=0) & (attack=2) -> npa:(http'=1)&(attack'=0) + pa:(http'=0)&(attack'=1);
endmodule
```

References

[1] Atm forum online. http://www.atmforum.com/.

[2] M. W. Alford, J. P. Ansart, G. Hommel, L. Lamport, B. Liskov, G. P. Mullery, and F. B. Schneider. *Distributed Systems: Methods and Tools for Specification.* LNCS 190, Springer-Verlag, 1985.

[3] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. In *Distributed Computing*, 1986.

[4] R. Alur, C. Courcoubetis, and D. L. Dill. Model checking in dense real-time. In *Information and Computation, 104(1):2-34*, 1993.

[5] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *Information and Computation 104:35-77,*, 1993.

[6] 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.

[7] P. Ammann and R. Sandhu. Extending the creation operation in the schematic protection model. In *In Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press*, 1990.

[8] Edward Amoroso. A policy model for denial of service. In *Proceedings of IEEE Computer Security Foundations Workshop III, pages 110–114, Franconia, NH USA*, June 1990.

[9] Ross Anderson. A security policy model for clinical information systems. In *Proceedings of 1996 IEEE Symposium on Research in Security and Privacy*, 1996.

[10] A. Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability. In *LAAS Report*, April 2001.

[11] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time markov chains. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV96), Vol 1102, LNCS, pp 269-276, Springer*, 1996.

[12] Boehm B. Software engineering. In *IEEE Transactions on Computers, C25(12):126-41*, Dec 1976.

[13] C. Baier and M. Z. Kwiatowska. Model checking for a probabilistic branching time logic with fairness. In *DISTCMP: Distributed Computing, 11*, 1998.

[14] D. Bell and L. La Padula. Secure computer systems: Unified exposition and multics interpretation. In *Technical Report MTR-2997, MITRE, Bedford, MA*, 1975.

[15] S. Bellovin. ICMP Traceback Messages. Internet draft, March 2000.

[16] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka. *Handbook of Process Algebra*. Elsevier, ISBN: 0-444-82830-3, 2001.

[17] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies*, May 2001.

[18] D. Bertsekas and R. Gallagher. *Data Networks, 2nd Edition*. Prentice Hall, 1992.

[19] Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In *In the Proceedings of the Foundations OF Software Technology and Theoretical Computer Science, Bangalore, India*, 1995.

[20] K. Biba. Integrity considerations for secure computer systems. In *Technical Report MTR-3153, MITRE Corporation, Bedford, MA*, Apr 1977.

[21] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley,ISBN 0-201-44099-7, 2003.

[22] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475.

[23] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Security Protocols International Workshop, Cambridge, England*, 1998.

[24] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205.

[25] D. Brewer and M. Nash. The chinese wall security model. In *Proceedings of 1989 IEEE Symposium on Research in Security and Privacy*, 1989.

[26] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi. Seraphim: dynamic interoperable security architecture for active networks. In *OPENARCH 2000*, Tel-Aviv, Israel, March 26–27, 2000.

[27] Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun. Internet traffic tends toward poisson and independent as the load increases. In *Nonlinear Estimation and Classification , eds. C. Holmes, D. Denison, M. Hansen, B. Yu, and B. Mallick, Springer, New York*, 2002.

[28] A. Chander, D. Dean, and J. Mitchell. A state-transition model of trust management and access control. In *14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, June 2001.

[29] Chroot. Securing and optimizing linux, redhat edition—a hands on guide. http://www.faqs.org/docs/securing/chap29sec254.html, 2003.

[30] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of 1987 IEEE Symposium on Research in Security and Privacy*, 1987.

[31] D. Clark and J. Wroclawski. An Approach to Service Allocation in the Internet. Internet-draft, July 1997.

[32] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. In *ACM Transactions on Programming Languages and Systems*, 1986.

[33] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[34] R. W. Conway, W. L. Maxwell, and H. L. Morgan. On the implementation of security measures in information systems. In *Communications of the ACM, Volume 15(4)*, Apr 1972.

[35] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *In Proceedings of the 23rd National Information Systems Security Conference (NISSC)*, October 2000.

[36] CSI-FBI. 2003 computer crime and security survey. http://www.gocsi.com/awareness/fbi.jhtml, May 2003.

[37] 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.

[38] D-Ward. Ddos network attack recognition and defense. http://www.lasr.cs.ucla.edu/ddos/.

[39] Luca de Alfaro. Temporal logics for the specification of performance and reliability. In *14th Symposium on Theoretical Aspects of Computer Science Hansestadt Lu"beck, Germany (STACS 97)*, Feb 1997.

[40] D. E. Denning, P. J. Denning, S. J. Garland, M. A. Harrison, and W. L. Ruzzo. Proving protection systems safe. In *Computer Science Department, Purdue University, Indiana*, Feb 1978.

[41] Dorothy Denning. A lattice model of secure information flow. In *Communications of the ACM, Volume 19(5):236-243*, May 1976.

[42] Dorothy Denning. *Cryptography and Data Security*. Addison Wesley, 1982.

[43] Peter J. Denning. Third generation computer systems. In *Computing Surveys, Volume 3(4)*, Dec 1971.

[44] 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.

[45] E. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. Communications of the ACM, 18(8):453–457, 1975.

[46] D. Dittrich. Stacheldraht Analysis. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.

[47] D. Dittrich. Tribal Flood Network analysis. http://staff.washington.edu/dittrich/misc/tfn.analysis.

[48] D. Dittrich. Trin00 analysis. http://staff.washington.edu/dittrich/misc/trinoo.analysis.

[49] D. Dittrich. DDoS: Is There Really a Threat. Invited Talk, USENIX Security Symposium, August 2000.

[50] 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.

[51] 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.

[52] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. In *Science of Computer Programming, 2:241-266*, 1982.

[53] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Journal of Computer and Systems Sciences, 30(1):1-24*, 1985.

[54] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source address Spoofing. RFC 2267.

[55] D. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. A proposed standard for role based access control. In *ACM Transactions on Information and System Security , vol. 4, no. 3*, Aug 2001.

[56] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NSA National Computer Security Conference*, Baltimore, MD, oct 1992.

[57] Sally Floyd, Steven M. Bellovin, John Ioannidis, Kireeti Kompella, Ratul Manajan, and Vern Paxson. Pushback messages for controlling aggregates in the network. draft-floyd-pushback-messages-00.txt, IETF internet-draft, work in progress, July 2001.

[58] V. D. Gligor. A note on the denial-of-service problem. In *Proceedings of IEEE Symposium on Security and Privacy, Oakland, California, USA*, April 1983.

[59] J. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of 1982 IEEE Symposium on Research in Security and Privacy*, 1982.

[60] J. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of 1984 IEEE Symposium on Research in Security and Privacy*, 1984.

[61] G. S. Graham and Peter J. Denning. Protection–principles and practice. In *Proceedings of Spring Joint Computer Conference, Vol40, AFIPS Press, Montvale NJ*, 1972.

[62] R. Graubert. On the need for a third form of access control. In *Proceedings of the 12th National Computer Security Conference*, Oct 1989.

[63] J. Gray. Probablistic interference. In *Proceedings of 1990 IEEE Symposium on Research in Security and Privacy*, 1990.

[64] J. Gray and P. Syverson. A logical approach to multilevel security of probablistic systems. In *Proceedings of 1992 IEEE Symposium on Research in Security and Privacy*, 1992.

[65] B. Jonsson H. Hansson. A logic for reasoning about time and probability. In *Formal Aspects of Computing, Vol. 6, pp 512-535*, 1994.

[66] Joseph Y. Halpern and Moshe Y. Vardi. Model checking vs. theorem proving: a manifesto. pages 151–176, 1991.

[67] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. In *Communications of the ACM, Vol 19(8)*, Aug 1976.

[68] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In *Real-Time: Theory in Practice, LNCS 600*, pages 226–251, 1991.

[69] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003. ACM.

[70] SANS Institute. Egress Filtering v 0.2. http://www.sans.org/y2k/egress.htm, 2000.

[71] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *Proceedings of the Network and Distributed System Security Symposium 2002, San Diego, California* , February 2002.

[72] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Elisa Bertino. A unified framework for enforcing multiple access control policies. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data, volume 26,2 of SIGMOD Record*, pages 474–485, 1997.

[73] S. Jha and J. Wing. Survivability analysis of networked systems. In *International Conference on Software Engineering (ICSE)*, May 2001.

[74] A. K. Jones, R. J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, 1976.

[75] P. Karger and A. Herbert. An augmented capability architecture to support lattice security a nd traceability of access. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy , pp. 2-12*, 1984.

[76] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings of ACM SIGCOMM'02, (Pittsburgh, PA), August 2002*, 2002.

[77] Leonard Kleinrock. *Queueing Systems, Vol I: Theory.* Wiley InterScience, 1975.

[78] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies*, May 2001.

[79] K.J. Kristoffersen, C. Pedersen, and H.R. Andersen. Runtime verification of timed ltl using disjunctive normalized equation systems. In *RV03 - Third Workshop on Runtime Verification, Boulder, Colorado, USA*, 2003.

[80] Orna Kupferman, P. Madhusudhan, P. S. Thiagarajan, and Moshe Y. Vardi. Open systems in reactive environments: Control and synthesis. In *Lecture Notes in Computer Science, Vol 1877*, 2000.

[81] Marta Z Kwiatkowska, Gethin Norman, and Jeremy Sproston. Pctl model checking of symbolic probabilistic systems. Technical Report CSR-03-2, University of Birmingham, School of Computer Science, April 2003.

[82] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems, Reprinted ACM Operating Systems Review 1974*, Mar 1971.

[83] R. Lipton and L. Snyder. On synchronization and security. In *In R. DeMillo, D. Dobkin, A. Jones , and R. Lipton, editors, Foundations of Secure Computation, Academic Press*, 1978.

[84] P. Liu and S. Jajodia. *Trusted Recovery and Defensive Information Warfare*. Kluwer Academic Publishers,ISBN 0-7923-7572-6, 2002.

[85] Zhaoyu Liu, Prasad Naldurg, Seung Yi, Tin Qian, Roy H. Campbell, and M. Dennis Mickunas. An agent based architecture for supporting application level security. In *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 25-27, 2000.

[86] N. Long and R. Thomas. Trends in denial of service attack technology. CERT Coordination Center, Summary, October 2001.

[87] John MacLean. A comment om the "basic security theorem" of bell and la padula. In *Information Processing Letters, 20(2):67-70*, Feb 1985.

[88] Ratul Manajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network (extended version. http://www.icir.org/pushback/, July 2001.

[89] Z. Manna and A. Pnueli. Verification of temporal programs: The temporal framework. In *R.S.Boyer and J. S. Moore, editors, The Correctness Problem in Computer Science, Academic Press, New York*, 1981.

[90] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proceedings of 1987 IEEE Symposium on Research in Security and Privacy*, 1987.

[91] John McLean. Security models and information flow. In *Proceedings of 1990 IEEE Symposium on Research in Security and Privacy*, 1990.

[92] John McLean. The specification and modeling of computer security. In *IEEE Computer. 23(1)9-16*, Jan 1990.

[93] John McLean. Proving noninterference and functional correctness using traces. In *Journal of Computer Security 1(1):37-57*, Jan 1992.

[94] John McLean. Security models. In *Encyclopedia of Software Engineering, Wiley Press*, 1994.

[95] Catherine Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.

[96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[97] J. K. Millen. A resource allocation model for Denial of Service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1992.

[98] N. Minsky. Selective and locally controlled transport of privileges. In *Proceedings of the ACM Transactions on Programming Languages and Systems, 6(4)*, Oct 1984.

[99] Jelena Mirkovic, Janice Martin, and Peter Reiher. A taxonomy of ddos attacks and ddos defense mechanisms. Technical report, UCLA CSD Technical Report no. 020018, 2002.

[100] Sape Mullender. *Distributed Systems, Second Edition.* Addison-Wesley, 1995.

[101] P. Naldurg and R. H. Campbell. Modeling insecurity: Policy engineering for survivability. In *To Appear in the Proceedings of First ACM Workshop on Survivable and Self-Regenerative Systems*, Oct 2003.

[102] Prasad Naldurg and Roy 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.

[103] Prasad Naldurg, Roy Campbell, and M. Dennis Mickunas. Developing dynamic security policies. In *Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA, USA, IEEE Computer Society Press, May 29-31*, 2002.

[104] Netcraft. Ddos takes sco site down. http://news.netcraft.com/, 2003.

[105] B. Panda and J. Giordano. Defensive information warfare. In *Communications of the ACM, Vol. 42, No. 7, p. 31-32,*, July 1999.

[106] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *Proceedings of InfoCom 2001*, April 2001.

[107] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[108] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall, 1981.

[109] A. Pnueli and L. D. Zuck. Probabilistic verification by tableaux. In *In the Proceedings of the First IEEE Symposium on Logic in Computer Science*, 1986.

[110] A. Pnueli and L. D. Zuck. Probabilistic verification. In *Information and Computation*, 1993.

[111] PRISM. Probabilistic symbolic model checker. http://www.cs.bham.ac.uk/ dxp/prism/, 2004.

[112] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceeding of Fifth International Symposium on Programming, LNCS, Vol. 37, pp 337-371Springer-Verlag*, 1982.

[113] Kemmerer R. Share resource matrix methodology: An approach to identifying storage and timing channels. In *ACM Transactions on Computer Systems, Volume 1 (3):256-277*, Aug 1983.

[114] A. Sabelfeld and A. Myers. Language-based information-flow security. In *IEEE Journal on Selected Areas in Communications, 21(1)*, 2003.

[115] Andrei Sabelfeld and David Sand. Probabilistic noninterference for multi-threaded programs. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, 2000.

[116] Geetanjali Sampemane, Prasad Naldurg, and Roy Campbell. Access control for active spaces. In *Proceedings of 18th Annual Computer Security Applications Conference (ACSAC)*, 2002.

[117] R. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy, Oakland, California, USA*, Apr 1992.

[118] Ravi Sandhu. The schematic protection model, its definition and analysis for acyclic attenuation schemes. In *Journal of the ACM, Volume 35(2)*, Apr 1988.

[119] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinsten, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[120] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference, pp. 295-306*, Stockholm, Sweden, August 2000.

[121] F. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[122] Clay Shields. What do we mean by network denial of service? In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security, West Point, N.Y.*, June 2002.

[123] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer. Hash-based ip traceback. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2001.

[124] D. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. Technical report, April 2001.

[125] R. Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[126] David Sutherland. A model of information. In *In Ninth National Computer Security Conference, NBS/NCSC*, 1986.

[127] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, pages 327-338, Portland, Oregon, 21-23*, Oct 1985.

[128] Dennis M. Volpano and Geoffrey Smith. A type-based approach to program security. In *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France, April 14-18, 1997, Proceedings*, volume 1214 of *Lecture Notes in Computer Science*. Springer, 1997.

[129] Stephen Weeks. Understanding trust management systems. In *2001 IEEE Symposium on Security and Privacy*, May 2001.

[130] J. Wing. A symbiotic relationship between formal methods and security,. In *Proceedings from Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution,*, Dec 1998.

[131] J. Wray. An analysis of covert timing channels. In *In the Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, 1990.

[132] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

[133] Avi Yaar, Adrian Perrig, and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, May 2003.

[134] C. Yu and V. Gligor. A specification and verification method for preventing denial of service. IEEE Transactions on Software Engineering, 16(6):581–592, June 1990, June 1990.

Vita

Prasad Naldurg is a native of Bangalore, India. He graduated from the University of Mysore, India in 1996 with a Bachelor of Engineering degree in Computer Science and Engineering. He was ranked first in his graduating class and awarded the Jayantilal Thakore Gold Medal for obtaining highest scores in all subjects in his final year of studies there. He enrolled in the graduate program at the Department of Computer Science, University of Illinois at Urbana-Champaign in 1997. He obtained his Master of Science Degree from the University of Illinois in Computer Science in August 2000 and his Doctor of Philosophy Degree in May 2004. He was also a Visiting Lecturer at the Department of Computer Science at Illinois from August 2003 to May 2004. In Fall 2003, he taught an introductory course on computer security and cryptography entitled "Introduction to Information Assurance" for senior undergraduates and graduate students. In Spring 2004 he taught a follow-up course "Computer Security Architecture" covering advanced material related to formal methods and security. His research interests include systems and network security and applications of formal methods and cryptography to related problems.