A GENERALIZED ADVERSARY DECISION ALGORITHM AND ANALYTIC SOLUTION METHODS FOR ADVISE MODELS

BY

MICHAEL D. FORD

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Professor William H. Sanders

ABSTRACT

Quantitative security metrics are becoming increasingly important to system administrators. ADVISE generates quantitative security metrics by combining a system vulnerability graph with an adversary profile through an adversary decision algorithm. Previously, the decision algorithm placed restrictive assumptions on the adversary profile, and simulation was the sole solution method for ADVISE models. In this thesis, the decision algorithm is generalized while simultaneously improving its performance by incorporating theory from discrete-time Markov games. Furthermore, by exploring the state-space and generating the transition probability matrix, numerical solution methods may be applied to solve ADVISE models. Identifying key properties allows the models to be tested for compatibility with alternative solution methods from the literature, enabling additional metrics for ADVISE models. Finally, the performance of simulation is improved significantly by introducing decision caching. Together these accomplishments expand the number of quantitative security metrics and solution methods available to ADVISE models while lifting restrictions on the adversary profile and improving performance.

To my family

ACKNOWLEDGMENTS

The work depicted here was performed, in part, with funding from the U.S. Department of Homeland Security under contract FA 8750-09-C-0039 with the Air Force Research Laboratory. I would like to thank Douglas Maughan, Director of the Cyber Security Division in the Homeland Security Advanced Research Projects Agency (HSARPA), within the Science and Technology Directorate of the Department of Homeland Security (DHS).

I wish to thank my adviser, Dr. William Sanders, for his support, advice, mentoring, and friendship. This work would not have been possible without his guidance. I would also like to thank Dr. Steve Lumetta and Dr. Klara Nahrstedt. Learning from and working with them has been a privilege, and I owe my development to their direction. I am grateful to Dr. Peter Buchholz for his assistance with the numerical techniques. His feedback and insight on many aspects of this work have been invaluable.

This work would not have been possible without the previous work by Dr. Elizabeth LeMay. I appreciate her advice and collaboration during my first year of graduate school. Ken Keefe supported the ADVISE implementation, and our discussions led to the performance improvements in this work. I thank Jenny Applequist for all of her editorial work. Thanks to the members of the PERFORM group for great discussion and time spent around the coffee machine.

I would like to thank Dr. Carol Muehrcke of the Cyber Defense Agency, Bruce Barnett and Michael Dell'Anno of GE Global Research, and Chad Ackerman and Dan Murphy of John Deere. GE funded the ADVISE project and John Deere funded my first research into reliability modeling. I value the dedication and observations of these colleagues.

Finally, to my family and friends, thank you for loving me, for encouraging me, and for always challenging me to be my best.

TABLE OF CONTENTS

LIST OF SYMBOLS

A The set of attack steps.

a_i A single attack step.

$at : H^n_\sigma \to [0,1]$ The attractiveness of an attack policy σ.

$attr^n : X \times A \to [0,1]$ The maximum attractiveness for a given starting state and attack step.

$B_i : X \to \{True, False\}$ The attack step Boolean precondition.

$\beta : X \to a_i$ The adversary decision algorithm.

$C_i : X \to \mathbb{R}^{\geq 0}$ The attack step cost.

$CC^n : H^n \to \mathbb{R}^{\geq 0}$ The cumulative cost of a set of histories.

$\mathbf{C}(s, s')$ The mean cost incurred from s' before a state from $X_a \cup X_n$ is reached.

$\mathbf{C}^C(s, s')$ The mean cost incurred from s' before a state from X_a is reached.

c^{abs} The mean cost to absorption.

c_a^{abs} The mean cost to absorption given success.

$E_i : X \times O_i \to X$ The outcome-dependent state transition function.

$F_i : X \times O_i \to [0,1]$ The probability of non-detection for each outcome.

$\mathbf{F}$ The non-detection transition probability matrix.

f_i The frequency of attack step a_i.

$FF^n : H^n \to [0,1]$ The cumulative non-detection of a set of histories.

G The set of attack goals.

g^{abs} The average gain until a state in $X_n \cup X_a$ is reached.

Γ The set of state variables.

γ	A state variable.
H^t	The set of all histories of t steps.
h^t	A history of t steps.
H^t_σ	The set of all policy-consistent histories.
h^t_σ	A policy-consistent history.
$H^t_\sigma(\langle s \rangle)$	The set of all policy-consistent histories beginning in state s.
$\mathbf{I}$	The identity matrix.
K	The set of system knowledge.
$L : S \to [0, 1]$	The attack skill level function.
$\mathbf{M}(s, s')$	The mean time in s' before a state from $X_a \cup X_n$ is reached.
$\mathbf{M}^C(s, s')$	The mean time in s' before a state from X_a is reached.
μ_s	The exponential distribution rate.
$\mathcal{N} \in \mathbb{N}$	The adversary's planning horizon.
$\mathbf{N}_{m,m}(s, s')$	The mean number of visits to s' before a state from $X_a \cup X_n$ is reached.
$\mathbf{N}^C_{m,m}$	The mean number of visits when absorption occurs in X_a.
O_i	The finite set of attack outcomes.
Ω^t	The set of all policies with depth t.
$\Omega^t_{a_i}$	The set of all policies with depth t and attack step a_i as the first attack step.
$P : X \to \mathbb{R}^{\geq 0}$	The payoff value function.
$PP^n : H^n \to \mathbb{R}^{\geq 0}$	The cumulative payoff of a set of histories.
$\mathbf{P}$	The transition probability matrix.
$\mathbf{P}_{m,a}$	The transition probability sub-matrix for $X_m \to X_a$.
$\mathbf{P}_{m,m}$	The transition probability sub-matrix for $X_m \to X_m$.
$\mathbf{P}_{m,n}$	The transition probability sub-matrix for $X_m \to X_n$.
$\mathbf{P}^C_{m,a}$	The transition probability matrix for a successful attack.
$\mathbf{P}^{\{i\}}$	The probability transition matrix containing only transition probabilities according to $a_i \in A$.

$\mathbf{P}^{\neg\{i\}}$ The probability transition matrix containing all transition probabilities except those according to $a_i \in A$.

$\overline{\mathbf{P}}$ The probability transition matrix where each step has a duration of Δ.

p_{ndet} The probability on non-detection.

p_{succ} The probability of successfully reaching a goal.

π_l The stationary vector.

$Pr_i : X \times O_i \to [0, 1]$ The outcome probability distribution.

R The set of system accesses.

S The set of attack skills.

s A state.

s_0 The initial state.

σ A policy.

σ^t A policy of depth t.

$T_i : X \times \mathbb{R}^+ \to [0, 1]$ The attack step time distribution.

t^{abs} The mean time to absorption.

t_a^{abs} The mean time to absorption given success.

τ A finite time horizon.

$U_C : \mathbb{R}^{\geq 0} \to [0, 1]$ The cost utility function.

$U_P : \mathbb{R}^{\geq 0} \to [0, 1]$ The payoff utility function.

$U_F : [0, 1] \to [0, 1]$ The non-detection utility functions.

$(s, n).\mathcal{V}$ The valuation for a state s and reachability depth n.

w_C The attack preference weight for cost.

w_F The attack preference weight for non-detection.

w_P The attack preference weight for payoff.

X The state-space.

$\mathcal{X}$ The reachable state-space.

X_a The set of states in which the adversary has achieved their goal.

X_m	The set of states in which the adversary may achieve their goal.
X_n	The set of states in which the adversary will never achieve their goal.
$\mathcal{Y}_t$	The set of states reachable in t steps.

CHAPTER 1

INTRODUCTION

Accessible quantitative security metrics are required to help system administrators accurately assess the security of their systems. The ADVISE method has been shown to produce insightful security metrics in [1] and [2]. We generalize the ADVISE adversary decision algorithm while simultaneously introducing significant performance gains. With these additions, ADVISE becomes applicable to a broad range of situations. By introducing the performance gains, we can analyze large-scale systems and ADVISE can be incorporated into live analysis. With generalized analysis algorithms, ADVISE can generate complex metrics without placing assumptions on attacker's behavior.

The thesis is organized as follows. Chapter 2 explores related work in depth, focusing on graph-based vulnerability representations. In Chapter 3, we present the ADVISE formalism and discuss important properties of the representation. We explore the adversary's decision algorithm in Chapter 4 by generalizing the decision process, presenting decision algorithms, and discussing their performance implications. Chapter 5 discusses solution methods to produce security metrics. We explore simulation, numerical techniques, property-specific methods, and finally we compare their performance. We conclude with Chapter 6 and highlight future work.

CHAPTER 2

RELATED WORK

The field of quantitative computer security grew out of the reliability field as the need for quantitative security metrics rose. Thus, quantitative security models closely parallel their reliability counterparts. Fault trees in the reliability field became attack trees in the security field. Markov models and Petri nets have undergone numerous design iterations as they were adapted for the security field. Even failure mode and effects analysis (FMEA) has found a home in some security domains.

We examine both the work of pioneers and recent successes in the quantitative security field while focusing largely on the use of graphs in the literature. Graphs are the natural way to represent connectivity between components. In previous work, graph nodes have represented single machines, specific privilege levels, or arbitrary security assets. Typically, edges in the graph represent security vulnerabilities that can be exploited by an attacker.

More complex schema have been developed over time as researchers recognized the fact that complex interactions in state needed to be modeled. Additionally, graphs and modeling environments have been extended in order to facilitate the computation of more complex metrics. Attack graphs, multi-prerequisite graphs, and exploit dependency graphs are described in the following sections.

2.1 Privilege Graphs

Dacier and Deswarte introduced privilege graphs in [3]. In a privilege graph, nodes represent privilege states, and weighted, directed edges represent attacks. Therefore, as in most attack graphs, the privilege graph is a graph of state variables, not one which represents the underlying state-space. The edge weights represent the time or effort an attacker must

spend on the specific attack.

An attacker begins at a single node, and follows a path to their goal. The main metric used to analyze a privilege graph is the *mean effort to failure* (MEFT). This metric satisfies three main properties outlined by Dacier and Deswarte as critical metric properties:

1. Security increases if the "length of the paths" leading to the target increases.
2. Security decreases if the "number of paths" leading to the target increases.
3. Security is mainly affected by the shortest path leading to the target.

In [4] and [5], privilege graphs are converted into Petri nets [6]. The underlying Markov chain [7] can be extracted from a stochastic Petri net and used to generate the state graph. Additionally, the Markov chain can also be used to compute metrics numerically. This parallels the computation of mean time to failure (MTTF) in the reliability field.

In a privilege graph, all attacks are monotonic; that is, an attack never loses a privilege level that they previously possessed. Additionally, each attack edge has a single enabling privilege, and a single resulting privilege. Finally, there is a single attack goal in the entire graph.

2.2 Multiple-Prerequisite Graphs

Often in networked systems, computers and other resources are grouped into subnetworks. After gaining access to a subnetwork, an attacker may launch the same attack against any other computer on the subnetwork. Lippmann *et al.* recognized the potential for a reduction of the attack graph in [8]. Their realization was that access to the subnetwork, or the more general "reachability group," and not just access to the machine, is a critical security state variable. Their new graph representation is the multiple-prerequisite (MP) graph.

Vulnerabilities in a typical attack graph are represented as edges, which implies using a single state variable as the precondition and a single state variable as the outcome. Lippmann *et al.* changed the structure of the attack graph by creating two new types of nodes: the prerequisite node and the vulnerability instance node. A prerequisite node is enabled by any

of the state nodes connecting to it. The prerequisite node enables all vulnerability nodes it connects to. The outcome of a vulnerability node is the single state node it connects to.

There are two ways to view MP graphs. The first is to consider both state nodes and prerequisite nodes as state variables. In this instance, the vulnerability node has a single prerequisite, and the outcome may affect several state variables. The second is to consider prerequisite nodes as simple Boolean algebraic expressions that define the enabling conditions for a vulnerability node. Then, the precondition for a vulnerability node is complex, however the outcome affects a single state variable.

Regardless of this distinction, the MP graph can be seen as a reduction of the full graph, which improves performance by reducing the number of vulnerabilities in the graph. In the analysis of the MP graph, Lippmann *et al.* assume monotonicity. This assumption implies that the order of attack steps does not impact the resulting state. Because the main analysis performed on MP graphs is the percentage of hosts that may be compromised, the monotonic assumption does not have any affect on analysis, only on performance.

2.3 Exploit Dependency Graphs

Noel *et al.* introduced the exploit dependency graph in [9]. The exploit dependency graph is constructed from a set of exploits, and is then reduced to produce those attacks which are sufficient and necessary in order to achieve the goals. The exploits are represented as a Boolean precondition and monotonic postcondition.

The construction resembles a forward state space exploration, beginning with an initial state and applying exploits to traverse into new states. Once the full dependency graph is constructed, a backward reduction is performed, producing the attack paths that are both sufficient and necessary to reach a goal in an exploit dependency graph.

The analysis of an exploit dependency graph is to produce the set of choices of which security countermeasures with the lowest total cost to deploy for network hardening. Analysis requires a further backward traversal of the graph. Each goal's preconditions are recursively replaced with the disjunction of the preconditions of any exploit which results in the goal. This results in a Boolean condition for accessing the goal in terms of the initial conditions.

Finally, the set of hardening measures that satisfy the safety of the goals while minimizing cost are suggested.

We have closely examined the most widely used formalisms for modeling vulnerabilities in and attacks against a system. Each of those formalisms make limiting assumptions for ease of analysis. These assumptions include the monotonicity of state-variables, the exponential nature of attack time, single state-variable prerequisites, and single state-variable outcomes. These assumptions place unnecessary restrictions on the types of analysis that can be completed. For instance, the monotonic assumption implies that an active defense cannot be modeled. These assumptions, while important, should not be forced on the model. We present a general attack graph where analysis methods may be applied depending on the structure of the model.

CHAPTER 3

ADVISE FORMALISM

In this chapter, we review the ADVISE formalism, which is defined in [1] and [2]. The *attack execution graph* (AEG) is a state-based attack graph that represents system vulnerabilities that an adversary may exploit. The *adversary profile* (AP) describes the attributes of an adversary, and the *adversary decision algorithm* (ADA) selects the optimal attack step to attempt from a given state. The ADA allows the attributes in the adversary's profile to produce emergent behavior. ADVISE combines the attack execution graph and the adversary profile via the adversary decision algorithm in order to predict how an adversary attacks a system.

In this chapter, we formally define the attack execution graph, present properties of the attack execution graph, define the adversary profile, and define the optimal conditions for the adversary decision algorithm.

3.1 The Attack Execution Graph

An attack execution graph is defined by the tuple

$$\langle A, R, K, S, G \rangle,$$

where

A is the set of attack steps,

R is the set of system accesses,

K is the set of system knowledge,

S is the set of attack skills,

and G is the set of attack goals.

3.1.1 Notion of State

The sets R, K, and G are sets of Boolean state variables. S is a set of skill variables that are statically defined for each adversary. Since an adversary's evaluation of all skill variables is static and known *a priori*, S is not included in the state. This evaluation is discussed further in Section 3.3.

We define:

$\Gamma = R \bigcup K \bigcup G$ as the set of state variables.

$\gamma \in \Gamma$ as a state variable.

$X = \mathcal{P}(\Gamma)$ as the state-space, or the set of all possible states.

$s \in X$ as a state. Therefore, $s \subseteq \Gamma$.

This representation of state relies on the assumption that all state variables in Γ are Boolean. Therefore, the two values a state variable takes on are represented by the state variable name being included in, true, or excluded from, false, the set that represents the state of the model. This assumption is just for notational convience; it is simple to extend the notion of state to a discrete set of values. We use this representation for its simplicity, especially with respect to the attack graph properties in Section 3.2.

3.1.2 Attack Steps

An attack step $a_i \in A$ is defined by the tuple

$$a_i = \langle B_i, T_i, C_i, O_i, Pr_i, F_i, E_i \rangle,$$

where

$B_i : X \rightarrow \{True, False\}$ is the attack step Boolean precondition,

$T_i : X \times \mathbb{R}^+ \rightarrow [0, 1]$ is the attack step time distribution,

$C_i : X \rightarrow \mathbb{R}^{\geq 0}$ is the attack step cost,

O_i is the finite set of attack outcomes,

$Pr_i : X \times O_i \rightarrow [0, 1]$ is the outcome probability distribution,

$F_i : X \times O_i \rightarrow [0, 1]$ is the probability of non-detection for each outcome,

$E_i : X \times O_i \rightarrow X$ is the outcome-dependent state transition function.

3.2 Properties of an Attack Execution Graph

In this section, we list properties of an attack execution graph (AEG) that are commonly used in the security metrics literature discussed in Chapter 2. The following properties are often required in order for a specific type of analysis to be performed. These properties show the modeling power of the AEG and allow us to identify which types of analysis can be used with a specific AEG.

Exponential (in Time)

The timing distribution is exponential for all attack steps in the AEG.
If $T_i(s) = \lambda e^{-\lambda t}$, $\lambda \in \mathbb{R}^+$, $\forall a_i \in A$, then the AEG is exponential in time.

Static

- Static with respect to Cost:
  If, for attack step i, $C_i(s) = c$, $\forall s \in X$, then attack step i is static w.r.t. cost.
- Static with respect to Time:
  If, for attack step i, $T_i(s) = \tau$, $\forall s \in X$, then attack step i is static w.r.t. time.
- Static with respect to Outcome Probability:
  If, for attack step i, $Pr_i(s) = p$, $\forall s \in X$, then attack step i is static w.r.t. outcome probability.

- Static with respect to Detection Probability:

  If, for attack step i, $D_i(s) = d$, $\forall s \in X$, then attack step i is static w.r.t. detection probability.

- Static with respect to State Transition:

  Let $e^+ \triangleq E_i(\emptyset, o)$ and $e^- \triangleq E_i(\Omega, o)$.

  If, for attack step i, $E_i(s, o) = s'$ and $s' \setminus s \subseteq e^+$ and $s \setminus s' \subseteq e^-$, $\forall o \in O_i$, $\forall s \in S$, then attack step i is static w.r.t. state transition.

If all attack steps in the AEG are static with respect to cost, time, outcome probability, and state transition, then the AEG is static.

Simple Enabling Conditions

An attack step's enabling condition depends on a single state variable.
The AEG has simple enabling conditions if $\exists$ a state variable $\gamma \in \Gamma$
such that $B_i(s) = \begin{cases} True & \gamma \in s \\ False & \gamma \notin s \end{cases}$.

Monotonic Enabling Conditions

An AEG has monotonic enabling conditions, if, when an attack step is enabled for a given state, the addition of any state variables to that state may not disable the attack step.
Formally, if $B_i(s) = True \Rightarrow B_i(s \cup s') = True$, $\forall s' \in X$, $\forall a_i \in A$, then the AEG has monotonic prerequisites.

Singular Outcomes

An AEG has singular outcomes, if, there is only a single outcome per attack step.
Formally, if $|O_i| = 1$ $\forall a_i \in A$, then the AEG has singular outcomes.

Simple Outcomes

An AEG has simple outcomes, if, any outcome transition from a give state affects only one state variable.
Formally, if E_i is of the form $E_i(s,o) = \begin{cases} s \\ s \cup \gamma \end{cases} \quad \forall o \in O_i, \forall s \in X, \forall a_i \in A$, then the AEG has simple outcomes.

Monotonic Outcomes

An AEG has monotonic outcomes, if, for any state, a transition from that state may never relinquishes a state variable. In effect, this means that once an adversary gains an access, a piece of knowledge or a goal, they never lose it by attempting another attack step.
Formally, if $s' = E_i(s,o) \Rightarrow s \subseteq s'$, $\forall o \in O_i$, $\forall s \in X$, $\forall a_i \in A$, then the AEG has monotonic outcomes.

It is important to note that the properties listed above are restrictions placed on an attack execution graph. Therefore, an attack execution graph is inherently not exponential, static, monotonic, or simple. However, these properties apply to other representations, and the analysis techniques used with those representations rely on assumptions that the properties provide. The AEG properties enable one to compare the assumptions of various representations and identify which analysis techniques can be applied to a representation.

3.3 The Adversary Profile

The adversary profile is defined by the tuple

$$\langle s_0, L, P, w_C, w_P, w_F, U_C, U_P, U_F, N \rangle,$$

where

$s_0 \in X$ is the initial model state,

$L : S \to [0, 1]$ is the attack skill level function,

$P : X \to \mathbb{R}^{\geq 0}$ is the payoff value function,

w_C, w_P, $w_F \in [0, 1]$ are the attack preference weights,

$U_C : \mathbb{R}^{\geq 0} \to [0, 1]$ is the cost utility function,

$U_P : \mathbb{R}^{\geq 0} \to [0, 1]$ is the payoff utility function,

$U_F : [0, 1] \to [0, 1]$ is the non-detection utility functions,

$N \in \mathbb{N}$ is the adversary's planning horizon.

Each adversary may begin with a unique initial state, s_0. The attack skill level function, L, translates skill variables in an AEG into adversary specific values. The value of a state to the adversary is given by the payoff value function, P. The utility functions U_C, U_P, and U_F convert the cost, payoff, and non-detection probability of an attack path into units of utility for the adversary. These utility values are weighted by the corresponding preference weights (w_C, w_P, and w_F) and then summed to create a measure of attractiveness of an attack path to the adversary. Finally, the adversary can plan N attack steps into the future.

The attack execution graph is a rich modeling formalism for describing the vulnerabilities in a system. We have identified key properties of the AEG that help us relate an AEG to other representations. Finally, the adversary profile provides a descriptive environment to model adversaries. The next chapter will explore the interaction between the adversary profile and the attack execution graph.

CHAPTER 4

THE ADVERSARY DECISION ALGORITHM

The adversary decision is the way in which the adversary profile combines with the attack execution graph to produce security metrics. In a given state, the adversary selects an attack step to attempt, the attack step's outcome is selected probabilistically, and the state changes based on the outcome effects. Then the cycle repeats.

The *Adversary Decision Algorithm* (ADA), or $\beta(s)$, dictates the way in which the adversary profile values deterministically influence the attack step selection. A specific recursive decision algorithm is presented in [1] and [2]. In this chapter, we introduce a generalization of the adversary decision algorithm, define an optimality condition, present specific decision algorithms, optimize them, and analyze their performance.

4.1 Generalizing the Attack Step Selection

The attack step selection explores the future state space and determines which attack step provides the optimal future based on their specific weights and utility functions in the adversary profile. This future value is called the attractiveness. Here, we present a formal framework for defining the optimal attack step for an adversary to attempt in a given state.

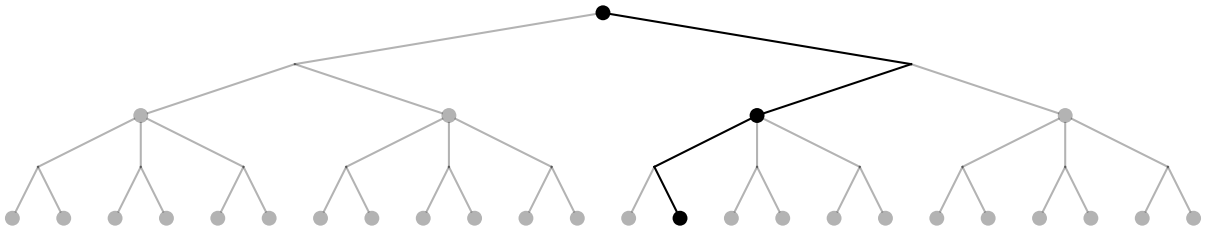
4.1.1 Exploring the Future

The adversary's decision is similar to one made in a stochastic game. Since a Markov perfect equilibrium exists in a stochastic game [10], it is possible to find the optimal adversary decision for an ADVISE model. Therefore we borrow the concepts of event histories and decision policies from the field of game theory.

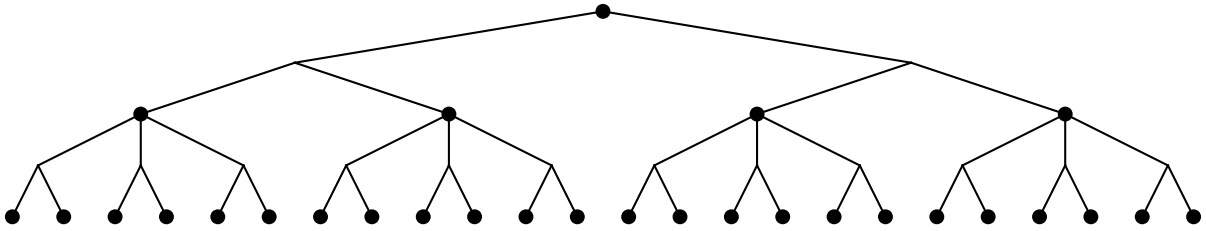
The term h^t denotes a history looking t steps into the future:

$$h^t = \langle s_0, a_0, o_0, s_1, \ldots, s_{t-1}, a_{t-1}, o_{t-1}, s_t \rangle.$$

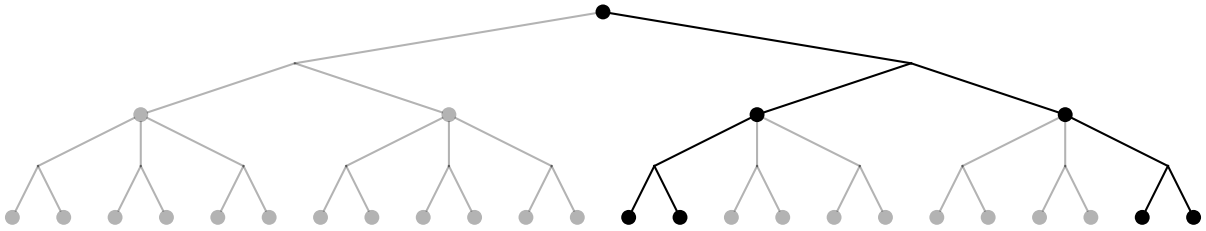
A single history consists of a sequence of states $s_0, \ldots, s_t \in X$ such that a state s_i transitions to s_{i+1} based on the effects of $o_i \in O_{a_i}$ for $0 \leq i < t$, or more formally, $s_{i+1} = E(s_i, o_i)$, $o_i \in O_{a_i}$. We use the notation $s_i \xrightarrow{a_i, o_i} s_{i+1}$ for compactness. Figure 4.1(a) depicts a single history h^t within the set of all possible histories extending from the initial state.



(a) h^t: A Single Path



(b) H^t: The Set of All Possible Paths



(c) H^t_σ: The Set of Possible Paths Given a Policy

Figure 4.1: History Trees

Let $H^t(\langle s_0 \rangle)$ denote the set of all possible histories h^t, with look-ahead t, that begin with state s_0. We extend this definition recursively as follows:

$$H^t(H^s) = \begin{cases} H^s & t = s \\ \{h^{t-1} \oplus \langle a, o, E_a(s_{t-1}, o) \rangle \text{ s.t.} & t > s \\ \quad h^{t-1} = (s_0, \ldots, s_{t-1}) \in H^{t-1}(H^s) & \\ \quad a \in A : B_a(s_{t-1}) = true, \text{ and } o \in O_a \}, & \end{cases}$$

where the symbol $\oplus$ represents the append operator that appends a new step, outcome, and state to a history, i.e., $\langle s_0, a_0, o_0, s_1, \ldots, s_{t-1}, a_{t-1}, o_{t-1}, s_t \rangle \oplus \langle a_t, o_t, s_{t+1} \rangle = \langle s_0, a_0, o_0, \ldots, a_t, o_t, s_{t+1} \rangle$.

Figure 4.1(b) depicts the set of all possible histories H^t within the state space of depth t. In this figure, from the initial state, the adversary selects one of two attack steps, each of which will result in either of two possible outcomes. From those resulting states, the adversary may select from three attack steps, again with two possible outcomes each.

Notice that h^t is a single path through the state space, while H^t represents the entire set of paths through the state space of depth t. A history h^t is the path an adversary could take if both attack steps and outcomes were fixed. H^t is the set of paths an adversary could take if neither the attack steps nor outcomes were fixed. What remains is to depict the potential paths of an adversary given the deterministic choice of the best attack step with respect to state and the non-deterministic nature of attack step outcomes.

Therefore, a policy σ is defined to be a mapping at each look-ahead level from a state to an attack step, and σ^t is defined to be a policy with a look-ahead of t. Then $\sigma^t = (f_1, \ldots, f_t)$, where $f_k : X \to A$, i.e., $f_k(s) = a$ means that attack a is chosen in step k. The set of all policies with look-ahead t is Ω^t, and $\Omega^t_{a_i}$ is the set of policies with look-ahead t and attack step a_i as the first attack step. Note that set $\Omega^t_{a_i}$ is empty in state s if $B_i(s) = false$. A policy-consistent history, h^t_σ, is a history such that $\forall k$, $f_k(s_k) = a_k$. That is, the history follows the attack steps prescribed by the policy.

Then, the set of all policy-consistent histories, denoted by $H^t_\sigma(\langle s \rangle)$, is the set of all possible sample paths through the state space starting with state s, given that the adversary

follows the policy σ for selecting attack steps and only the outcomes are stochastic. See Figure 4.1(c).

4.1.2 Policy Attractiveness

With those notations, we can extend the notation for reward values that are required for the computation of the attractiveness. In particular, let

$$CC^n(H^n(\langle s\rangle)) = \sum_{h^n \in H^n} \left(\sum_{i=0}^{n-1} C_{a_i}(s_i) \times \prod_{i=0}^{n-1} Pr_{a_i}(s_i, o_i) \right)$$

$$PP^n(H^n(\langle s\rangle)) = \sum_{h^n \in H^n} \left(\sum_{i=0}^{n-1} G_{a_i}(s_i, o_i) \times \prod_{i=0}^{n-1} Pr_{a_i}(s_i, o_i) \right)$$

$$FF^n(H^n(\langle s\rangle)) = \sum_{h^n \in H^n} \left(\prod_{i=0}^{n-1} F_{a_i}(s_i, o_i) \times \prod_{i=0}^{n-1} Pr_{a_i}(s_i, o_i) \right).$$

The values are the cumulative cost (CC^n), gain (PP^n), and probability that the attack is not detected (FF^n) over all possible histories of length n. To help understand these formulae, we describe how the cumulative cost is computed. First, the sum of the costs for all of the attack steps in a single history is computed. Then this sum is weighted by the probability of that history, that is by the product of the probabilities for each outcome in the history. The sum of the probabilistically weighted cost of each history is the cumulative cost for the policy.

With those values, the attractiveness of a policy σ is given by converting the cumulative cost, payoff, and probability of nondetection to a common unit of adversary utility and weighting the values by the adversary's preference weights:

$$\begin{aligned} at(H^n_\sigma(\langle s\rangle)) = {} & w_C U_C(CC^n(H^n_\sigma(\langle s\rangle))) + w_P U_P(PP^n(H^n_\sigma(\langle s\rangle))) \\ & + w_F U_F(FF^n(H^n_\sigma(\langle s\rangle))) \end{aligned} \tag{4.1}$$

Then, the attractiveness of an attack step, a_i, is the maximum of all policy attractivenesses

where the policy dictates that a_i be attempted from state s:

$$attr^n(s, a_i) = \max_{\sigma \in \Omega^n_{a_i}} \left(at(H^n_\sigma(\langle s \rangle)) \right). \tag{4.2}$$

Observe that the functions $U_{(.)}$ in (Eq. 4.1) can be arbitrary non-linear functions. Function β is defined for a look-ahead of N as

$$\beta(s) = \operatorname*{argmax}_{a_i \in A} \left(attr^N(s, a_i) \right). \tag{4.3}$$

Therefore, $\beta(s)$ gives the attack step that results in the most attractive future based on the adversary's utility of the cumulative cost, gain, and probability of non-detection.

In general, as already noted in [2, Theorem 3.2], the computation of $\beta(s)$ requires the computation of $at(s, .)$ for all policies of length N, which means that for each of the $(S \cdot A)$ policies, $|A|^{|O|^{N-1}}$ paths have to be considered. That becomes impractical for larger values of N. Therefore, we consider cases where the computation over all paths can be avoided. Of course, that implicitly or explicitly puts some restrictions on the functions $U_{(.)}$. However, restrictions, such as monotonically increasing utility functions, seem to be natural.

4.1.3 Defining an Optimality Ordering

In order to reduce the computational complexity of $\beta(s)$, we define an ordering on the cumulative cost, payoff, and probability of nondetection values. This ordering enables the direct comparison of two policies at intermediate locations in the $\beta(s)$ computation. If one policy sufficiently outperforms a second policy, the second policy can be ingored in the remainder of the $\beta(s)$ computation.

The attractiveness of using the policy σ in state s is characterized by three values: $c = CC^n(H^n_\sigma(\langle s \rangle))$, $p = PP^n(H^n_\sigma(\langle s \rangle))$, and $d = FF^n(H^n_\sigma(\langle s \rangle))$. We define an order $\gg$ on the triples (c, p, d).

To formally define $(c,p,d) \gg (c',p',d')$, let $\sigma_1^t, \sigma_2^t \in AS^t$ such that:

$$(c,p,d) = \left(CC^n(H^n_{\sigma_1^t}(\langle s\rangle)), PP^n(H^n_{\sigma_1^t}(\langle s\rangle)), FF^n(H^n_{\sigma_1^t}(\langle s\rangle))\right)$$
$$(c',p',d') = \left(CC^n(H^n_{\sigma_2^t}(\langle s\rangle)), PP^n(H^n_{\sigma_2^t}(\langle s\rangle)), FF^n(H^n_{\sigma_2^t}(\langle s\rangle))\right)$$

We use the notation $(c,p,d) \gg^n (c',p',g')$, $n > t$ if:
$at\left(H^{l+t}_{\sigma^l\sigma_1^t}(s)\right) \geq at\left(H^{l+t}_{\sigma^l\sigma_s^t}(s)\right) \quad \forall l \in \{1,\ldots,n-t\}$, $\forall \sigma^l \in \Omega^l$, and $\forall s \in X$.
The notation $(c,p,d) \gg (c',p',d')$ is used if $(c,p,d) \gg^n (c',p',d') \quad \forall n \in \mathbb{N}^+$.

Usually, the adversary's utility functions $U_{(.)}(.)$ are of such a form that $(c,p,d) \gg (c',p',d')$ holds if $c \leq c'$, $p \geq p'$, and $d \geq d'$, i.e., the costs are smaller and the gain and the probability of not being detected are higher. However, there may be other cases, depending on the definition of the utility functions.

Specifically, we consider the Markov decision process case (*MDP-case*), where an adversary's decisions are independent of the past. Markov decision processes are discussed in general in [11], while their impact on the adversary decision is discussed in [2]. For this case, the adversary decision function, $\beta^n(s)$, can be defined recursively. Then,

$$\beta^1(s) = \underset{a_i \in A}{\operatorname{argmax}} \left(w_C U_C(C_i(s)) + w_P U_P(P_i(s)) + w_F U_F(F_i(s))\right)$$

with $C^1_*(s) = C_{\beta^1(s)}(s)$, $P^1_*(s) = P_{\beta^1(s)}(s)$, and $D^1_*(s) = D_{\beta^1(s)}(s)$. That makes it possible to recursively compute $C^n_*(s), P^n_*(s), D^n_*(s)$ as well as $\beta^n(s)$ (see [2, Eq. (3.8)-(3.12)]. Let $v = w_C U_C(c) + w_P U_P(p) + w_F U_F(d)$ and $v' = w_C U_C(c') + w_P U_P(p') + w_F U_F(d')$.

Then in the MDP-case, the relation $\gg$ is defined as follows:

$$v > v' \Leftrightarrow (c,p,d) \gg (c',p',d').$$

Furthermore, if the two values are identical, we define

$$v = v' \Leftrightarrow (c,p,d) \approx (c',p',d'),$$

and in this case, it does not matter whether we choose (c,p,d) or (c',p',d'). Consequently,

in the MDP-case $\gg$ and $\approx$ define a total order, i.e., two values (c, p, d) and (c', p', d') are in relation $\gg$ or in relation $\ll$ or in relation $\approx$. Below, we use the notation $(c, p, d) \gtrapprox (c', p', d')$ for $(c, p, d) \gg (c', p', d')$ or $(c, p, d) \approx (c', p', d')$ and use this relation in Algorithm 3 in the next section for evaluating function $\beta(s)$.

4.2 Computation of the Adversary Decision Algorithm

In the most general case, the function $\beta()$ must evaluate all policies from Ω^N, and therefore, all histories H^N_σ, $\sigma \in \Omega^N$. This can be done using Algorithm 1, in which $\bullet$ denotes the *undefined* value.

Algorithm 1 Simple $\beta(s)$ Computation

1: $\beta = \bullet$ and $best\_val = -\infty$;
2: **for all** $\sigma \in \Omega^N$ **do**
3: determine $val = at(H^n_\sigma(\langle s \rangle)$ using (4.1);
4: **if** $val > best\_val$ **then**
5: $best\_val = val$;
6: $\beta = i_1$;
7: **return** β;

Algorithm 1 is rather naive and may compute values recursively several times. If the relation $\gtrapprox$ is available, a graph-based approach that can exploit the relation is much more efficient. In that case, successor states up to N steps apart are explored and an acyclic multigraph is generated. This alternative algorithm for determining $\beta(s)$ consists of a forward and a backward phase as given in Algorithm 2 and 3 respectively. In the forward phase, all states are generated that can be reached in $1, 2, \ldots, N$ steps (Lines 4-7), and they are stored in sets $\mathcal{Y}_n$ (Line 8). Furthermore, all transitions $s \overset{(a_i,o)}{\rightarrow} s'$ with $s \in \mathcal{Y}_n$ and $s' \in \mathcal{Y}_{n+1}$ are stored with the source states to be used in the subsequent backward phase (Line 9). Since several transitions can exist between two states s and s', the graph is a multigraph.

In the subsequent backward phase, the values are evaluated bottom-up in the multigraph. Since the valuation is computed bottom-up, we store a set of valuations $\mathcal{V}$ with each state-level pair $\langle s, n \rangle$ in the form $(s, n).\mathcal{V}$. A valuation is of the form (c, p, d), with the same

Algorithm 2 Forward $\beta(s)$ Computation

$\mathcal{Y}_0 = \{s_0\}$;
for $n = 1$ to N **do**
  $\mathcal{Y}_n = \emptyset$;
  for all $s \in \mathcal{Y}_{n-1}$ **do**
    for all $a_i \in A$ with $B_i(s) = true$ **do**
      for all $o \in O_i$ **do**
        let $s' = E_i(s, o)$;
        $\mathcal{Y}_n = \mathcal{Y}_n \cup \{s'\}$;
        add an edge between s and s' labeled with (a_i, o) to state s;
return $\mathcal{Y}_0, \ldots, \mathcal{Y}_N$;

meaning described above. For the states $s \in \mathcal{Y}_N$, $(s, N).\mathcal{V} = \{(0, 0, 1)\}$ is used for initialization; the other values are computed in Algorithm 3.

Algorithm 3 Backward $\beta(s)$ Computation

$\beta = \bullet$ and $best\_val = -\infty$;
for $n = N - 1$ downto 0 **do**
  for all $s \in \mathcal{Y}_n$ **do**
    for all $a_i \in A$ with $B_i(s) = true$ **do**
      for all combinations $(c', p', d') \in (s', n+1).\mathcal{V}$ where $s \xrightarrow{(a_i,o)} s'$ **do**
        $c = C_i(s) + \sum_{o \in O_i} Pr_i(s, o)c'$;
        $p = \sum_{o \in O_i} Pr_i(s, o)(G_i(s, o, s') + g')$;
        $d = \sum_{o \in O_i} Pr_i(s, o)(F_i(s, o)d')$;
        if $n = 0$ **then**
          $val = w_C U_C(c) + w_P U_P(p) + w_F U_F(d)$;
          if $val > best\_val$ **then**
            $best\_val = val$;
            $\beta = a_i$;
        else if no $(c, p, d)' \in (s, n).\mathcal{V}$ with $(c, p, d)' \underline{\gg} (c, p, d)$ exists **then**
          $(s, n).\mathcal{V} = (s, n).\mathcal{V} \cup \{(c, p, d)\}$;
          remove all $(c, p, d)'$ from $(s, n).\mathcal{V}$ with $(c, p, d) \underline{\gg} (c, p, d)'$;
return β;

Step 5, in which all combinations of possible valuations in successor states have to be evaluated, is crucial. Since O_i usually has two outcomes, success or failure, the number of possible valuations grows with the product of the valuations in the successor states. In the worst case, the number of valuations in the algorithm equals $N \cdot |A|^N$, as for Algorithm 1. However, if states contain identical successor states or $\underline{\gg}$ can be used to reduce the number of valuations, then the effort is reduced. In the MDP-case, we only need to store one valuation

per state, which implies that the effort equals the number of edges in the graph defined by $\mathcal{Y}_0, \ldots, \mathcal{Y}_N$.

A further improvement can be achieved if results are reused in consecutive $\beta(s)$ calculations. We store the values of $(s, n).\mathcal{V}$ between $\beta(s)$ calculations. If, in a subsequent forward phase, state $s \in \mathcal{Y}_n$ is generated, $(s, n).\mathcal{V}$ can be used directly, and no successors of s need to be generated, since the evaluation for the backward phase is already available. Therefore, the condition **if** $|(s, n).\mathcal{V}| > 0$ **then** may be inserted into Algorithm 2 at line 5, surrounding the inner two **for** loops and stopping further exploration of the path if the valuation is available. That reuse of results should improve runtimes significantly for most models. If memory becomes a problem, only values from the upper levels of $\mathcal{Y}_n$ (where n is small) are stored, since their reuse introduces the largest benefits.

In [2], the adversary decision algorithm is a depth-first search of the state space in N steps. The ADA does not utalize caching, so parts of the state space may be explored repeatedly. Additionally, in order for the ADA in [2] to generate the optimal attack step, the adversary's utility functions must follow a specified form. This form allows for the recursive combination of the cost, detection, and payoff to satisfy Bellman's Equation.

4.3 Performance of the Adversary Decision Algorithm

The time to complete the adversary decision algorithm corresponds directly to the number of states that are explored. For typical models, the complexity can be hard to compute since the number of enabled attacks is not known *a priori*. However, for a general model in which all attack steps are enabled, and each attack step has a fixed number of outcomes, O, the recursive algorithm in [2] visits $|A|^{|O|^N}$ states and is therefore exponential in the look-ahead.

4.3.1 Algorithm Comparison

The simplest method of executing the adversary decision algorithm is to use the depth-first, recursive approach. That method leads to $|A|^{|O|^N}$ states being explored, as mentioned above. An improvement is to use a graph-based approach of Algorithms 2 and 3, caching

intermediate valuations and reusing them if the same state is encountered again at the same look-ahead level. A comparison of the depth-first exploration and the graph-based exploration are presented in Figure 4.2(a) and 4.2(b) respectively.

In practice, we cache the cost, detection probability, and expected payoff of the sub-graph for each state and look-ahead level pair rather than construct the entire graph and iterate over it. This is the caching extension to Algorithms 2 and 3 outlined in the previous section. The nature of Algorithms 2 and 3, as well as the addition of the caching technique, significantly reduce the branching of the decision algorithm.

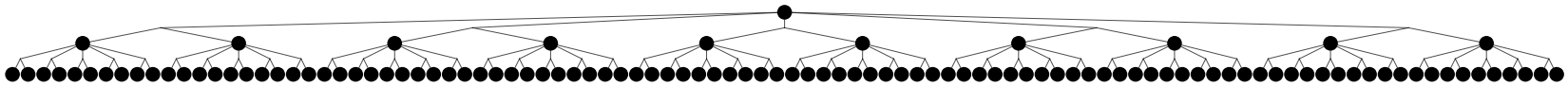
Because all intermediate values are cached, the number of states explored is bounded by the look-ahead times the size of the state space ($N * 2^{|A||K||G|}$). That can still be very large; however, because of the sequential nature of attacks, in typical models, the reachable state space is significantly smaller than the potential state space.

4.3.2 Performance Data

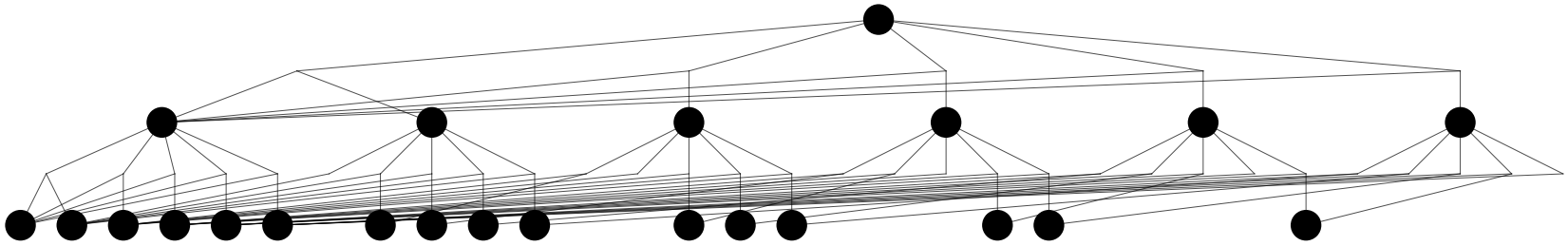
In Table 4.1, we present full execution timings for the base implementation of the recursive algorithm and when caching was used within Algorithms 2 and 3 with caching. Those timings are generated from an easily extensible model.

The model consists of eight accesses, eight attack steps, and seven goals. One access is designated as the initial access. Then, seven of the attack steps are set up in order, so that they can only be performed sequentially. The eighth attack step is always enabled. Additionally, each attack step has two outcomes, success and failure. The structure of the model provides the exact number of states exlpored in the decision algorithm, since two attack steps are always enabled, and each attack step has two outcomes. The number of states explored is 4^N, where N is the look-ahead.

The recursive algorithm scales poorly with respect to the look-ahead since it is exponential. By caching within the algorithm, essentially converting the tree to a graph, we achieve significant improvements. Most notably, the caching algorithm scales linearly with respect to the look-ahead.



(a) Depth-First Exploration



(b) Graph-Based Exploration

Figure 4.2: ADA Exploration with a Lookahead of 2

Table 4.1: Simulation Timing (in seconds)

Look-ahead	Recursive Algorithm	Caching in Algorithms 2 & 3
1	0.87	1.12
2	1.71	2.59
3	4.79	5.23
4	17.80	8.96
5	69.12	13.89
6	289.81	19.78
7	1134.02	26.99
8	4303.11	34.63
9	18455.96	42.50

Applying theory from stochastic games in order to allow the adversary utility functions to take on general forms and using a forward-backward algorithm augmented with caching to improve performance are both significant accomplishments. However, without the optimality ordering, there would be no way to combine the two. While the ordering does place some restrictions on the adversary utility functions, they are not as strict as the previous optimality requirements. By using the optimality ordering within the forward-backward algorithm with caching, performance is significantly improved while placing as few restrictions on the adversary profile as possible.

CHAPTER 5

SOLUTION METHODS

In this chapter, we identify solution methods to produce quantitative security metrics from the combination of the attack execution graph and the adversary profile.

5.1 State-Space and Transition Matrix Generation

Exploring the reachable state-space offers an alternative to the on-the-fly adversary decision and attack step outcome used during simulation. It also enables the creation of the transition matrix, which is required for the numerical solution methods presented in Section 5.3. In this section, we present the algorithm used for state-space generation and discuss the generation of the transition matrix.

5.1.1 State-Space Generation

Let $\hat{X}$ be the potential state space that contains all combinations of values of the Boolean state variables and, consequently, has a cardinality of $2^{|R|\cdot|K|\cdot|G|}$. In ADVISE models, the potential state space may be too large to be enumerated; however, for typical models, the reachable state space X usually only contains a small subset of the states.

For the basic state-space generation algorithm, we use a data structure $\mathcal{U}$ to store state descriptions and a data structure $\mathcal{X}$ to store state description plus the adversary decision $\beta(s)$. Then, Algorithm 4 presents a generic algorithm for state-space generation.

For finite state spaces, the algorithm terminates after computing the set of states and the index of the attack step to be performed in the state. That information is sufficient to characterize the complete stochastic process.

Algorithm 4 State Space Generation

```
1: U = {s_0};
2: while U ≠ ∅ do
3:    remove s from U;
4:    a_i = determine_beta(s);
5:    for all o ∈ O_i do
6:       s' = E_i(s, o);
7:       if s' ∉ X ∪ U ∪ {s} then
8:          U = U ∪ {s'};
9:    X = X ∪ {(s, a_i)};
```

5.1.2 Transition Matrix Generation

After the state-space generation, $\mathcal{X}$ contains the selected attack steps together with the transitions. Therefore, it contains all information necessary for the analysis steps. Let m be the cardinality of the state space. We describe the underlying stochastic process now in terms of vectors and matrices. The dimension of the vector is $1 \times m$ or $m \times 1$, and the order of the matrices is $m \times m$.

Let $\mathbf{P}$ be the transition matrix of the embedded discrete-time Markov chain (DTMC) that considers the state of the system when an attack steps ends. We have

$$\mathbf{P}(s, s') = \sum_{o \in O_{\beta(s)}} \delta(E_{\beta(s)}(s, o) = s')\, Pr_{\beta(s)}(s, o) \text{ s.t. } \delta(b) = \begin{cases} 1 & b = true \\ 0 & b = false \end{cases}$$

Since $\mathbf{P}$ is usually not irreducible, the state space may be analyzed and reordered before the matrix is generated. The analysis is done using an algorithm, such as Tarjan's algorithm [12], to detect strongly connected components in the graph. Then X can be decomposed into X_0, which is the set of transient states, and $X_1, \ldots, X_L$, which are L absorbing, irreducible subsets. As long as X_0 is nonempty, we may generate $\mathbf{P}$ according to this decomposition.

In doing so, we obtain the following structure:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \cdots & \cdots & \mathbf{P}_{0,L} \\ \mathbf{0} & \mathbf{P}_{1,1} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \mathbf{0} & \mathbf{P}_{2,2} & \mathbf{0} & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{P}_{L,L} \end{pmatrix}.$$

$\mathbf{P}_{0,0}$ is a sub-stochastic matrix for the subset X_0, and

$$\mathbf{N}_{0,0} = (\mathbf{I} - \mathbf{P}_{0,0})^{-1} = \sum_{k=0}^{\infty} (\mathbf{P}_{0,0})^k$$

exists and is nonnegative [7]. Moreover, $\mathbf{N}_{0,0}(s, s')$ equals the mean number of visits to state s' before the set of transient states, X_0, is left if the current state is s. s_0 belongs to X_0, and we can assume that it is the first state in the set. Often $\mathbf{P}_{0,0}$ has an additional structure such that $\mathbf{N}_{0,0}$ will not become a full matrix. However, this point will not be considered here. Using standard arguments from absorbing Markov chains [7], we find that the probability of entering subset X_l from the initial state equals

$$pe_l = \mathbf{e}_1 \mathbf{N}_{0,0} \mathbf{P}_{0,l} \mathbb{1} = \mathbf{n}_{1*} \mathbf{P}_{0,l} \mathbb{1},$$

where $\mathbb{1}$ is a column vector of 1 of appropriate length. Since we have a unique initial state s_0, only the first row of matrix $\mathbf{N}$ is required which results from the solution of $\mathbf{n}_{1*}(\mathbf{I} - \mathbf{P}_{0,0}) = \mathbf{e}_1$.

If X_0 is empty, then the set of states is irreducible, and we can use matrix $\mathbf{P}$ as it is; we assume that s_0 is the first state. Thus, the initial vector is defined as $\mathbf{e}_1$, which is a row vector in which the first element is 1 and all remaining elements are 0.

5.2 Simulation

The first method of analysis is simulation. Inherently, simulation explores a single path that an adversary may take while attacking the system. The results of the single run are aggre-

gated over thousands of runs, so that the resulting mean result has statistical significance.

There are two methods for supplying the simulation loop with the adversary decision and attack step outcome. The first method is to generate the values as needed. That is, for each loop of the simulation, the adversary's decision is recalculated, and an attack step outcome is selected probabilistically. The second option is to generate the state transition matrix as detailed in the previous section (Sec. 5.1), and use it to drive the simulation.

The on-the-fly approach may be used if the state-space is extremely large, and the transition matrix can not be generated due to memory limitations. There are methods to reduce the number of adversary decision computations performed over the course of all of the simulation runs. This will be discussed in detail in Section 5.6. Additionally, once the transition matrix is generated, many metrics may be produced directly, as detailed in the following section.

5.2.1 Simulation with As-Needed Adversary Decision Computations

A typical simulation loop consists of identifying enabled transitions, selecting one probabilistically, and changing model state. An example ADVISE simulation algorithm is presented in Algorithm 5. By including the adversary decision in the simulation loop (Line 6), we require an optimization problem to be solved for each state transition. The inclusion of the adversary decision poses a significant performance hurdle.

Algorithm 5 Simple As-Needed Simulation

1: $r = 0$;
2: **while** $r <$ MaxRuns **do**
3: $\quad s = s_0$;
4: $\quad t = 0$;
5: $\quad$ **while** $t < \tau$ **do**
6: $\quad\quad a_i = \beta(s)$;
7: $\quad\quad o = O$, $O \sim Pr_i(s)$;
8: $\quad\quad t = t + T$, $T \sim T_i(s)$;
9: $\quad\quad s = E_i(s, o)$;

The exponential adversary decision algorithm is executed for each state the adversary enters, over many simulation iterations. However, the adversary's decision is static with respect

to the state. By caching the decision associated with a state, as presented in Algorithm 6, we avoid subsequent state-space exploration, and the complexity of the decision algorithm is reduced from exploring attack paths to a simple lookup. The cache quickly returns the adversary's decision if the adversary revisits a state.

Algorithm 6 As-Needed Simulation with Caching

```
1: r = 0;
2: Q(s) = • ∀s ∈ X;
3: while r < MaxRuns do
4:    s = s_0;
5:    t = 0;
6:    while t < τ do
7:       if Q(s) = • then
8:          Q(s) = β(s);
9:       a_i = Q(s);
10:      o = O, O ∼ Pr_i(s);
11:      t = t + T, T ∼ T_i(s);
12:      s = E_i(s, o);
13:   r = r + 1;
```

In addition, the cache can be used for more than a single simulation iteration. In order to return accurate metrics within a specified confidence interval, a given experiment is run many times. Since nothing in the underlying model changes between iterations, we can continue to use the cache. Depending on the model, we can achieve a very high cache hit rate after the first few simulation iterations, since only the attack outcomes are probabilistic. The performance implications are discussed further in Section 5.6.

5.2.2 Simulation with the Transition Matrix

Using the transition matrix within the simulation loop to determine state transitions ensures that the adversary decision must never be computed during any simulation run. However, there are drawbacks. In the case of extremely rare events, states resulting from rare events are explored in the state-space exploration and are therefore present in the transition matrix despite their low probability of appearing during simulation. Since the accuracy of the metrics is limited by the number of simulation runs, and the probabilistic nature of sampling

the attack step outcomes, the extra time to compute the adversary's decision for a rare state is wasted computation.

Reusing the transition matrix is the most efficient when simulation is not the only analysis method being performed. There are metrics that dictate the use of other solution methods. In the following section, we identify other numerical metrics that can be computed directly from the transition matrix.

5.3 Numerical Solutions

In this section, we explore the use of numerical solution methods after generating the transition matrix. These metrics are formed by direct manipulation of the transition matrix. We begin with infinite horizon metrics, highlighting both state-based and attack-specific metrics, then we present various types of transient solution methods.

5.3.1 State-Based Metrics

Several example metrics have been defined in [2, Chapter 4.5]. We show some metrics that can be computed using analytical techniques that work on the state space. We also define several new metrics, which provide fresh insight.

The first class of metrics considers the adversary who achieves a goal or a set of goals. These measures can be defined for states. Thus, the state space can be decomposed into three subsets. X_a is the set of states in which the adversary has already achieved his or her goal. X_m is the subset in which the adversary may achieve his or her goal in the future. I.e., there exists a path from each state in X_m to a state in X_a. Finally, X_n is the set of states from which the adversary will never achieve his or her goal. I.e., from a state in X_n, no state in X_a is reachable.

The three subsets can be computed with an extended algorithm for reachability analysis using the directed graph defined by matrix P. Initially X_a contains all states where the adversary already achieved his or her goal. All states $s \in X$ that are not connected to any $s' \in X_a$ are collected in X_n. Then all states $s \in X \setminus \{X_a \cup X_n\}$ without a connection to a

state $s'' \in X_n$ are added to X_a since from those states the adversary will eventually reach the goal. X_m consists of the remaining states that are neither in X_a nor in X_n.

If the initial state belongs to X_a or X_n, analysis is not necessary. In the former case, the probability of reaching the required goals is 1, and the latter case, it is 0. To compute the probability if s_0 belongs to X_m, we make states in the sets X_a and X_n absorbing by removing all outgoing transitions and setting the probability of remaining in the state to 1. If we reorder the states according to the sets, $\mathbf{P}$ looks as follows:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{m,m} & \mathbf{P}_{m,a} & \mathbf{P}_{m,n} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix}.$$

The success probability corresponds to the absorption probability of the process in X_a, which equals

$$p_{succ} = \mathbf{e}_1 \sum_{k=0}^{\infty} (\mathbf{P}_{m,m})^k \mathbf{P}_{m,a} \mathbb{1} = \mathbf{e}_1 (\mathbf{I} - \mathbf{P}_{m,m})^{-1} \mathbf{P}_{m,a} \mathbb{1}.$$

Since all states in X_m are connected to states in X_a, the above inverse matrix exists. Again, $\mathbf{e}_1(\mathbf{I} - \mathbf{P}_{m,m})^{-1}$ can be determined by solving a set of linear equations of order $|X_m|$.

Let $\mathbf{N}_{m,m} = (\mathbf{I} - \mathbf{P}_{m,m})^{-1}$. Then $\mathbf{N}_{m,m}(s, s')$ is the mean number of visits to state s' before a state from $X_a \cup X_n$ is reached, under the condition that the current state is $s \in X_m$. To integrate timing information, we define a row vector $\mathbf{t}$ with $\mathbf{t}(s) = E[T_{\beta(s)}(s)]$. Consequently, $\mathbf{M}(s, s') = \mathbf{N}_{m,m}(s, s') \cdot \mathbf{t}(s')$ is the mean time spent in state s' and $t^{abs} = \mathbf{e}_1 \mathbf{M} \mathbb{1}$ is the mean time to absorption. t^{abs} is the mean time before the goal is achieved or it becomes clear that it will not be achieved.

Time

To compute the time under the condition that the goal is eventually achieved, we have to consider the conditional absorbing Markov chain (see [13]). Let $\mathbf{p}_{m,a} = \mathbf{P}_{m,a} \mathbb{1}$ and define

$$\mathbf{u} = (\mathbf{I} - \mathbf{P}_{m,m})^{-1} \mathbf{p}_{m,a}.$$

Then, $\mathbf{u}$ contains no 0 elements, since we have assumed that from every state in X_m a state in X_a is reachable. Define $\mathbf{U} = diag(\mathbf{u})$, $\mathbf{P}^C_{m,m} = \mathbf{U}^{-1}\mathbf{P}_{m,m}\mathbf{U}$, and $\mathbf{p}^C_{m,a} = \mathbf{U}^{-1}\mathbf{p}^C_{m,a}$. Matrix $\mathbf{P}^C_{m,a}$ and vector $\mathbf{p}^C_{m,a}$ describe a new absorbing Markov chain with

$$\mathbf{N}^C_{m,m} = (\mathbf{I} - \mathbf{P}^C_{m,m})^{-1} = \mathbf{U}^{-1}\mathbf{N}_{m,m}\mathbf{U}$$

such that $\mathbf{N}^C_{m,m}(s, s')$ contains the main number of visits to state s' under the condition that the current state is s and absorption occurs in a state from X_a. Consequently, we can use $\mathbf{N}^C_{m,m}$ rather than $\mathbf{N}_{m,m}$ to compute the mean time to a successful attack by defining $\mathbf{M}^C(s, s') = \mathbf{N}^C_{m,m}(s, s') \cdot \mathbf{t}(s')$ and $t^{abs}_a = \mathbf{e}_1\mathbf{M}^C\,\mathbb{1}$ as the mean time of a successful attack under the condition that the attack is successful.

Cost

The computation of cost-related measures is similar to the computation of mean times. Thus, define

$$\mathbf{C}(s, s') = \mathbf{N}_{m,m}(s, s')C_{\beta(s)}(s).$$

Then $c^{abs} = \mathbf{e}_1\mathbf{C}\,\mathbb{1}$ is the average cost before absorption, i.e., before the goal states have been reached or become unreachable. The costs of a successful attack are $c^{abs}_a = \mathbf{e}_1\mathbf{C}^C\,\mathbb{1}$, where $\mathbf{C}^C$ is built like $\mathbf{C}$ using $\mathbf{N}^C_{m,m}$ rather than $\mathbf{N}_{m,m}$.

Gain

Computation of the gain is again similar; the only difference arises from the gain's association with transitions rather than states. Thus, define

$$\mathbf{g}_m(s) = \sum_{s' \in X_m} \left(\sum_{(i,o): a_i = \beta(s) \wedge s \xrightarrow{i,o} s'} Pr_i(s, o)G_i(s, o, s') \right).$$

Then

$$\sum_{s' \in X_m} \mathbf{N}_{m,m}(s_0, s')\mathbf{g}_m(s')$$

is the average gain due to attack steps that do not leave X_m. To compute the average gain until a state outside X_m is reached, the gain of the transition that leaves X_m must also be considered. The probability of leaving X_m from state $s \in X_m$ such that $s' \in X_a \cup X_n$ equals $\mathbf{N}_{m,m}(s_0, s)\mathbf{P}(s, s')$. That implies that we obtain g^{abs} for the average gain until a state in $X_n \cup X_a$ is reached.

$$g'_m(s') = \sum_{s'' \in X_a \cup X_n} \left(\sum_{(i,o): a_i = \beta(s') \wedge s' \xrightarrow{i,o} s''} Pr_i(s', o) G_i(s', o, s'') \right)$$

$$g^{abs} = \sum_{s' \in X_m} \mathbf{N}_{m,m}(s_0, s') \left(\mathbf{g}_m(s') + g'_m(s') \right)$$

Again, the conditional gain if a state from X_a is entered can be computed using $\mathbf{N}^C_{m,m}$ in the previous equations.

Detection

For the computation of the probability that an attack step is not detected, let $\mathbf{F}$ be an $m \times m$ matrix defined as follows:

$$\mathbf{F}(s, s') = \sum_{(i,o): a_i = \beta(s) \wedge s \xrightarrow{i,o} s'} Pr_i(s, o) F_i(s, o).$$

Matrix $\mathbf{F}$ includes the probabilities that an attack step is performed and not detected. $\mathbf{F}$ can be decomposed like $\mathbf{P}$, and since $\mathbf{F}(s, s') \leq \mathbf{P}(s, s')$, the inverse matrix $(\mathbf{I} - \mathbf{F}_{m,m})^{-1}$ exists and

$$p_{ndet} = \mathbf{e}_1 (\mathbf{I} - \mathbf{F}_{m,m})^{-1} \mathbf{P}_{m,a} \, \mathbb{1}$$

is the probability that the adversary reaches a state in X_a without being detected.

5.3.2 Attack-Specific Metrics

Another class of metrics is related to the occurrence of specific attack steps or outcomes of attack steps in a sequence of attacks. To determine these measures, we use the matrices $\mathbf{P}^{\{i\}}$ and $\mathbf{P}^{\neg\{i\}}$.

Matrix $\mathbf{P}$ does not distinguish between different attack steps. To introduce such a distinction, we define $\mathbf{P}^{\{i\}}$ as the matrix containing only transition probabilities according to $a_i \in A$.

$$\mathbf{P} = \sum_{a_i \in A} \mathbf{P}^{\{i\}}.$$

Define $\mathbf{P}^{\neg\{i\}} = \mathbf{P} - \mathbf{P}^{\{i\}}$ and $\mathbf{p}^{\{i\}} = \mathbf{P}^{\{i\}}\,\mathbb{1}$. Of course, $\mathbf{P}^{\{i\}}$ and $\mathbf{P}^{\neg\{i\}}$ are structured like $\mathbf{P}$, but some of the non-zero sub-matrices in $\mathbf{P}$ may be zero in $\mathbf{P}^{\{i\}}$ or $\mathbf{P}^{\neg\{i\}}$. The notation can be extended to sets of attack steps or pairs of attack steps and outcomes. E.g., $\mathbf{P}^{\{(i,o),j\}}$ captures all transitions according to attack step i with outcome o and attack step j with an arbitrary outcome.

We overload notation slightly; from here on, we use $\mathbf{P}^{\{i\}}$ for probabilities related to specific attack steps as well as attack step outcome pairs, where the index (i, o) would be more appropriate. If the state is irreducible and $\mathbf{P}^{\{i\}} \neq \mathbf{0}$, then the probability of performing attack step i is 1. Otherwise, the probability of performing attack step i in a transient state equals $pa_0^i = 1 - \mathbf{e}_1(\mathbf{I} - \mathbf{P}_{0,0}^{\neg\{i\}})^{-1} \sum_{l=1}^{L} \mathbf{P}_{0,l}^{\neg\{i\}}\,\mathbb{1}$, and the probability of performing attack step i in X_l $(l = 1, \ldots, L)$ under the condition that it has not been performed in a state from X_0 equals 0 if $\mathbf{P}_{l,l}^{\{i\}} = \mathbf{0}$ and equals $pa_0^i = \mathbf{e}_1(\mathbf{I} - \mathbf{P}_{0,0}^{\neg\{i\}})^{-1}\mathbf{P}_{0,l}^{\neg\{i\}}\,\mathbb{1}$ otherwise, such that the probability of performing attack step i is given by $pa^i = \sum_{l=0}^{L} pa_l^i$. The computations can be extended to sequences of attack steps and attack step outcome pairs. E.g., the probability of performing attack step i before j is given by

$$\mathbf{e}_1 \sum_{k=0}^{\infty} (\mathbf{P}^{\neg\{i,j\}})^k \mathbf{P}^{\{i\}} \sum_{k=0}^{\infty} (\mathbf{P}^{\neg\{j\}})^k \mathbf{P}^{\{j\}}\,\mathbb{1}.$$

If all states in matrices $\mathbf{P}^{\neg\{i,j\}}$ or $\mathbf{P}^{\neg\{j\}}$ are transient, then the infinite matrix sums can be replaced by the inverse matrices $(\mathbf{I} - \mathbf{P}^{\neg\{i,j\}})^{-1}$ and $(\mathbf{I} - \mathbf{P}^{\neg\{j\}})^{-1}$ if appropriate.

In stationary analysis, one must compute the stationary distribution inside each irreducible subset; it is then multiplied by the probability of reaching the subset from the initial state, i.e., by probability pe_l. The embedded stationary distribution of the states in subset l is given by the solution of

$$\mathbf{p}_l = \mathbf{p}_l \mathbf{P}_{l,l} \text{ and } \mathbf{p}_l \mathbb{1} = pe_l.$$

The embedded stationary distribution does not consider the different sojourn times due to different values for $T(s)$. The stationary vector can be achieved by normalizing the values of the embedded stationary distribution according to the sojourn times for the states in the irreducible subset. The components of the stationary vector π are computed as

$$\pi_l(s) = \frac{\mathbf{p}_l(s)\mathbf{t}(s)}{\sum_{s' \in X_l} \mathbf{p}_l(s')\mathbf{t}(s')}.$$

The frequency of attack step i is then given by

$$f_i = \sum_{l=1}^{L} \sum_{s \in X_l} \frac{\pi_l(s) \sum_{s' \in X_l} \mathbf{P}^{\{i\}}(s, s')}{T(s)}.$$

5.3.3 Types of Transient Analysis

The metrics computed up to now are all related to an infinite time horizon on which the time might be stopped when a specific condition is observed, i.e., a state has been reached or an attack has occurred. The situation is more complex if the system should be analyzed for a finite time horizon τ. In that case, transient analysis has to be performed. We briefly consider five different approaches, all of which have specific advantages and disadvantages.

First, if $\mathbf{P}$ can be reordered to an upper triangular matrix, then the system is completely acyclic. In that case, the behavior of the system is described by a finite set of paths, and a *path-based analysis* can be performed. Each path starts in the initial state s_0 and ends after finitely many steps in an absorbing state. For a path, the path probability and the probability of being in a specific state at time τ can be computed. However, computation of the latter requires the computation of convolutions of the distributions T_i along the path,

which is cumbersome for longer paths.

The second approach is to substitute attack step timing distributions by geometric distributions with the same mean. Define $0 < \Delta \leq \min_{s \in X}(T_{\beta(s)}(s)/(1 - \mathbf{P}(s,s)))$ as the basic time step of the new discrete-time Markov chain. Then, a new transition matrix $\overline{\mathbf{P}}$ is defined as

$$(1 - \overline{\mathbf{P}}(s,s)) = (1 - \mathbf{P}(s,s))\frac{\Delta}{T_{\beta(s)}(s)},$$

which implies $0 \leq \overline{\mathbf{P}}(s,s) < 1$. Then define

$$\overline{\mathbf{P}}(s,s') = \mathbf{P}(s,s')\frac{\Delta}{T_{\beta(s)}(s)} \text{ for } s \neq s',$$

which implies

$$\begin{aligned}\sum_{s' \in X} \overline{\mathbf{P}}(s,s') &= \sum_{s' \in X \setminus \{s\}} \overline{\mathbf{P}}(s,s') + \overline{\mathbf{P}}(s,s) \\ &= 1 - (1 - \mathbf{P}(s,s))\frac{\Delta}{T_{\beta(s)}(s)} + (1 - \mathbf{P}(s,s))\frac{\Delta}{T_{\beta(s)}(s)} = 1.\end{aligned}$$

DTMC analysis can be performed using matrix $\overline{\mathbf{P}}$, in which every step has a duration of Δ such that the distribution of the DTMC at discrete time points $k \cdot \Delta$ can be computed.

Similarly, one can use continuous-time Markov chains (CTMCs) for analysis. In that case, each sojourn time is replaced by an exponential distribution with rate $\mu_s = (E[T_{\beta(s)}(s)])^{-1}$. The resulting model is a CTMC on the same state space X, and standard *CTMC analysis* can be applied.

The approach may be extended through the representation of each sojourn time $T_{\beta(s)}(s)$ as a PH distribution. E.g., fitting the first two moments results in a PH distribution with 2 phases, if the coefficient of variation is not too small. In that case, the state space of the CTMC is enlarged, since the phase of the distribution has to be considered in addition to the state from X. If each sojourn time is replaced with a PH distribution with 2 phases, the size of the state space doubles. *Extended CTMC analysis* can then use standard techniques for the transient analysis of CTMCs.

The final analysis method is *simulation*, which can easily be performed using matrix $\mathbf{P}$

and $T_{\beta(s)}(s)$ for all $s \in X$. This method has been discussed in detail in Section 5.2.2. This method is more efficient than traditional simulation since the values for $\beta(s)$ are available, and do not need to be recomputed during.

5.4 Other Analyses

Other types of analysis may be possible if the attack execution graph conforms to certain assumptions. In this section, we highlight some of the analysis techniques used in the related work that are applicable to an attack execution graph. The solution methods presented up to this point are general and apply to any attack execution graph and adversary profile combination.

5.4.1 Exponential Time

If the AEG is exponential in time, then the mean effort to failure (METF) method discussed in [4] is applicable. Since calculating the METF requires analyzing the underlying Markov chain, the steps to explore the reachable state-space and generate the transition matrix must be completed.

Once the transition probability matrix is available, metrics such as mean up time, mean down time, mean cycle time, mean time to first failure, and mean time to failure are easily computable using the techniques in [14].

5.4.2 Attack Step Interchangeability

In exploring the related work, we recognized that the assumption of monotonicity was often made during analysis. The monotonic property is not invoked in order to minimize the size of the graph by eliminating loops or by minimizing the reachable state-space. Rather, it ensures that the order of the attack steps does not affect the analysis. So long as the ordering of attack steps is relevant, the analysis remains exponential in complexity.

In order for the two attack steps shown in Figure 5.1, a_1 and a_2, to be interchangeable,

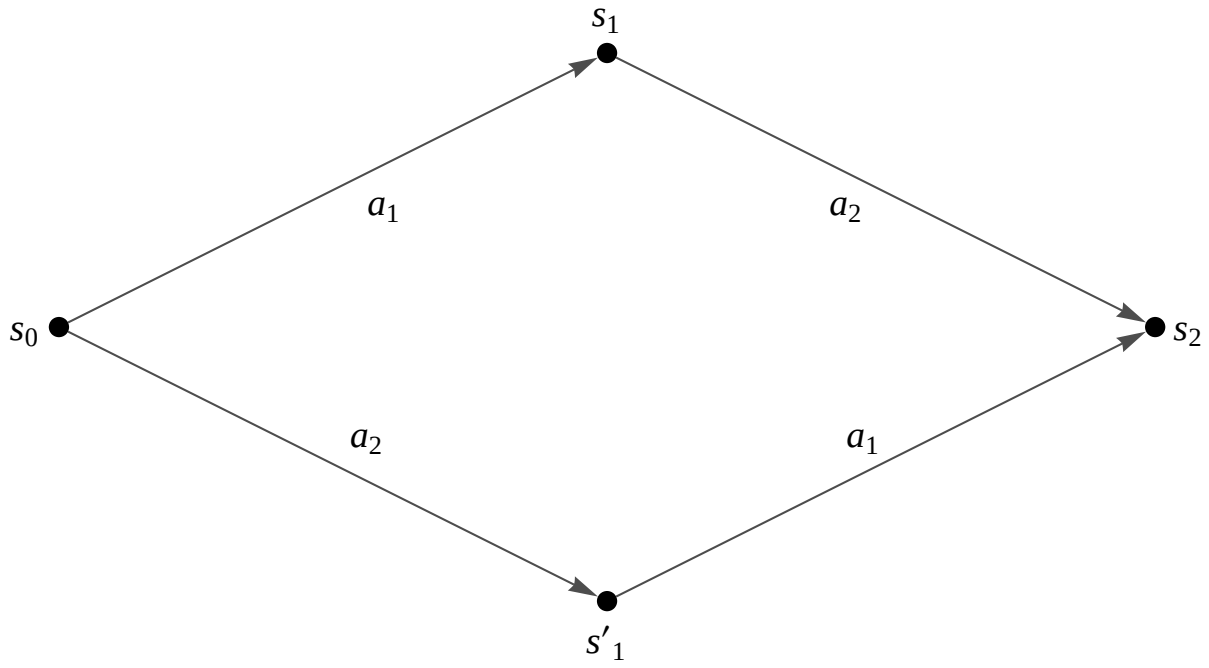


Figure 5.1: The Interchangeability of Attack Steps

they must conform to the following two requirements, which depend on the assumption that both attack steps are enabled from some starting state s_0. That is, $B_1(s_0) = True$ and $B_2(s_0) = True$. The following conditions must hold regardless of that initial state.

Then, each attack step must remain enabled regardless of the outcome of the other attack step.

$$\begin{aligned} B_2(E_1(s_0, o_1)) = True \quad \forall o_1 \in O_{a_1} \\ B_1(E_2(s_0, o_2)) = True \quad \forall o_2 \in O_{a_2} \end{aligned} \tag{5.1}$$

And finally, the resulting state must be the same regardless of the ordering of the attack steps for all possible outcome combinations.

$$\begin{aligned} E_2(E_1(s_0, o_1), o_2) = E_1(E_2(s_0, o_2), o_1) \\ \forall o_1 \in O_{a_1} \quad \forall o_2 \in O_{a_2} \end{aligned} \tag{5.2}$$

Equation 5.1 implies that the AEG must be static with respect to state transitions. If the AEG is not static w.r.t. transitions, then Eq. 5.1 does not hold for all initial states s_0.

If an AEG is static w.r.t. transitions and has monotonic enabling conditions and out-

comes, then both Equations 5.1 and 5.2 are satisfied. This means that once an attack step is enabled, regardless of the additions to the state, the attack step remains enabled. Additionally, the outcomes of attack steps only add to the state. Together, these imply that once an attack step is enabled, it is always enabled.

The same holds if simple outcomes are substituted for monotonic outcomes, and if simple enabling conditions are substituted for monotonic enabling conditions since the "simple" properties imply the corresponding monotonic properties.

Once the interchangeability of attack steps is ensured, we may apply the reachability algorithms from [8] and [15] to the AEG. While these algorithms only identify if a goal is reachable from the starting state, they may be augmented to use additional AEG information. For instance, one may analyze the cost to a goal. However, the aggregation function, in this case the sum of the costs, must also be proven to be independent of attack step ordering.

5.5 Example Metrics

In this section, we present example metrics for two models; a DMZ and a non-DMZ model. In order to validate our method, we compare these metrics to the metrics presented for the same models in [1]. We also present new metrics to provide further insight.

Each of these models represents a SCADA network. The two models demonstrate the effects of using a demilitarized zone (DMZ) between networks. One model allows direct communication between the corporate and control networks (non-DMZ), and the other separates the two with a DMZ. The non-DMZ model contains 19 state variables, while the DMZ model contains 20 state variables. Note that the adversaries we define for analysis typically have a look-ahead of 3 or 4 steps. The models and adversaries are fully documented in Appendix A.1.

First, we consider the time that the adversary takes to reach a goal. This is the metric that was reported in [1] using simulation. Table 5.1 contains the resulting metric mean and 95% confidence interval range from a simulation over 500 runs as well as the result of the numerical analysis. Note that the simulation results are imprecise when compared to the

results from numerical analysis. The precision and the short computation time are significant advantages of the numerical analysis. Performance comparisons are presented in Section 5.6.

Table 5.1: Time to Goal (in minutes)

	Simulation		Numerical	
Adversary	Non-DMZ	DMZ	Non-DMZ	DMZ
Nation State	116.3 ± 3.58	209.8 ± 5.2	114.3	209.3
Lone Hacker	116.3 ± 3.58	-	114.3	-
Terrorist Org	348.9 ± 13.8	353.1 ± 20.1	370.	370.
Employee	15.04 ± 0.48	-	15.3	-
Sys Admin	15.72 ± 0.76	15.2 ± 1.5	15.29	15.29

Next, we present two new metrics: the probability of detecting an attack, and the frequency of specific attacks within a sequence of attacks. The frequency of an attack step is explored in [2], but not in [1]. In order to enable calculation of the frequency of an attack step during simulation, a reward event must be fired each time an attack step is selected. If we had comparison values, we would likely see the same imprecision in the results from simulation as we did for the "time to goal" metric. It would be due simply to the limitations of simulation as a solution method.

Table 5.2: Frequency of Attack Steps

Adversary	Non-DMZ		DMZ	
Nation State	Hack User Acct.	87.5%	Hack User Acct.	57.3%
	Access Data	12.5%	Hack Data Server	35.8%
			Access Data	6.8%
Lone Hacker	Hack User Acct.	87.5%	Do Nothing	100%
	Access Data	12.5%		
Terrorist Org	Hack User Acct.	32.4%	Hack User Acct.	32.4%
	Elevate Privilege	13.5%	Elevate Privilege	13.5%
	Hack Ctrl. Server	54.0%	Hack Ctrl. Server	54.0%
Employee	User Login	6.5%	Do Nothing	100%
	Access Data	93.5%		
Sys Admin	Admin Login	6.5%	Admin Login	6.5%
	Access Data	93.5%	Access Data	93.5%

Table 5.2 reports the percentage of time spent attempting specific attack steps. Here, it is once again notable that adding the DMZ to the system deters both the Lone Hacker and the Employee adversaries from attempting any attack whatsoever. Also, the Nation

State adversary clearly changes its route to the goal. In addition to tracing the paths of adversaries through the system, the frequency of attack steps is also useful for determining the effects of deterrent mechanisms on the time an adversary spends on a given attack.

Table 5.3: Probability of Detection

Adversary	Non-DMZ	DMZ
Nation State	0.0784516	0.220143
Lone Hacker	0.0784516	-
Terrorist Org	0.478188	0.478188
Employee	0.0241125	-
Sys Admin	0.0241125	0.0635423

Finally, we present the probability of detection in Table 5.3. This metric is simple to compute using numerical analysis; to produce it via simulation, however, modification of the main simulation loop would be required. We note that the Terrorist Organization attacker is much more likely to be detected than any other attacker. Furthermore, by adding the demilitarized zone to the network, we triple the probability of detecting the Nation State and System Administrator adversaries.

5.6 Performance Comparison

While the types of numerical analysis we demonstrate can significantly reduce the analysis runtime when compared to the simulation approach used in [1] and [2], they require full exploration of the reachable state space. In this section, we present timing measurements to assess the scalability of the state-space generation algorithm and the attack step selection algorithm.

First, we present an example model that scales easily in both the state space and the reachable state space. The attack execution graph for this "gatekeeper model" is shown in Figure 5.2. The AEG consists of a single initial state variable that enables the entire row of attack steps. The successful outcome for each attack step captures the corresponding state variable in the row below. One final attack step is enabled by possessing any of the state variables in the state variable row, and a successful outcome results in achieving the goal.

The gatekeeper model is fully documented in Appendix A.2.

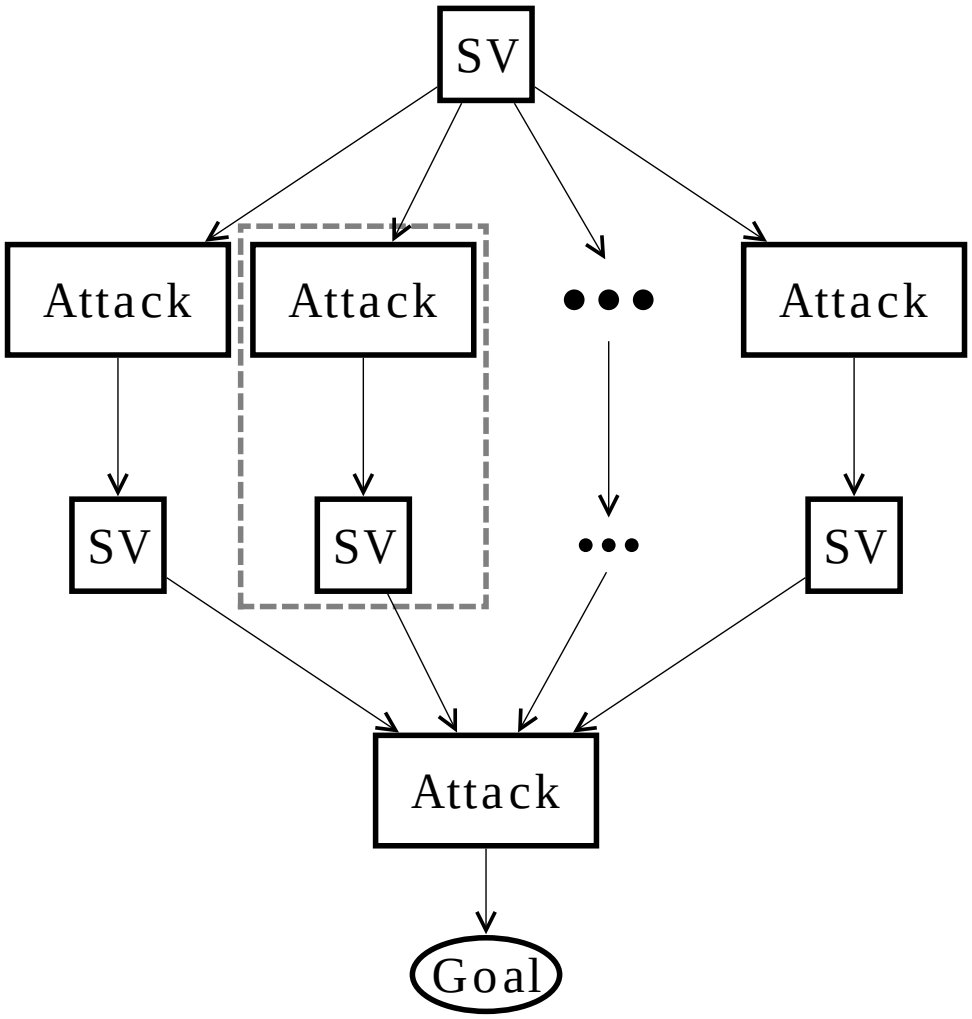


Figure 5.2: Gatekeeper Model

We took timing measurements while varying two factors in the model. One was the number of attack step and state variable node pairs in the center rows of the AEG. One node pair is circled in the figure for clarity. The other factor was the look-ahead of the adversary in the adversary profile. The size of the state-space is $2^{(N+2)}$, where N is the number of nodes in the AEG. More importantly, the complexity of the determine_beta function is on the order of $O((2N)^L)$, where L represents the look-ahead. For each configuration, we measured the overall time spent to generate the state space. The results are shown in Table 5.4.

The time to generate the state space followed the general exponential formula expressed

Table 5.4: State-Space Generation Timing (in seconds)

Look-ahead	5 Nodes	10 Nodes	20 Nodes	40 Nodes
3	0.002	0.036	0.513	15.16
4	0.022	0.577	16.128	1003
5	0.174	8.885	569.98	-
6	1.336	149.20	-	-
7	10.131	-	-	-

earlier in the paper. The time increased significantly as the look-ahead and the number of available attack steps increased. Some configurations took too long to compute within reasonable bounds.

However, those results were generated from a model in which the reachable state space was almost as large as the overall state space. In typical models, that would not be the case. There is an inherent ordering to attack steps that prevents the reachable state space from being anywhere near the size of the full state space. In order to demonstrate that, we repeated the timing experiments on the two models used in Section 5.5 and in [1].

Table 5.5: Simulation and Numerical Solution Timing (in seconds)

	Simulation		Numerical	
Adversary	Non-DMZ	DMZ	Non-DMZ	DMZ
Nation State	121.94	74.99	0.009	0.004
Lone Hacker	123.17	5.94	0.008	0.001
Terrorist Org	40.63	18.90	0.017	0.005
Employee	6.58	5.09	0.002	0.001
Sys Admin	75.10	55.65	0.021	0.010

Table 5.5 contains timing measurements for both simulation and state-space generation of the two typical models. The numerical solution techniques significantly outperform the simulations. That is primarily due to the optimization of computing determine_beta by using a graph-based approach.

In comparison to the state-space generation times of our Gatekeeper model, real-world models with the same number of state variables take considerably less time to complete. First, we note that the size of the reachable state space is very small compared to the size of the full state space. Also, although the DMZ model has one more state variable, its reachable state space is smaller than that of the non-DMZ model. It shows that the

structure of the model has a significant impact on the timing. And, most importantly, the state-space generation completes quickly for typical models.

Simulation was the only method available to solve ADVISE models. In this chapter, we improved the performance of the simulator by introducing a caching technique. We also introduced new solution methods that require state space and transition probability matrix generation. Those solution methods enable new quantitative security metrics, such as the mean cost to achieve a goal. The state space and transition probability matrix generation complete quickly, and solving for metrics using this approach is faster than the origional simulation method. ADVISE models can be quickly analyzed using a variety of techniques.

CHAPTER 6

CONCLUSION

We have extended the ADVISE method by generalizing the adversary decision algorithm, introducing new solution methods, and improving performance.

By generalizing the adversary decision algorithm, we remove the assumptions previously placed on system adversaries. More complex adversaries may be modeled since there are no restrictions placed on the adversary's utility functions. This enables adversary decisions based on thresholds such as budgets, time-dependent valuations, and any other arbitrary function.

The solution methods we introduce enable ADVISE to produce insightful metrics. Simulation is inherently a transient analysis. We have enabled the computation of both steady-state and transient metrics by generating the state-space and transition probability matrix. We also introduce alternative analyses that may be enabled if the model satisfies specific assumptions.

Finally, performance optimizations to the adversary decision algorithm and the simulation algorithm allow ADVISE to produce accurate metrics quickly and efficiently. We achieve this by adapting the adversary decision to use a graph-based rather than a tree-based algorithm. We also improve simulation by caching the adversary's decision over the course of the simulation.

These extensions to ADVISE allow extensive modeling of system security and adversary preferences, and enable the accelerated computation of relevant quantitative security metrics.

APPENDIX A

MODEL DOCUMENTATION

A.1 SCADA Network Models

There are two SCADA network models, the Non-DMZ model and the DMZ model. There are only slight differences between the models, and therefore they are documented together. Table A.1 documents the state-variables in both models. Any state-variable without additional marking is included in both models, while a state-variable with an asterisk(*) is unique to the Non-DMZ model and a state-variable with a dagger(†) is unique to the DMZ model. The adversary profile is documented in Table A.2. The AP is common to both models. Finally, the attack steps are presented in Tables A.3 and A.4. Table A.3 contains all of the attack steps which are common to both models, while Table A.4 documents the model-unique attack steps using the same labels as above.

A.2 Gatekeeper Model

The gatekeeper model depends on two variables: the look-ahead $\mathcal{N}$ and the number of attack step-access nodes $\mathcal{M}$. Table A.5 documents the gatekeeper model.

Table A.1: SCADA: Non-DMZ*and DMZ†: AEG Variables

Accesses	Goals	Knowledge	Skills
ApplicationServerAccess	CorruptData	VPNPassword	VPNHackSkill
PhysicalWorkstationAccess	RunAuthorizedPLCCode	HMIPassword	HackSkill
HMIUserAccess	RunUnauthorizedPLCCode	AdminWorkstationPassword	
AdminWorkstationAccess	RunUnauthorizedControlServerCode	UserWorkstationPassword	
InternetAccess	AccessData		
DataHistorianAccess			
HMIPhysicalAccess			
UserWorkstationAccess			
DataServerAccess†			

Table A.2: SCADA: Non-DMZ and DMZ: Adversary Profiles

(a) Adversary Skill Values

	Nation State	Hacker	Hostile Org.	Employee	Sys Admin
VPNHackSkill	900	900	900	200	600
HackSkill	900	900	900	200	600

(b) Adversary Goal Payoff

	Nation State	Hacker	Hostile Org.	Employee	Sys Admin
AccessData	400	400		400	400
RunAuthorizedPLCCode	100	100	100	100	100
RunUnauthorizedControlServerCode	500	500	500	500	500
CorruptData	200	200	200	200	200
RunUnauthorizedPLCCode	300	300	300	300	300
RunUnauthorizedControl ServerCode			1000		

(c) Adversary Weights and Initial Conditions

	Look-ahead	w_C	w_F	w_P	Initial Accesses	Initial Knowledge
Nation State	4	0.01	0.59	0.4	InternetAccess	-
Hacker	4	0.2	0.4	0.4	InternetAccess	-
Hostile Org.	4	0.05	0.15	0.8	InternetAccess	-
Employee	4	0.4	0.1	0.5	InternetAccess PhysicalWorkstationAccess	VPNPassword UserWorkstationPassword
Sys Admin	4	0.4	0.1	0.5	InternetAccess PhysicalWorkstationAccess	VPNPassword UserWorkstationPassword AdminWorkstationPassword HMIPassword

Table A.3: SCADA Attack Steps Part 1: Combined Non-DMZ and DMZ

Attack Step Name	Precondition	Cost	Time	Outcomes	Probability	Detection Prob.	Effects
AdminWorkstationLocalLogin	((!AdminWorkstationAccess) && PhysicalWorkstationAccess && AdminWorkstationPassword)	1	1	Success	1	0.05	AdminWorkstationAccess
ChangePLCInstructionsfromControlServer	((!RunAuthorizedPLCCode) && RunUnauthorizedControlServerCode)	20	20	Success Failure	.9 0.1	.5 0	RunAuthorizedPLCCode
CorruptDataHistorian	((!CorruptData) && (AdminWorkstationAccess || DataHistorianAccess))	20	20	Success Failure	.8 0.2	0.5 0	CorruptData
DoNothing		0	1	Outcome 1	1	0	
EscalateNetworkPrivilege	((!AdminWorkstationAccess) && UserWorkstationAccess && (HackSkill > 500))	30	30	Success Failure	0.6 0.4	0.25 0	AdminWorkstationAccess
ForwardUnauthorizedPLCCode	((!RunUnauthorizedPLCCode) && RunUnauthorizedControlServerCode)	45	90	Success Failure	0.5 0.5	.85 0	RunUnauthorizedPLCCode
HackControlServerfromAdmin	((!RunUnauthorizedControlServerCode) && (HackSkill > 500) && AdminWorkstationAccess)	60	150	Success Failure	0.75 0.25	0.05 0	RunUnauthorizedControlServerCode
HackControlServerfromHMI	((!RunUnauthorizedControlServerCode) && (HackSkill > 500) && HMIUserAccess)	60	90	Success Failure	.65 0.35	0.05 0	RunUnauthorizedControlServerCode
HackUserWorkstation	((!UserWorkstationAccess) && (HackSkill > 500) && (ApplicationServerAccess || InternetAccess))	60	60	Success Failure	0.5 0.5	0.05 0	UserWorkstationAccess
HMILogin	((!HMIUserAccess) && HMIPhysicalAccess && HMIPassword)	1	1	Success	1	0.01	HMIUserAccess
ObtainHMIPhysicalAccess	((!HMIPhysicalAccess) && PhysicalWorkstationAccess)	10	10	Success Failure	0.7 0.3	0 1	HMIPhysicalAccess
ReportFalseDataUpstream	((!CorruptData) && RunUnauthorizedControlServerCode)	20	20	Success Failure	0.85 0.15	0.1 0	CorruptData
UserWorkstationLocalLogin	((!UserWorkstationAccess) && UserWorkstationPassword && PhysicalWorkstationAccess)	1	1	Success	1	0.02	UserWorkstationAccess
ChangePLCInstructionsfromHMI	((!RunAuthorizedPLCCode) && HMIUserAccess)	20	20	Success Failure	0.95 0.05	0.5 0	RunAuthorizedPLCCode
UserWorkstationVPNLogin	((!UserWorkstationAccess) && InternetAccess && VPNPassword)	1	1	Success	1	0.01	UserWorkstationAccess
VPNHack	((!UserWorkstationAccess) && InternetAccess && (VPNHackSkill > 500))	30	30	Success Failure	0.3 0.7	0.02 0	UserWorkstationAccess

Table A.4: SCADA Attack Steps Part 2: Unique Non-DMZ*and DMZ†

Attack Step Name	Precondition	Cost	Time	Outcomes	Probability	Detection Prob.	Effects
AccessDataHistorian*	((!AccessData) && (UserWorkstationAccess || DataHistorianAccess || AdminWorkstationAccess || ApplicationServerAccess))	10	10	Success	.7	0.01	AccessData
				Failure	0.3	0	
AccessDataHistorian†	((!AccessData) && DataHistorianAccess)	10	10	Success	.95	0.01	AccessData
				Failure	0.05	0	
AccessDataServer†	((!AccessData) && (AdminWorkstationAccess || DataServerAccess || ApplicationServerAccess))	10	10	Success	0.7	0.01	AccessData
				Failure	0.3	0	
HackApplicationServer*	((!ApplicationServerAccess) && (HackSkill > 500) && (UserWorkstationAccess || InternetAccess || DataHistorianAccess))	60	90	Success	0.3	0.35	ApplicationServerAccess
				Failure	0.7	0	
HackApplicationServer†	((!ApplicationServerAccess) && (HackSkill > 500) && (UserWorkstationAccess || InternetAccess))	60	90	Success	0.3	0.35	ApplicationServerAccess
				Failure	0.7	0	
HackDataHistorian*	((!DataHistorianAccess) && (HackSkill > 500) && (InternetAccess || UserWorkstationAccess || ApplicationServerAccess))	60	60	Success	0.1	0.5	DataHistorianAccess
				Failure	0.9	0	
HackDataHistorian†	((!DataHistorianAccess) && (HackSkill > 500) && DataServerAccess)	60	60	Success	0.7	0.5	DataHistorianAccess
				Failure	0.3	0	
HackDataServer†	((!DataServerAccess) && (HackSkill > 500) && (ApplicationServerAccess || UserWorkstationAccess || AdminWorkstationAccess))	60	60	Success	.8	0.1	DataServerAccess
				Failure	0.2	0	

Table A.5: Gatekeeper Model

(a) AEG Variables

Accesses	Goals	Knowledge	Skills
InitialAccess	GateGoal	-	-
Access1			
⋮			
Access$\mathcal{M}$			

(b) Adversary Profile

Look-ahead	w_C	w_F	w_P	Initial Accesses	Goals	Value	Initial Knowledge	Skills	Value
$\mathcal{N}$	0	0	1	InitialAccess	GateGoal	1000	-	-	-

REFERENCES

[1] E. A. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehreke, "Model-based security metrics using adversary view security evaluation (advise)," in *Quantitative Evaluation of Systems (QEST)*. IEEE, 2011, pp. 191–200.

[2] E. A. LeMay, "Adversary-driven state-based system security evaluation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2011.

[3] M. Dacier and Y. Deswarte, "Privilege graph: an extension to the typed access matrix model," *Computer SecurityESORICS 94*, pp. 319–334, 1994.

[4] M. Dacier, Y. Deswarte, and M. Kaâniche, "Quantitative assessment of operational security: Models and tools," *LAAS Research Report*, vol. 96493, p. 5, 1996.

[5] R. Ortalo, Y. Deswarte, and M. Kaâniche, "Experimenting with quantitative evaluation tools for monitoring operational security," *Software Engineering, IEEE Transactions on*, vol. 25, no. 5, pp. 633–650, 1999.

[6] J. Peterson, *Petri Net Theory and the Modeling of Systems*. Pretence-Hall, Inc., Englewood Cliffs, NJ, 1981.

[7] J. Kemeny and J. Snell, *Finite Markov Chains*. Springer, 1960.

[8] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 121–130.

[9] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Proceedings of the 19th Annual Computer Security Applications Conference*. IEEE, 2003, pp. 86–95.

[10] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991.

[11] H. Mine and S. Osaki, *Markovian Decision Processes*. American Elsevier, 1970.

[12] R. E. Tarjan, "Depth-first search and linear graph-algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[13] P. Narain, "The conditional Markov chain in a genetic context," *Journal of Genetics*, vol. 63, no. 2, pp. 49–62, 1977.

[14] J. Buzacott, "Markov approach to finding failure times of repairable systems," *IEEE Transactions on Reliability*, vol. 19, no. 4, pp. 128–134, 1970.

[15] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. New York, NY: Springer, 2005, ch. 9.