Model-based Security Metrics using ADversary VIew Security Evaluation (ADVISE)

Elizabeth LeMay, Michael D. Ford, Ken Keefe, William H. Sanders
Information Trust Institute, Coordinated Science Laboratory,
and Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign
Urbana, IL, USA
Email: {evanrui2, mdford2, kjkeefe, whs}@illinois.edu

Carol Muehrcke
Cyber Defense Agency
Wisconsin Rapids, WI, USA
Email: cmuehrcke@cyberdefenseagency.com

***Abstract*—System architects need quantitative security metrics to make informed trade-off decisions involving system security. The security metrics need to provide insight on weak points in the system defense, considering characteristics of both the system and its adversaries. To provide such metrics, we formally define the ADversary VIew Security Evaluation (ADVISE) method. Our approach is to create an executable state-based security model of a system and an adversary that represents how the adversary is likely to attack the system and the results of such an attack. The attack decision function uses information about adversary attack preferences and possible attacks against the system to mimic how the adversary selects the most attractive next attack step. The adversary's decision involves looking ahead some number of attack steps. System architects can use ADVISE to compare the security strength of system architecture variants and analyze the threats posed by different adversaries. We demonstrate the feasibility and benefits of ADVISE using a case study. To produce quantitative model-based security metrics, we have implemented the ADVISE method in a tool that facilitates user input of system and adversary data and automatically generates executable models.**

***Keywords*- Quantitative Security Metrics, State-based Security Model, Adversary Attack Decisions**

I. Introduction

System architects need quantitative security metrics to make informed trade-off decisions involving system security. When comparing several system architecture variants, it is useful to know not only which architectures are more secure but how much more secure they are. Placing a server in a separate DMZ subnetwork is likely more secure than not using a DMZ, but how much more secure? Is the extra security worth the extra cost? Cost can be quantified in monetary units, but security quantification is less obvious.

To produce predictive, quantitative security metrics, we have developed the ADversary VIew Security Evaluation (ADVISE) method. Our approach is to create executable security models. As shown in Figure 1, an ADVISE executable model combines information about the system, the adversary, and the desired security metrics to produce quantitative metrics data.

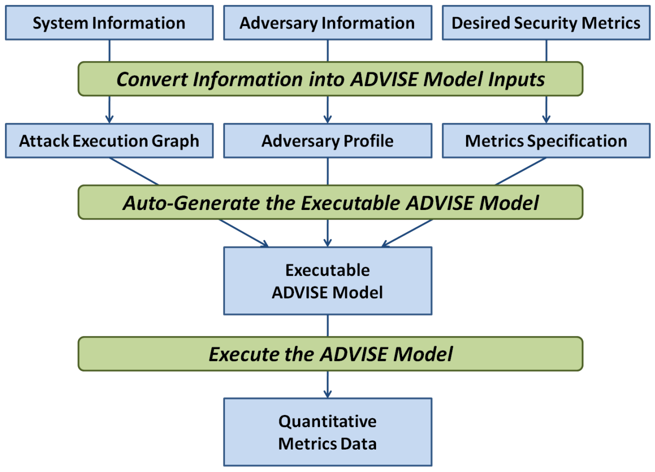

Figure 1. The ADversary VIew Security Evaluation (ADVISE) method produces quantitative security metrics data using an executable model.

Attacks against a system can be regarded as sequences of smaller attack steps. In an ADVISE model, these attack steps are precisely defined and organized into an *attack execution graph*. An attack execution graph differs from other attack graphs in that the attack execution graph contains timing, cost, probabilistic outcomes, and other information about each attack step. This extra information makes it possible to analyze ADVISE models using discrete-event simulation.

Not all adversaries are alike. A nation-state entity, an insider individual, and a criminal organization may have very different attack goals and may prefer different types of attack steps. In an ADVISE model, the adversary profile captures a particular adversary's attack preferences, attack goals, and attack skills.

The ADVISE model execution algorithm uses the adversary profile and the attack steps in the attack execution graph to mimic how the adversary is likely to attack the system. The adversary selects the best next attack step by evaluating the attractiveness of several attack steps, considering cost, payoff, and the probability of detection. The execution algorithm also stochastically determines the outcome of each attack step that is attempted.

The metrics specified for an ADVISE model determine what measurements are recorded from discrete-event simulation runs. Metrics can assess the probability of compromise within a particular time period and can also give insight on the speed of compromise and the most likely attack steps.

The most significant contribution of this work is the sophisticated attack decision function that rates the relative attractiveness of attack steps by incorporating the adversary's attack preferences and attack goals. We introduce the *state look-ahead tree* (SLAT) to recursively compute how future attack decisions influence the attractiveness values of the current attack step options. Other contributions of this work include the ADVISE execution algorithm and a case study demonstrating the feasibility and benefits of ADVISE.

This paper contains both formal definitions and an execution algorithm for ADVISE and a case study demonstrating the use of ADVISE. Section II defines the ADVISE model formalism, including the attack execution graph and the adversary profile. Section III defines the ADVISE attack decision function. The ADVISE metrics specification format is described in Section IV, and the ADVISE execution algorithm is described in Section V. In Section VI, we present a case study of a SCADA system. Related work is discussed in Section VII, and the paper concludes in Section VIII.

II. ADVISE Model Formalism

Performing a system security analysis using the ADVISE method involves specifying an attack execution graph (AEG) to represent potential attack steps against the system and specifying an adversary profile. The ADVISE model formalism collects and organizes specific information about possible attacks and adversaries. This information comes from system experts and adversary experts. This information is then used to automatically generate an executable model that represents how the adversary is likely to attack the system.

A. Attack Execution Graph Definition

A security analyst builds an AEG by thinking about attacks in terms of many small attack steps. Each attack step achieves some attack goal or makes progress toward an attack goal by changing the adversary's access to or knowledge of the system. For example, one attack step could be to obtain a user password (gaining knowledge); another attack step could be to log in to an internal network (gaining access); still other attack steps could allow an adversary to crash a server or read proprietary information (achieving attack goals).

An *attack execution graph* is defined by the tuple

$$\langle A, R, K, S, G \rangle, \tag{1}$$

where A is the set of *attack steps* against the system, R is the set of *access domains* in the system, K is the set of *knowledge items* relevant to attacking the system, S is the set of adversary *attack skills* relevant to attacking the system, and G is the set of adversary *attack goals* relevant to the system.

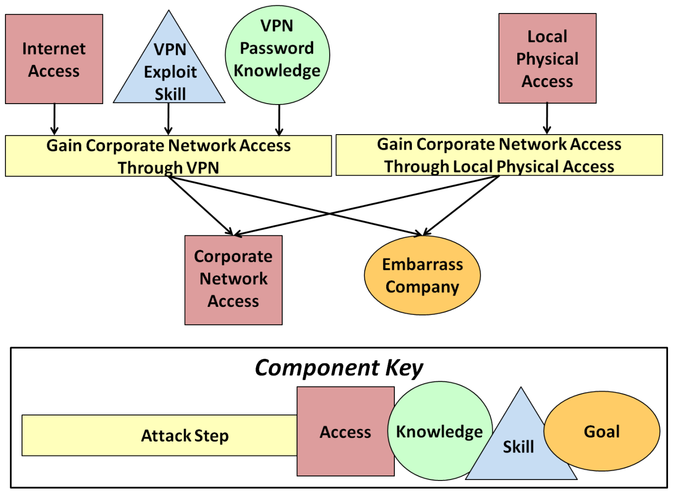

Figure 2. An Attack Execution Graph (AEG) represents possible attacks against a system that an adversary can use to reach his or her attack goals.

In a pictorial representation of an AEG (see Figure 2), the attack steps are rectangular boxes, the access domains are squares, the knowledge items are circles, the attack skills are triangles, and the attack goals are ovals. In this simple AEG, the attack goal is to embarrass the company by gaining corporate network access. There are two attack approaches represented by the two attack steps: "Gain Corporate Network Access Through VPN" and "Gain Corporate Network Access Through Local Physical Access." The objects with arrows pointing towards an attack step are the access, skills, knowledge, or goals that are relevant to determining if the adversary can attempt the attack step. For example, the adversary must first possess "Local Physical Access" before attempting the latter attack step. The objects with arrows pointing away from the attack step are the access, knowledge, or goals that can be affected when an adversary attempts the attack step. For example, either attack step in Figure 2 can enable the adversary to obtain "Corporate Network Access" and achieve the "Embarrass Company" goal.

In more complex AEGs, attack steps are joined through the access, knowledge, and goal objects to form a graph-like structure, as shown in the case study AEG (Figure 6).

B. Attack Step Definition

Each *attack step* $a_i \in A$ in the AEG is defined in such a way that the AEG can be converted later into an executable model. Formally, an attack step a_i is a tuple:

$$a_i = \langle B_i, T_i, C_i, O_i, Pr_i, D_i, E_i \rangle. \tag{2}$$

The components of an attack step are defined as functions of model state $s \in X$, where X is the set of all model states (to be defined later).

$B_i : X \to \{True, False\}$ is a Boolean precondition. The precondition determines if the adversary possesses what is required to attempt the attack step (specific access, knowledge, and/or attack skills) but does not already possess what can be gained by successfully completing this attack step (specific access, knowledge, and/or attack goals). For example, to attempt the attack step "Gain Corporate Network Access Through VPN" in Figure 2, the precondition requires that the adversary have Internet access and either the skill to exploit the VPN or knowledge of a VPN account password. The precondition also specifies that the adversary will not attempt this attack step if he or she already possesses "Corporate Network Access."

$T_i : X \times \mathbf{R}^+ \to [0, 1]$ is the length of time required to attempt the attack step. $T_i(s)$ is a random variable defined by a probability distribution function over the positive real numbers.

$C_i : X \to \mathbf{R}^{\geq 0}$ is the cost of attempting the attack step (regardless of the outcome).

O_i is the finite set of outcomes. For many attack steps, the set of outcomes contains two elements: success and failure.

$Pr_i : X \times O_i \to [0, 1]$ is the probability of outcome $o \in O_i$ occurring after the attack step is attempted, where $\sum_{o \in O_i} Pr_i(s, o) = 1$ for all s. System defenses and countermeasures can affect the outcome probabilities.

$D_i : X \times O_i \to [0, 1]$ is the probability of the attack being detected when outcome $o \in O_i$ occurs.

$E_i : X \times O_i \to X$ is the next-state that results when outcome $o \in O_i$ occurs.

Every AEG contains a "do-nothing" attack step, a_{DN}. The do-nothing attack step represents the option of an adversary to refrain from attempting any active attack against the system. The precondition B_{DN} is always true. The time, T_{DN}, is the period of time that elapses before the adversary reconsiders the decision to do nothing, which depends on the particular adversary. For most AEGs, the cost C_{DN} is zero, the detection probability D_{DN} is zero, the next-state is the same as the current state ($E_{DN}(s, o) = s, \forall s \in X$), there is only one outcome $o \in O_{DN}$, and the probability of that outcome, $Pr_{DN}(s, o)$, is one. The existence of the do-nothing attack step means that, regardless of the model state, there is always at least one attack step in the AEG whose precondition is satisfied.

C. Model State Definition

The *model state*, $s \in X$, reflects the progress of the adversary in attacking the system and is defined by

$$s = \langle R_s, K_s, G_s \rangle, \tag{3}$$

where $R_s \subseteq R$ is the set of access domains that the adversary can access, $K_s \subseteq K$ is the set of knowledge items that the adversary possesses, and $G_s \subseteq G$ is the set of attack goals the adversary has achieved. X is the set of all reachable model states.

D. Adversary Profile Definition

The *adversary profile* is defined by the tuple

$$\langle s_0, L, V, w_C, w_P, w_D, U_C, U_P, U_D, N \rangle, \tag{4}$$

where $s_0 \in X$ is the initial model state; L is the attack skill level function; V is the attack goal value function; w_C, w_P, and w_D are the attack preference weights for cost, payoff, and detection probability, respectively; U_C, U_P, and U_D are the utility functions for cost, payoff, and detection probability, respectively; and N is the planning horizon (to be described in Section III-B).

The *initial model state*, s_0, describes the starting point of the adversary's attack. An insider adversary will likely start with more access and knowledge than an outsider.

The *attack skill level function*, $L : S \to [0, 1]$, describes the attack proficiency of the adversary by mapping each attack skill in the AEG to a value in the range $[0, 1]$. A greater skill level is represented by a larger value. To provide a reference for assigning consistent skill level ratings, an analyst-defined rubric for each attack skill describes the attack skill level for a few key values.

The *attack goal value function*, $V : G \to \mathbf{R}^{\geq 0}$, describes the monetary-equivalent value of each attack goal in the AEG from the viewpoint of the adversary. Each attack goal is assigned a nonnegative real number. A more valuable goal is assigned a larger value. The *payoff value*, $P(s)$, of a model state s is a function of the values of all goals achieved in that model state: $P(s) = f(V(g))$.

The *attack preference weights* describe the relative attractiveness of unit changes in each of the three core criteria that adversaries consider when deciding among their attack options. The weight w_C is the relative attractiveness of decreasing the cost to the adversary in attempting the attack step. The weight w_P is the relative attractiveness of increasing the payoff to the adversary for successfully executing the attack step. The weight w_D is the relative attractiveness of decreasing the probability of being detected by the system during or after attempting the attack step. All three attack preference weights are between zero and one ($w_C, w_P, w_D \in [0, 1]$).

The *utility functions* (U_C, U_P, and U_D) map the native values of each attractiveness criterion to a $[0, 1]$ utility scale, where higher utility values represent more desirable values. The *cost utility function*, $U_C : \mathbf{R}^{\geq 0} \to [0, 1]$, maps the monetary value of the attack step cost to its utility according to the adversary. Because lower costs are more desirable (have a higher utility), the cost utility function exhibits a negative correlation. The *payoff utility function*, $U_P : \mathbf{R}^{\geq 0} \to [0, 1]$, maps the monetary value of the attack step payoff to its utility according to the adversary. The *detection utility function*, $U_D : [0, 1] \to [0, 1]$, maps the probability of attack step detection to its utility according to the adversary. Because lower detection probabilities are more desirable (have a higher utility), the detection utility function exhibits a negative correlation.

III. ADVISE Model Execution Functions

The ADVISE model formalism enables the creation of executable models. An adversary profile is coupled with an attack execution graph to produce an executable model that represents how the adversary is likely to attack the system.

The executable model includes the initial model state and the functions that govern state transitions. When the model is analyzed using discrete event simulation, the execution algorithm determines the sequence of state transitions that occur during a simulation run. Taking into account the current model state and the adversary's attack preferences, the attack decision function mimics how an adversary chooses the next attack step. The outcome of that attempt (whether the attack step succeeds or fails) determines the next state of the model. This process repeats with the attack decision function choosing the next attack step.

The ADVISE execution functions consist of the attack decision function and the attack step outcomes. The decision function produces a deterministic attack decision. Given the same adversary attack preferences (defined in the adversary profile), the same attack execution graph, and the same current model state, the decision function will always select the same next attack step. In contrast, the attack step outcomes are stochastic. In the attack execution graph, an attack step definition includes the probability of each outcome given that the step is attempted. An attack step outcome is pseudo-randomly generated using these probability distributions. The attack step outcomes determine the sequence of state transitions.

Attack step selection is based on evaluating the attractiveness of all the available attack steps from the viewpoint of the adversary and choosing a most attractive attack step. The decision is based on the current model state s.

First, the decision function checks the precondition, $B_i(s)$, of each attack step, a_i, in the attack execution graph. Recall that the precondition specifies the access, knowledge, and skill that the adversary must possess to attempt the attack step. Attack steps whose preconditions are satisfied based on the current model state s comprise the set of *available attack steps* A_s:

$$A_s = \{a_i \in A | (B_i(s) = \mathit{True})\}, \tag{5}$$

where A is the set of all attack steps.

Next, the decision function evaluates the attractiveness of each available attack step using three decision criteria: the cost of attempting the attack, the expected probability of detection, and the expected payoff in the state reached after the attack. The adversary profile contains attack preference weights that reflect the relative importance of incremental changes in these three decision dimensions.

We begin by explaining the attack decision function for a short-sighted adversary. This simplified adversary serves to introduce concepts we will extend in our explanation of the long-range-planning adversary.

A. The Short-Sighted Adversary

In a short-sighted attack, the adversary only considers the immediate next attack steps and the immediate next states that could result from those next steps. The attractiveness, $attr(a_i, s)$, of available attack step $a_i \in A_s$ based on current model state s is a linear combination of the adversary attack preference weights with the data about the attack step:

$$attr(a_i, s) = w_C \cdot C_i(s) + w_P \cdot P_i(s) + w_D \cdot D_i(s), \tag{6}$$

where $C_i(s)$ is the cost of attempting attack step a_i from state s, w_C is the adversary preference weight for cost, $P_i(s)$ is the expected payoff after attempting attack step a_i from state s, w_P is the adversary preference weight for payoff, $D_i(s)$ is the expected probability of detection associated with attempting attack step a_i from state s, and w_D is the adversary preference weight for detection.

$P_i(s)$, the expected payoff after attempting attack step a_i from state s, is computed as the sum of the payoff in each possible next-state weighted by the probability of the attack step outcome leading to that next-state:

$$P_i(s) = \sum_{o \in O_i} (P(E_i(s,o)) \cdot Pr_i(s,o)), \tag{7}$$

where $P(r)$ is the payoff in state r (as defined in Section II-D). $D_i(s)$, the expected detection probability after attempting attack step a_i from state s, is computed similarly:

$$D_i(s) = \sum_{o \in O_i} (D_i(s,o) \cdot Pr_i(s,o)). \tag{8}$$

Then, for current model state s, the attack decision function selects a best next attack step, $\beta(s)$, as an available attack step with the maximal attractiveness value:

$$\beta(s) \in \{a^* \in A_s | attr(a^*, s) = \max_{a_i \in A_s} attr(a_i, s)\} \tag{9}$$

If multiple attack steps have the same maximal attractiveness value, then the attack decision function uniformly selects one attack step from the set of maximally attractive attack steps.

B. The Long-Range-Planning Adversary

Equation (9) describes an attack decision based on information about the immediate next attack step options and the resulting immediate next states; however, most real adversaries do not attempt multi-step attacks by plunging blindly ahead looking only one step into the future. Modeling a more sophisticated adversary decision requires a long-range-planning attack decision function. The adversary considers what goals can be achieved using a sequence of attack steps. We modify Equations (6)–(9) to model the goal-driven attack step selection of a long-range-planning adversary.

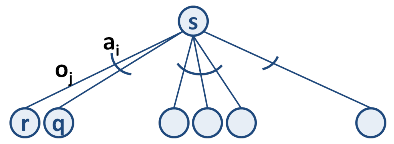

Figure 3. A State Look-Ahead Tree (SLAT) explores all possible outcomes of all possible attack steps from the root node (state s) to determine all possible next-states. Each next-state becomes a child node in the SLAT.

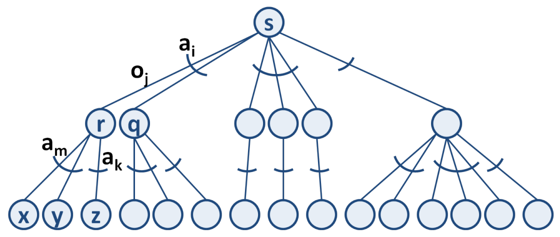

Figure 4. Construction of the SLAT continues by exploring the available attack steps in the leaf node states of the SLAT and adding next-state child nodes until the leaf nodes are N attack steps distant from the root node. Here, $N = 2$.

The planning horizon, N, is the number of steps into the future the adversary can consider when making an attack decision. The planning horizon is analogous to the number of moves a chess player can think ahead when planning his next move. The chess player must consider what he can directly control (his own future moves), as well as what he cannot control (the possible future moves of his opponent). Similarly, the adversary in our model must consider both what he can control (his own attack step decisions), as well as what he cannot control (the outcome of an attack attempt).

To analyze all possible sequences of attack steps of length N, we introduce the State Look-Ahead Tree (SLAT). The root node of the SLAT is the current state (labeled as state s in Figure 3). The child nodes are the possible next states of the parent state. Each available attack step in the parent state produces one or more child nodes. Each possible outcome of an attack step produces one child node. All child nodes produced by the same attack step are grouped together by a hash mark. In Figure 3, attack step a_i is an available attack step in state s, and outcome o_j of attack step a_i results in the model state transitioning to state r. Construction of the SLAT continues by exploring the available attack steps in the leaf node states of the SLAT and adding next-state child nodes that become the new leaf nodes. This tree-building process continues until the leaf nodes are N attack steps distant from the root node. Because the "do-nothing" attack step is an available attack step in any state, we can always build an tree with leaf nodes N attack steps distant from the root node. In the example SLAT in Figure 4, the planning horizon, N, is two.

After the top-down construction of the SLAT, the bottom-up best next step analysis begins. To evaluate the attractiveness of attack step a_i with a planning horizon of two, the decision function needs to consider not only the cost, detection probability, and payoff of a_i itself, but also the cost, detection probability, and payoff of the attack step that will be performed after a_i. Assume that attempting attack step a_i could transition the model state to state r or state q. Considering state r, the decision function needs to evaluate the attractiveness of attack steps a_m and a_k to determine the best attack step in state r. For illustration, let the best attack step from state r be a_m. The cost, detection probability, and payoff of a_m are reported back to be used in the attractiveness calculation for attack step a_i. The other available attack steps (here, a_k) are pruned from the tree. In this way, attack decisions made at the bottom of the SLAT determine which cost, detection probability, and payoff values are factored into the attack decisions higher in the SLAT. A pruned SLAT is shown in Figure 5. For each state in the pruned SLAT, there is only one attack step group left; this attack step is the best next attack step at that state considering all attack step data between that state and the planning horizon states.

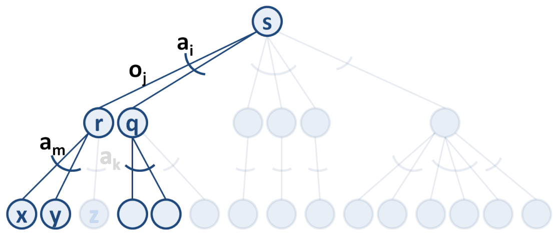

Figure 5. After the top-down construction of the SLAT, the bottom-up best next-step analysis prunes off branches with non-maximal attractiveness. For each state in the pruned SLAT, there is only one attack step group left; this attack step is the best next attack step at that state, considering all attack step data between that state and the planning horizon states.

More formally, the best next attack step, $\beta^N(s)$, is uniformly selected from among the available attack steps in model state s with the maximal attractiveness value, where the attractiveness value is computed recursively using a planning horizon of N:

$$\beta^N(s) \in \{a^* \in A_s | attr^N(a^*, s) = \max_{a_i \in A_s} attr^N(a_i, s)\}. \tag{10}$$

The attractiveness of an available attack step a_i in state s is computed recursively in a way that evaluates all possible sequences of attack steps of length N that begin with attack step a_i:

$$attr^N(a_i, s) = w_C \cdot C_i^N(s) + w_P \cdot P_i^N(s) + w_D \cdot D_i^N(s), \tag{11}$$

where $C_i^N(s)$ is the recursively-computed *expected path cost*, w_C is the adversary preference weight for cost, $P_i^N(s)$ is the recursively-computed *expected horizon payoff*, w_P is the adversary preference weight for payoff, $D_i^N(s)$ is the recursively-computed *expected path detection*, and w_D is the adversary preference weight for detection.

The expected horizon payoff is based on the states that can be reached at the edge of the planning horizon; the expected path cost and expected path detection are based on the attack steps between the current state and the states at the edge of the planning horizon.

The expected path cost, $C_i^N(s)$, is the expected sum of attack step costs starting from state s with attack step a_i and continuing a total of N steps into the future:

$$C_i^N(s) = \begin{cases} C_i(s), & \text{when } N = 1 \\ C_i(s) + \sum_{o \in O_i} (C_*^{N-1}(r) \cdot Pr_i(s,o), & \\ & \text{when } N > 1, \end{cases} \tag{12}$$

where state $r = E_i(s,o)$ is the next state after starting in state s, attempting attack step a_i, and obtaining outcome o. This definition of state r is also used in Equations (13)–(17).

This recursive definition for expected path cost requires determining the best next attack step from state r with planning horizon $(N-1)$:

$$C_*^{N-1}(r) \equiv C_k^{N-1}(r) \text{ when } \beta^{N-1}(r) = a_k. \tag{13}$$

The expected horizon payoff, $P_i^N(s)$, is the expected payoff in the state reached after attempting attack step a_i and continuing a total of N steps into the future:

$$P_i^N(s) = \begin{cases} \sum_{o \in O_i} (P(E_i(s,o)) \cdot Pr_i(s,o)), \\ \qquad \text{when } N = 1 \\ \sum_{o \in O_i} (P_*^{N-1}(r) \cdot Pr_i(s,o)), \\ \qquad \text{when } N > 1. \end{cases} \tag{14}$$

Just as for the expected path cost, the computation of the expected horizon payoff requires determining the best next attack step from state r with planning horizon $(N-1)$:

$$P_*^{N-1}(r) \equiv P_k^{N-1}(r) \text{ when } \beta^{N-1}(r) = a_k. \tag{15}$$

The expected path detection, $D_i^N(s)$, is the expected probability of detection at any point during attack step a_i or the other attack steps continuing a total of N steps into the future.

$$D_i^N(s) = \begin{cases} \sum_{o \in O_i} (D_i(s,o) \cdot Pr_i(s,o)), \\ \qquad \text{when } N = 1 \\ \sum_{o \in O_i} ((1 - (1 - D_i(s,o)) \\ \cdot (1 - D_*^{N-1}(r))) \cdot Pr_i(s,o)), \\ \qquad \text{when } N > 1. \end{cases} \tag{16}$$

Just as for the expected path cost and expected horizon payoff, the computation of the expected path detection requires determining the best next attack step from state r with planning horizon $(N-1)$:

$$D_*^{N-1}(r) \equiv D_k^{N-1}(r) \text{ when } \beta^{N-1}(r) = a_k. \tag{17}$$

Note that when the planning horizon N is one, Equation (11) is equivalent to Equation (6).

As N grows, the recursive attractiveness computation will be slow due to an exponential growth in the number of states to explore while constructing the SLAT. However, for small values of N, the computation is tractable.

For ease of explanation, we have not explicitly used the adversary's utility functions (U_C, U_P, and U_D) in Equations (6) or (11). Recall that these functions map the native values of each attractiveness criterion to a $[0,1]$ utility scale, where higher utility values represent more desirable values. To incorporate the utility functions into Equation (11), replace $C_i^N(s)$ with $U_C(C_i^N(s))$, replace $P_i^N(s)$ with $U_P(P_i^N(s))$, and replace $D_i^N(s)$ with $U_D(D_i^N(s))$. The utility functions convert cost, payoff, and detection probability into an common unit of "utility" so that the weighted average computation (in the attractiveness function) is mathematically valid.

An attractiveness computation that incorporates utility functions can better reflect real-world decisions by acknowledging the often nonlinear nature of utility. For example, to a resource-constrained adversary, all attack costs above some monetary value may be of equally low utility value. For this adversary, there may be no utility difference between a cost of \$2.01 million and \$2.05 million, even if there is a difference in utility between a cost of \$10,000 and \$50,000; the cost utility function expresses this nonlinearity by explicitly mapping the monetary value of attack step cost to its utility value according to the adversary.

IV. ADVISE Metrics Specification

The purpose of building ADVISE models is to generate quantitative metrics that provide insight on the security strength of a particular system against a particular adversary. Metrics can be generated by running discrete-event simulations of the adversary attacking the system.

In general, metrics generated by discrete-event simulation can take many forms. To facilitate rapid model analysis, we specify a particular format for ADVISE metrics.

There are two types of ADVISE metrics: state metrics and event metrics. State metrics analyze the model state. Event metrics analyze events, namely state changes, attack step attempts, and attack step outcomes. For each metric, there are three required parts: time, metric type, and the state or event indicator function. Conditional reporting, which is optional, requires a condition expression.

A. State Metrics

State metrics take the form

$$\langle \tau, \lambda, \sigma \rangle, \tag{18}$$

where τ is the end time such that the metric reports on the time period $[0, \tau]$, $\lambda \in \{EndProb, AvgTime\}$ specifies the type of state metric, and σ is the state indicator function. *EndProb* is the ending state occupancy probability metric, i.e., the probability of the model state s at time τ being a state of interest ($\sigma(s) =$ True). *AvgTime* is the average time metric, i.e., the average amount of time during the time interval $[0, \tau]$ spent in a model state s such that $\sigma(s) =$ True.

The state indicator function, σ, returns a True value for model states of interest:

$$\sigma : X \to \{True, False\}, \tag{19}$$

where X is the set of all model states. Recall that the model state $s \in X$ is defined by the tuple $\langle R_s, K_s, G_s \rangle$. The state indicator function separates states of interest from those not of interest. For example, a state metric could measure the probability that the model is in a state s at time $\tau = 200$ in which goal $g_1 \in G$ has been achieved (i.e., $g_1 \in G_s$).

B. Event Metrics

Event metrics take the form

$$\langle \tau, \delta, \epsilon \rangle, \tag{20}$$

where τ is the end time such that the metric reports on the time period $[0, \tau]$, $\delta \in \{\mathit{Freq, ProbOcc}\}$ specifies the type of event metric, and ϵ is the event set. *Freq* is the frequency metric, i.e., the number of occurrences of events in ϵ during the time interval $[0, \tau]$. *ProbOcc* is the probability of occurrence metric, i.e., the probability that the events in ϵ occur at least once during the time interval $[0, \tau]$.

The event set ϵ is specified as the union of eight sets:

$$\epsilon = A_\epsilon \cup O_\epsilon \cup R_+ \cup R_- \cup K_+ \cup K_- \cup G_+ \cup G_-, \tag{21}$$

where $A_\epsilon \subseteq A$ is the set of attack steps whose execution constitutes an event; $O_\epsilon \subseteq \{\bigcup_{a_i \in A} O_i\}$ is the set of attack step outcomes whose execution constitutes an event; R_+, K_+, and G_+ are the sets of access domains, knowledge items, and attack goals, respectively, whose addition to the model state $\langle R_s, K_s, G_s \rangle$ constitutes an event; and R_-, K_-, and G_- are the sets of access domains, knowledge items, and attack goals, respectively, whose removal from the model state $\langle R_s, K_s, G_s \rangle$ constitutes an event. For simple metrics, all but one of these sets may be empty. For example, an event metric could measure the frequency with which the adversary attempts attack step a_i in the interval $[0, \tau]$. In this example, $\epsilon = \{a_i\}$ because a_i is the only event of interest.

C. Conditional Reporting

Conditional reporting enables analysts to study correlations between different metrics. Typically, metrics data are generated from many simulation runs and averaged across all runs. With conditional reporting, only the simulation runs that meet a specified condition are included in the average. The conditional expression includes a state or event metric, an equality or inequality relation, and a value. For example, one conditional expression could be that the frequency of attempts of attack step a_i is greater than four during time interval $[0, \tau]$. When a conditional expression is linked with a second state or event metric, correlations can be measured directly. One example of a metric with conditional reporting is the following: if, for this simulation run, the frequency of attack attempts for attack step a_i is greater than four, then use the data from this run to compute the ending state occupancy probability of a model state in which attack goal g_3 is achieved. This metric enables analysts to examine the correlation between frequent attempts of attack step a_i and the achievement of attack goal g_3.

V. ADVISE Model Execution Algorithm

When an ADVISE model is solved using discrete-event simulation, the initial model state is s_0 from the adversary profile, and the state transitions are governed by the attack decision function $\beta^N(s)$, the outcome probability distributions Pr_i, and the next-state functions E_i. The attack decision function chooses one attack step; the outcome probability distributions are used to randomly select one outcome of that attack step; and the next-state function of that attack step outcome determines the next state. The model state changes to that next state, and then the state transition process repeats.

During initialization, the simulation time is set to zero. Each attack step attempt advances the simulation clock by an amount of time determined by a random sample from the attack step execution time distribution defined by T_i. The simulation ends when the simulation clock reaches a specified simulation end time τ.

The ADVISE model execution algorithm is shown in Algorithm 1. The outcome probability distribution function $Prob_i$(State) refers to a discrete probability distribution, in which each $o \in O_i$ has probability Pr_i(State, o), as defined in the attack step definition.

Algorithm 1 ADVISE Model Execution

```
1: Time ⇐ 0
2: State ⇐ $s_0$
3: while Time < $\tau$ do
4:     Attack$_i$ ← $\beta^N$(State)
5:     Outcome ← $o$, where $o \sim Prob_i$(State)
6:     Time ← Time + $t$, where $t \sim T_i$(State)
7:     State ← $E_i$(State,Outcome)
8: end while
```

VI. Case Study of a SCADA System

We implemented the ADVISE method as a new atomic model type in the Möbius modeling tool [1], which we used to perform a case study. This case study demonstrates how the quantitative security metrics produced by ADVISE can aid system design and provide insight on system security.

System design often involves trade-off decisions involving cost, response time, reliability, security, and other performance metrics. Often, improving one aspect of the system performance (i.e., reliability) negatively impacts another (i.e., cost). Quantitative security metrics enable system architects to make informed trade-off decisions by quantifying the change in security when the system design is modified.

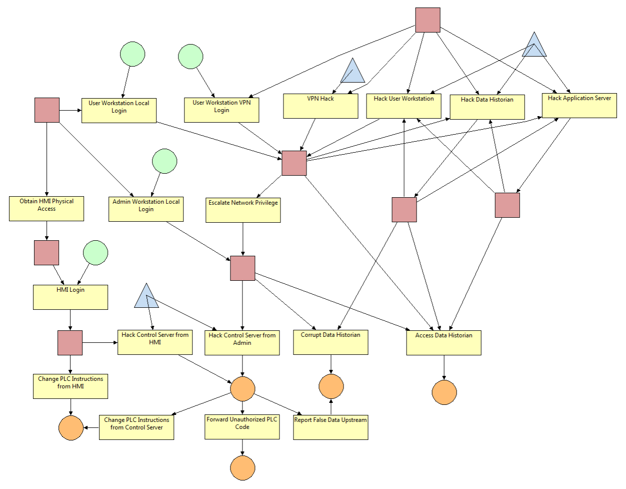
User Workstation Local Login
User Workstation VPN Login
VPN Hack
Hack User Workstation
Hack Data Historian
Hack Application Server
Obtain HMI Physical Access
Admin Workstation Local Login
Escalate Network Privilege
HMI Login
Hack Control Server from HMI
Hack Control Server from Admin
Corrupt Data Historian
Access Data Historian
Change PLC Instructions from HMI
Change PLC Instructions from Control Server
Forward Unauthorized PLC Code
Report False Data Upstream

Figure 6. The structure of the attack execution graph for the non-DMZ SCADA architecture contains multiple paths to the attack goals.

For the case study, we use ADVISE to evaluate the security of two variants of a Supervisory Control and Data Acquisition (SCADA) system architecture. We measure the security of the system relative to five different adversaries: a nation-state, a terrorist organization, a lone hacker, a disgruntled employee, and a disgruntled system administrator.

A. SCADA System Architectures

SCADA systems use a centralized control center to remotely monitor and control devices located at distant field sites. SCADA systems are used to control electric power grids, water distribution and wastewater collection systems, and oil and natural gas pipelines. We analyze two SCADA system architectures presented in Figures 5-1 and 5-3 of the National Institute of Standards and Technology Guide to Industrial Control Systems Security (NIST SP 800-82) [2].

Both SCADA system architectures consist of a corporate network and a control network separated by a firewall. The corporate network is also connected to the Internet through another firewall. In one architecture, the data historian is located in the corporate network. In the other architecture, a demilitarized zone (DMZ) is added between the control and corporate networks, and the data historian is moved to the DMZ network. A data server is also placed in the DMZ to mediate communication between the data historian and machines in the corporate and control networks. In the non-DMZ architecture, machines in the corporate and control networks can communicate with the data historian directly.

The AEG that represents possible attacks against the non-DMZ architecture is shown in Figure 6. This AEG consists of 18 attack steps, eight access domains, four knowledge items, two attack skills, and five attack goals. The AEG for the DMZ architecture contains two more attack steps and one more access domain.

Table I
ATTACK PREFERENCE WEIGHTS FOR FIVE ADVERSARIES

Adversary	***Cost***	***Payoff***	***Detection***
Nation-State	0.01	0.40	0.59
Lone Hacker	0.20	0.40	0.40
Terrorist Organization	0.05	0.80	0.15
Disgruntled Employee	0.40	0.50	0.10
Disgruntled Administrator	0.40	0.50	0.10

B. Adversaries

We consider five distinct adversaries in our analysis of the two architectures. Each has a unique profile. The *nation-state* adversary is well-funded and has a low tolerance for detection. The nation-state possesses high skill levels. The *lone hacker* possesses a high technological skill level, and gaining payoff and avoiding detection are equally important. The *terrorist organization* favors attacks with large payoff values and pays little attention to detection risk. However, limited skills may make some attack steps unavailable or unattractive. The *disgruntled employee* and *disgruntled system administrator* have limited resources but start the attack with an insider advantage. Table I lists the attack preference weights for each adversary. For this analysis, the planning horizon (N) is four for all adversaries.

C. Metrics

To assess how quickly the adversaries can achieve each attack goal $g \in G$, we measure the average amount of time during the time interval $[0, 500 \text{ min}]$ spent in a model state in which goal g has been achieved.

Formally, we specify a state metric for each attack goal $g \in G$: $\langle \tau, \lambda, \sigma \rangle$, where $\tau = 500$, $\lambda = \mathit{AvgTime}$, and $\sigma(s) =$ True, if $g \in G_s$; False, otherwise.

D. Results and Analysis

For each adversary, we ran discrete-event simulations of our ADVISE executable models and measured the average time during $[0, 500 \text{ min}]$ that the system was in a secure state. Figure 7 reveals how each adversary fared in attacking the DMZ and non-DMZ SCADA architectures. The two attack goals of interest are compromising data on the data historian and compromising the control server on the control network.

In both architectures, the nation-state compromises data, but it takes longer to compromise the DMZ architecture because more attack steps are required to accomplish the compromise when the data historian is stored in the DMZ.

The lone hacker only compromises data in the non-DMZ architecture. For the lone hacker, the increased number of attack steps to compromise data in the DMZ architecture is enough to deter him or her from attempting the attack. In this case, the “do-nothing” attack step is more attractive than any of the other attack steps.

Both the nation-state and the employee compromise data in the non-DMZ architecture, but the employee achieves the data compromise much more quickly due to his or her

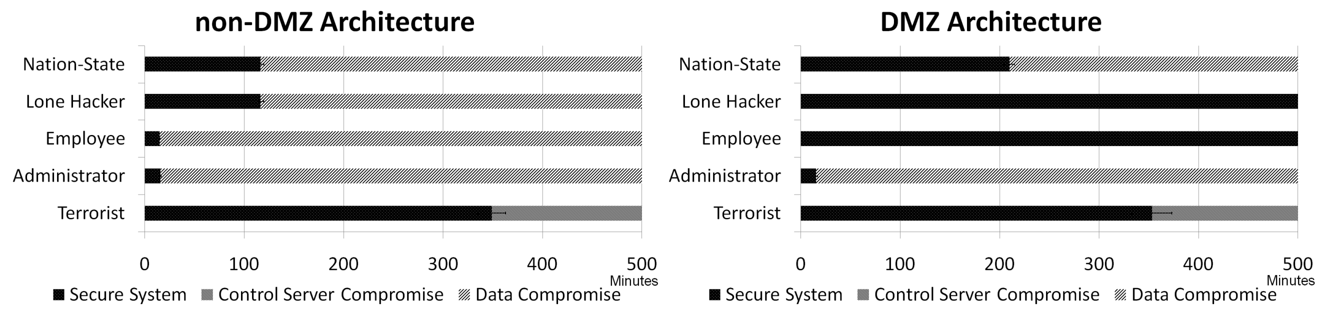

Figure 7. For each adversary and architecture combination, this graph shows the average time during $[0, 500 \text{ min}]$ that the system is in a secure state. All adversaries other than the terrorist are attempting to access data on the data historian. The terrorist is attempting to run unauthorized code on the control server.

insider status. Logging on to the corporate network is much faster than hacking in from the Internet.

However, the employee does not attempt the data compromise attack against the DMZ architecture. The employee lacks the technical attack skills needed to access the data historian when it is in the DMZ.

Unlike the standard employee, the administrator employee does have the technical attack skills to gain access to data in the DMZ, so the DMZ architecture is no more secure than the non-DMZ architecture against this threat. (Make sure you can trust your system administrators!)

The addition of the DMZ also does not affect the terrorist's attack against the control server. This is because the adversary's path to the control server on the control network does not change when the DMZ is added. The DMZ does not address the issue of control server compromise.

In summary, the case study shows that the DMZ SCADA architecture offers better protection than the non-DMZ architecture against data compromise by a nation-state, a lone hacker, or a standard employee, but an administrator employee remains undaunted by the DMZ. Also, the DMZ does not impact the ability of the terrorist to compromise the control server in the control network. In this case, adding one security defense mechanism does not protect against all types of adversaries and all types of compromise. The system architects may want to reconsider their design. The quantitative security metrics produced by ADVISE can aid system design by enabling such insights on system security.

VII. Related Security Analysis Methods

Our method is not the first instance of model-based security analysis. Attack trees [3] have been used to describe how sets of events can constitute a security compromise. Attack trees are useful for thinking about multiple ways that an attacker can reach an attack goal. However, attack trees do not contain a notion of time, and this prohibits expressing attacks as time-ordered sequences of events.

Attack graphs [4], [5], [6] and privilege graphs [7], [8], [9] extend attack trees by introducing state to the analysis. The nodes in a privilege graph represent privilege states. An attacker starts at one node in the privilege graph and works toward an attack goal by gaining privilege and transitioning to new privilege states. Attack graphs and privilege graphs enable state-based analysis, but they do not consider the different attack goals and attack preferences of individual adversaries. Our ADVISE method extends the attack graph concept by creating executable models that are customized to reflect the attack behavior of different types of adversaries.

Adversary-based analysis is the focus of some other system security analysis techniques. Mission Oriented Risk and Design Analysis (MORDA) [10] was developed by the U.S. National Security Agency (NSA). MORDA assesses system risk by calculating attack scores for a set of system attacks. The scores are based on adversary attack preferences and the impact of the attack on the system mission. Our work in characterizing adversaries was inspired by the MORDA adversary characterization.

Network Risk Assessment Tool (NRAT) [11] was also developed at NSA. NRAT assesses mission risk by computing the attack competency of potential attackers and the system vulnerability. These computations are performed by examining a set of attributes of the threat actors (adversaries), the attacks, and the information system protection (defense). NRAT analyzes each attack individually; there is no support for analyzing multiple-step attacks.

Neither NRAT nor MORDA is designed for state-based analysis. The adversary attack decision represented in these methods is a one-time selection of a full attack path. In contrast, our ADVISE method models step-by-step decisions, in which the outcome of previous attack step decisions impacts the adversary's subsequent decisions.

While ADVISE focuses on analyzing architectural-level vulnerabilities, other complementary analysis methods focus on finding implementation vulnerabilities. For example, NetSPA [12] (and its successor GARNET [13]) from MIT Lincoln Labs and other similar tools require that network scanners be deployed on operational systems. These tools perform detailed configuration analysis of deployed systems. In contrast, ADVISE can be used for design decisions before the system is deployed or before network changes are implemented. The security of several configuration options can be analyzed before one is chosen for deployment.

The early development of the ADVISE method is described in [14]. As the method has matured and been implemented as a tool, many parts have been modified. This paper takes precedence over [14].

VIII. Conclusions

To produce model-based quantitative security metrics, we have formally defined and implemented the ADversary VIew Security Evaluation (ADVISE) method. In an ADVISE model, attack steps are precisely defined and organized into an attack execution graph, and an adversary profile captures a particular adversary's attack preferences and attack goals. The ADVISE model execution algorithm uses the adversary profile and the attack execution graph to imitate how the adversary is likely to attack the system. The adversary selects the best next attack step by evaluating the attractiveness of several attack steps, considering cost, payoff, and the probability of detection. The State Look-Ahead Tree (SLAT) recursively computes how future attack decisions influence the attractiveness values of the current attack step options.

System architects can use ADVISE to compare the security strength of system architecture variants and analyze the threats posed by different adversaries. The practical application of ADVISE was demonstrated in a case study of four adversaries attacking two variants of a SCADA architecture. This case study demonstrated how the quantitative security metrics produced by ADVISE can aid system design and provide insight on system security.

Acknowledgments

The work depicted here was performed, in part, with funding from the Department of Homeland Security under contract FA8750-09-C-0039 with the Air Force Research Laboratory. We particularly wish to thank Douglas Maughan, Director of the Cyber Security Division in the Homeland Security Advanced Research Projects Agency (HSARPA), within the Science and Technology (S&T) Directorate of the Department of Homeland Security (DHS).

The authors also acknowledge valuable technical contributions from Willard Unkenholz and Donald Parks, U.S. Department of Defense, during the early development of the ADVISE method. We thank Jenny Applequist for her editorial comments.

References

[1] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius framework and its implementation," *IEEE Trans. on Software Engineering*, vol. 28, no. 10, pp. 956–969, Oct. 2002.

[2] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security," National Institute of Standards and Technology Special Publication 800-82, Gaithersburg, MD, September 2008.

[3] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004.

[4] O. M. Sheyner, "Scenario graphs and attack graphs," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 2004.

[5] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proc. of the 2002 IEEE Symp. on Security and Privacy*. Washington, D.C.: IEEE Computer Society, 2002, p. 273.

[6] L. Wang, A. Singhal, and S. Jajodia, "Toward measuring network security using attack graphs," in *Proc. of the 2007 ACM Workshop on Quality of Protection (QoP '07)*. New York, NY, USA: ACM, 2007, pp. 49–54.

[7] M. Dacier and Y. Deswarte, "Privilege graph: An extension to the typed access matrix model," in *ESORICS '94: Proc. of the Third European Symposium on Research in Computer Security*. London, UK: Springer-Verlag, 1994, pp. 319–334.

[8] M. Dacier, Y. Deswarte, and M. Kaâniche, "Models and tools for quantitative assessment of operational security," in *Information systems security: Facing the information society of the 21st century*, S. K. Katsikas and D. Gritzalis, Eds. London, UK: Chapman & Hall, Ltd., 1996, pp. 177–186.

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

[10] S. Evans and J. Wallner, "Risk-based security engineering through the eyes of the adversary," in *Proc. of the 2005 IEEE Workshop on Information Assurance*. United States Military Academy, West Point, NY, June 2005, pp. 158–165.

[11] B. Whiteman, "Network Risk Assessment Tool (NRAT)," *IA Newsletter*, vol. 11, no. 1, pp. 4–8, Spring 2008.

[12] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Proc. of the 22nd Annual Computer Security Applications Conference*. Washington, D.C.: IEEE Computer Society, 2006, pp. 121–130.

[13] L. Williams, R. Lippmann, and K. Ingols, "Garnet: A graphical attack graph and reachability network evaluation tool," in *Proc. of the 5th International Workshop on Visualization for Computer Security (VizSec '08)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 44–59.

[14] E. LeMay, W. Unkenholz, D. Parks, C. Muehrcke, K. Keefe, and W. H. Sanders, "Adversary-driven state-based system security evaluation," in *Proc. of the 6th International Workshop on Security Measurements and Metrics (MetriSec '10)*. New York, NY: ACM, 2010, pp. 5:1–5:9.