Implementing the ADVISE Security Modeling Formalism in Möbius

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

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

***Abstract*—The ADversary VIew Security Evaluation (ADVISE) model formalism provides a system security model from the perspective of an adversary. An ADVISE atomic model consists of an attack execution graph (AEG) composed of attack steps, system state variables, and attack goals, as well as an adversary profile that defines the abilities and interests of a particular adversary. The ADVISE formalism has been implemented as a Möbius atomic model formalism in order to leverage the existing set of mature modeling formalisms and solution techniques offered by Möbius. This tool paper explains the ADVISE implementation in Möbius and provides technical details for Möbius users who want to use ADVISE either alone or in combination with other modeling formalisms provided by Möbius.**

***Index Terms*—Quantitative Security Metrics, State-based Security Model, Möbius Atomic Model Formalism**

I. Introduction

The Möbius discrete-event modeling environment is a framework that supports multiple modeling formalisms and multiple solution techniques [1]. The framework is based on an abstract functional interface (AFI) that facilitates the addition of new modeling formalism modules and new solver modules [2][3]. Many of the modeling formalisms available in the Möbius tool are often used in system performance and dependability modeling [4]. However, the most recent addition to Möbius is a new atomic model formalism designed specifically for system security analysis. The new ADversary VIew Security Evaluation (ADVISE) modeling formalism was developed to provide quantitative, state-based analysis of system security [5].

There is a need for quantitative security analysis because although security was once considered a binary quantity (the system is either secure or not), we now recognize that levels of security are more accurately regarded as a continuum. At one end of the continuum, a system is perfectly secure. At the other end, a system is completely insecure. Real-world systems fall somewhere between these extremes, with some systems being more secure than others. Security metrics enable analysts to quantify the differences in security between systems. Which system is more secure? And, how much more secure is it?

There is also a need for state-based security analysis because an attack against a system often progresses in stages as an attacker penetrates deeper into the system, compromising the system and gaining access to and knowledge about the system. State-based analysis techniques can aid security analysts in identifying and correcting the weakest points in a system's defense.

The ADVISE modeling formalism aggregates security-relevant system and adversary (attacker) information that is then converted into an executable discrete-event model. The ADVISE model execution algorithms simulate how the adversary is likely to attack the system and how well the system is likely to defend itself against attack.

Just as discrete-event modeling can be used to predict the performance and dependability of a system by modeling relevant system state and state transitions, discrete-event modeling can also be used to predict the security of a system against an attacker. For a security model, relevant system state includes the adversary's access to different system domains and subnetworks, the adversary's specialized knowledge about the system (passwords, architecture details, etc.), and the adversary's progress in achieving system compromise attack goals. For a security model, relevant state transitions are the result of the adversary's attack attempts—sometimes successful, other times not.

The primary contribution of this paper is a detailed discussion of the implementation of the ADVISE formalism within the Möbius tool. A complete discussion of the theoretical foundation of the ADVISE formalism is available in [6] and [7].

Section II provides an overview of the two main components of an ADVISE model: the attack execution graph and the adversary profile. Section III describes the implementation of the ADVISE atomic model formalism within the Möbius framework. Section IV describes how the Möbius tool implementation of ADVISE performs in simulation and how optimizations have been implemented to improve performance. Section V compares the Möbius tool with other security analysis tools.

II. ADVISE Formalism Review

An ADVISE atomic model consists of an attack execution graph, which defines the set of attack steps that an attacker might attempt in order to achieve his or her goals, and an adversary profile, which details various attributes of the attacker. During the execution of an ADVISE model, an adversary evaluates the state of the system, determines the most attractive attack step to attempt next, and then attempts the attack step. This adversary decision process is repeated throughout the entire simulation execution [6] (as further described in Section II-C).

A. Attack Execution Graph

An attack execution graph (AEG) is a directed bipartite graph (S, V, A) consisting of attack step nodes (S), state variable nodes (V), and directed arcs between elements of S and V (A). State variable nodes store the current state of a model during execution. For example, state variable nodes in an AEG can store whether or not the adversary has knowledge of an administrator password, whether or not the adversary has access to a workstation, and the level of skill an adversary possesses in performing SQL injection attacks. Each attack step node describes an attack action that an adversary assaulting the system might attempt. The attack step has preconditions, which must be satisfied before the adversary can attempt it, and a set of outcomes, from which one outcome will be stochastically chosen once the adversary attempts the attack step.

State variables connected to an attack step with an incoming arc (i.e., the directed arc is pointing from the state variable to the attack step) may be used in the precondition, outcomes, or other attack step parameters. State variables connected to an attack step with an outgoing arc may be affected by one or more outcomes of the attack step.

1) State Variables: State variable nodes in an AEG come in four varieties: access, knowledge, skills, and goals. Access state variables represent the kinds of system access an adversary can obtain, e.g., user workstation access. Knowledge state variables model the pieces of knowledge an adversary can obtain, e.g., administrator passwords. Skill state variables store a skill proficiency level an adversary possesses in some kind of skill, e.g., lockpicking skill. Goal state variables represent the goals an adversary may attempt to achieve during execution, e.g., deface public web server.

Access, knowledge, and goals are Boolean variables that can change value during the execution of the model. For example, if an attacker has *web server access* and *root password knowledge*, an attack step can be performed that would shut down the web server machine; if that step is successful, the attacker would achieve the *denial of web server service goal* (and the attacker would lose the web server access). As implemented, skill state variables are constant and contain a value between 0 and 1000. The value of the skill state variable represents the skill proficiency level of an adversary, with 1000 being perfect proficiency and 0 being no proficiency.

2) Attack Steps: Attack steps are the only AEG elements that can cause the model to change state. An attack step has a precondition expression, a time distribution, a cost expression, and a nonempty set of outcomes. The precondition expression of an attack step can be any mathematical expression that returns true or false indicating whether an attack step can be attempted. The time distribution is any of the probability distributions supported by Möbius and, when sampled, will provide the nonnegative time it takes to attempt the attack step. The cost expression can be any mathematical expression that yields an integer indicating the general cost of an attack step. The precondition expression, cost expression, and parameters of the time distribution can all use state information in the expression.

Each attack step outcome consists of an outcome probability expression, a detection probability expression, and an effects expression. The outcome probability expression is any mathematical expression that returns a positive real number. When an attack step completes, the outcome probability expression for each outcome is evaluated and normalized, and an outcome from the set is uniformly selected. The detection probability expression is any mathematical expression that returns a real number between 0 and 1 and represents the risk of being detected if this outcome is chosen when the attack step completes. The effects expression specifies how the model state will be altered when the attack step completes, if this outcome is chosen.

B. Adversary Profile

While the attack execution graph defines a set of possible attack steps and how they would impact the state of the system and the adversary, the adversary profile provides values for a set of adversary attributes that are necessary for the execution of the model.

The adversary profile defines the set of access and knowledge the adversary possesses in the initial state of the model. For every skill defined in the attack execution graph, the adversary's proficiency level in that skill is stated in the profile. Finally, the profile specifies the set of the adversary's goals and the associated payoff values for the adversary.

The adversary profile also contains preference weights [8][9], which specify how the adversary values cost minimization, payoff maximization, and detection avoidance, and a planning horizon, which defines how many attack steps into the future an adversary can consider in making decisions.

C. Model Execution Algorithm

The execution cycle of an ADVISE model has two phases: the adversary decision phase and the adversary action phase. Before an adversary can perform an attack step, he or she must choose an attack step to perform. During the adversary decision phase, the adversary explores a state look-ahead tree (SLAT) [6], which defines a hierarchical state space that is rooted at the current state. Each node in the SLAT is a model state, and the arcs are attack step/outcome pairs. The adversary builds the SLAT with a depth equal to the planning horizon.

At the leaves of the tree, the state utility is calculated using the cost, detection, and payoff preference weights. This utility is then propagated back up the branches of the tree to the root branches, where the most attractive next attack step is chosen. (A more detailed explanation is available in [6].)

The adversary action phase begins once a most attractive next attack step has been selected. The adversary attempts this attack step, and an outcome from the attack step is stochastically chosen. The effects of the outcome fire, which may update the state, and the adversary decision phase begins again. The execution of an ADVISE model alternates between these two phases as long as the model executes.

In an analytical solution, rather than simulating the adversary's path through the potential state space, the reachable state space is generated. Given the adversary's initial starting state, the most attractive next attack step is computed and the states resulting from all of the potential outcomes are used as new initial starting states. Once the set of reachable states is computed, they can be rearranged into a state transition matrix, from which typical Markov chain analysis can be done [10].

III. Implementation in Möbius

The ADVISE atomic model formalism implementation in the Möbius framework provides a graphical front-end for creating and modifying ADVISE models. The model definitions are stored in a textual, XML-based format. Möbius then uses code from the ADVISE implementation to generate C++ code that compiles and links with Möbius framework libraries, creating an executable model. The generated code contains a set of classes that inherit from base classes in the Möbius AFI and ADVISE implementation.

A. ADVISE Atomic Model Editor

The ADVISE graphical editor, like all other graphical components of Möbius, uses Java, but it breaks new ground with its use of Eclipse libraries. Eclipse is a popular integrated development environment and application framework. The ADVISE implementation leverages the Eclipse Rich Client Platform (RCP) [11] to provide a clean interface and enable several useful sub-projects. Thanks to the Eclipse RCP, the ADVISE implementation uses SWT [12] (a windowing toolkit), JFace [13] (a UI toolkit), and GEF [14] (a graphical editing framework), which are all provided by the Eclipse community. In addition, the Eclipse RCP application framework is used in the latest version of Möbius; it provides console logging, error reporting, UI customization, and greatly improved Möbius plug-in management. The existing Möbius components have remained functional following the incorporation of Eclipse RCP, but the addition of Eclipse RCP marks a significant change in the way Möbius components will be written in the future.

The Eclipse RCP framework provides a large set of top-level UI components, called editors. The ADVISE atomic model editor is a `MultiPageEditorPart` with two pages, one for the attack execution graph editor and one for the adversary profile editor. While the multi-page editor breaks the UI into those two facets, the editor works on a single ADVISE model.

1) Attack Execution Graph Editor Page: The attack execution graph editor (Fig. 1) consists of a graphical canvas on which AEG elements can be positioned and connected by directed arcs. The various kinds of AEG elements have different colors and also different shapes, to allow for gray-scale presentation. Access elements are red squares, knowledge elements are green circles, skill elements are blue triangles, attack steps are yellow rectangles with their names displayed inside the object, and attack goals are orange ovals. To the left of the canvas is a palette containing new objects that can be dropped onto the canvas. A connection tool is also available in the palette, and arcs are automatically laid out by the canvas to provide the clearest presentation.

When an individual node is selected on the graph, a details view appears, docked to the right of the canvas. Fig. 1 shows the details view that appears when an attack step is selected. The details view presents different sets of information based on the type of object selected. An access object only has name and code name (name of this object when used in a code expression) fields, but an attack step requires more information, which is organized into four collapsible sections: attack cost, attack execution time, preconditions, and outcomes.

The attack cost section provides a text box for entering an attack cost code expression. The expression can be any valid C++ code fragment that returns an integer. The attack execution time section offers a probabilistic distribution widget that allows the user to choose a type of distribution and enter code expressions for each of the distribution parameters. The preconditions section provides a text box for entering a precondition code expression that must return a Boolean value that indicates whether it is possible for the adversary to attempt this attack step. A list of visible state variable objects (objects that are connected to the step by an incoming arc) is provided as a convenience to the user; the user can double-click on a list item to insert its code name into the text box.

Finally, the outcomes section provides a set of controls that can be used to change the quantity of outcomes and edit the details for each outcome. Each outcome has a name field, an outcome probability, a detection probability, and an effects code expression. The name is a simple human-readable label. The outcome and detection probabilities are both code expressions that must return a real number between 0 and 1 inclusive. The effects code expression allows the user to express how the state of the model will be affected should the outcome be selected when the attack step is attempted. A list of visible state variable objects (objects that are connected to the step by an arc in either direction) is provided as a convenience to the user. Again, one can double-click on a list item to insert the code name of that object into the text box.

Since almost all fields of an attack step's details are code expressions of some form, the modeler is provided with great flexibility. Also, the values of code expression fields can call upon state information using variable names defined by the rest of the attack execution graph. For example, an execution time

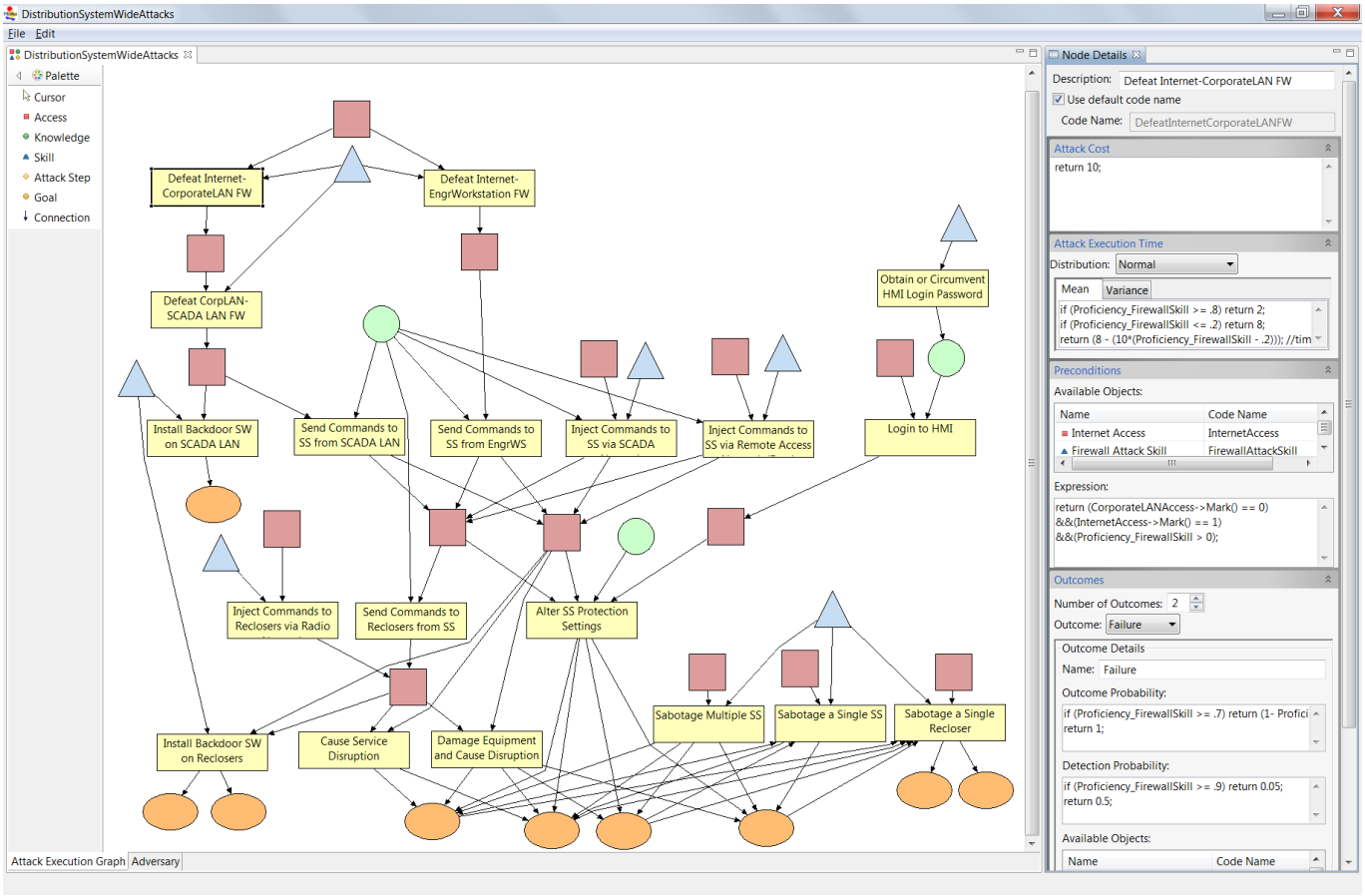

Fig. 1. An example of the ADVISE attack execution graph editor page.

mean can be one value if an adversary has a specific access, but otherwise be a function of the adversary's proficiency in a certain skill.

2) Adversary Profile Editor Page: The adversary profile editor page (Fig. 2) consists of name and code name fields and several collapsible sections: decision parameters, skills, initial access, initial knowledge, and goals. The decision parameters section contains the planning horizon, attack preference weights, and future discount factors. The planning horizon field defines the depth to which the decision algorithm will build the SLAT. Attack preference weight fields for cost, detection, and payoff each take a real number between 0 and 1 inclusive. The future discount factor fields allow a linear discounting factor to be applied to the values of the future attack step cost, detection, and payoff.

The skills section provides a table containing the set of skills the adversary possesses and a proficiency rating for each skill. Skills can be added and removed with a simple wizard that offers the set of skills defined in the AEG. The initial access and initial knowledge sections provide a similar table containing the set of access and knowledge objects that the adversary possesses at the beginning of the model execution. Finally, the goals section provides a table similar to the skills table, but with a column for payoff values, which can be any integer, instead of proficiency values. Goals and skills that are not added to the adversary profile are considered to have a payoff value or skill proficiency of 0.

B. Möbius AFI Overview

The Möbius AFI defines a set of classes from which all formalisms must inherit. Given the principles of object-oriented programming and its implementation in C++, subclasses are required to implement certain virtual member functions that the Möbius framework uses to interact with the formalism implementation in a well-defined manner [2].

Every atomic model formalism in Möbius must translate its formalism to a common language that the AFI works with [2]. The language consists of state variables and actions. A state variable is an object that stores a value of a defined type. State variables must inherit from and implement all virtual functions of the `BaseStateVariableClass` class (a base class in the Möbius AFI). All ADVISE state variables inherit from the `SharableSV` class, which is a child class of `BaseStateVariableClass`. By inheriting from `SharableSV`, ADVISE models may share state variables with other atomic models via Möbius's composition formalisms [15].

An action is a model element that changes the value of state variables and thereby transitions the model from state to state. An action contains details on timing, an enabling predicate function, an input function, an output function, a grouping ability to enable stochastic choice of an action from a group of

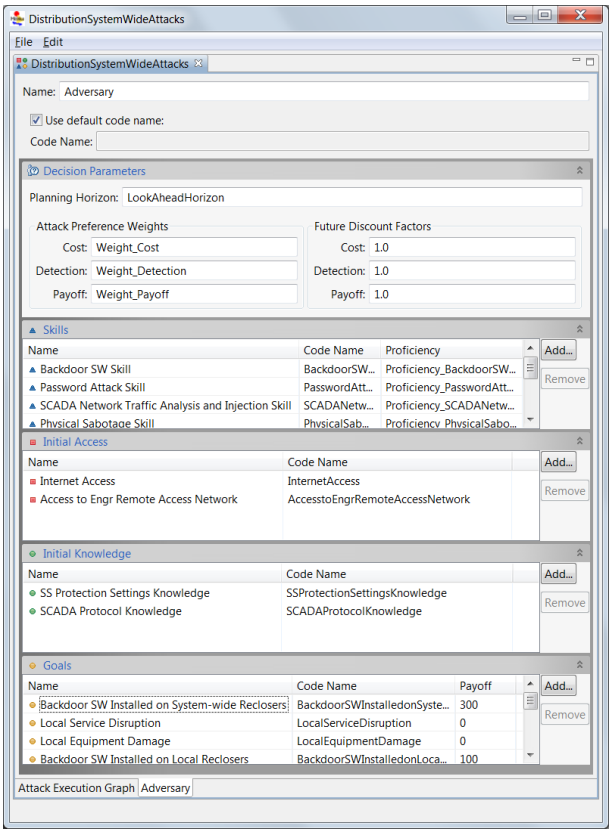

Fig. 2. An example of the ADVISE adversary profile editor page.

actions to fire when the group completes, and much more. The base class for all Möbius actions is the `BaseGroupClass`.

C. ADVISE AFI Implementation

The `ADVISEModel` is the top-level object that stores pointers to the model's state variables and actions, which will be accessed and evaluated by other components in the Möbius framework, such as reward variable models and the discrete-event simulator. The `ADVISEModel` also hosts functions that implement algorithms unique and necessary to the ADVISE formalism. For example, a member function called `recalculateAdversaryDecisionWeights()` builds the SLAT and calculates the attractiveness of every available attack path.

ADVISE models have six state variable classes, all with an integer type: `Access`, `Skill`, `Knowledge`, `Goal`, `BeginAdversaryDecision`, and `StepChosen`. The first four classes directly correspond to the ADVISE state variables discussed in Section II-A1. The last two will be discussed shortly. ADVISE models have two action classes: `AdversaryDecision` and `Step`. A step action is actually the implementation of an attack step/outcome pair. The `AdversaryDecision` action group will be discussed shortly.

Fig. 3 shows an action-state variable diagram representing a translated ADVISE atomic model. Large white circles are state variables, and dark vertical oblongs are action groups;

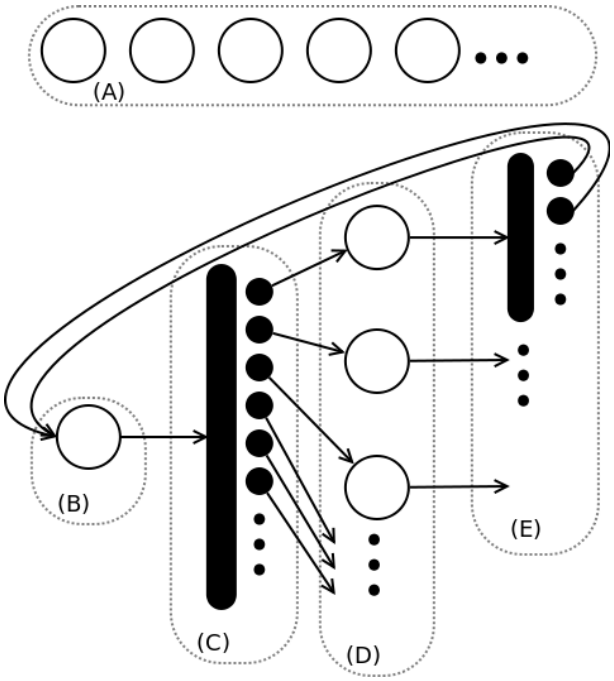

Fig. 3. The action-state variable translation of an ADVISE atomic model. (A) ADVISE model state variables (access, knowledge, skill, goals). (B) BeginAdversaryDecision state variable. (C) AdversaryDecision action group. (D) StepChosen state variables. (E) Attack step/outcome action groups.

individual actions are indicated by small dark circles to the right of the groups. The arcs in the diagram express the flow of tokens through the model. Part A of Fig. 3 shows the set of all ADVISE state variables defined in the model (access, knowledge, skills, and goals). These state variables are not connected by arcs, but are globally accessible by all actions.

Every ADVISE model has just one `BeginAdversaryDecision` (B in the figure) instance, and when that contains a token, it signifies that the adversary decision algorithm should be executed next. At the beginning of the model's execution, a single token exists in the `BeginAdversaryDecision` instance. When a token exists in the `BeginAdversaryDecision` instance, the `AdversaryDecision` action group (C in the figure) becomes enabled. For every attack step in a given ADVISE model, an `AdversaryDecision` action exists in the translation. When the group is enabled, the adversary decision algorithm in the `ADVISEModel` object is executed, which returns the most attractive next step, and the token is placed into one of the `StepChosen` instances (D in the figure). Each `StepChosen` instance corresponds to an ADVISE attack step. When the `StepChosen` instance contains a token, it indicates that the adversary has decided to attempt the corresponding attack step. At the beginning of model execution, all `StepChosen` instances have zero tokens. Finally, the action groups in part E of Fig. 3 represent attack steps in the ADVISE model. Each action in the group corresponds to an outcome in the attack step. Thus, the `Step` instances are actually ADVISE step-outcome pairs.

When an attack step is chosen, the group becomes enabled

and begins firing according to the details specified in the ADVISE model. Once the step has completed firing, an outcome is chosen according to the outcome probabilities described in the model. The outcome's effects code expression is executed, which updates the state of the ADVISE model (section A in Fig. 3), and the token moves from one of the `StepChosen` instances back to the `BeginAdversaryDecision` instance. Parts B and C of the figure show the adversary decision phase, and parts D and E show the adversary action phase, as described in Section II-C.

D. Composing ADVISE Models

ADVISE models are designed to be composable with other Möbius models. A model can share any of the six types of state variables described in Section III-C.

The four types of ADVISE state variables are named using the user-supplied code names. The `BeginAdversaryDecision` instance is always named `MakeDecision`, and the `StepChosen` instances are named using the format `<Attack Step Code Name>Chosen`.

The action-synchronization composition formalism will also synchronize any of the action groups defined earlier with action groups from other atomic models. The `AdversaryDecision` action group is always named `AdversaryDecisionGroup`, and the `Step` action groups are always named using the format `<Attack Step Code Name>Group`.

The adversary attack decision algorithm only considers the information contained in the attack execution graph and the adversary profile; the algorithm is blind to the effects of attack decisions on other Möbius models. The SLAT does not incorporate any information contained in other Möbius models, and therefore the decision algorithm does not consider, in any way, the state transitions that may occur in other Möbius models as a result of attack actions. For example, if an ADVISE model and a stochastic activity network (SAN) [16] model are composed together, the behavior of the SAN model and any effect it has on the value of shared state variables will not be considered in the adversary decision algorithm.

E. Specifying Metrics

Metrics for an ADVISE model can be defined using the standard performance variable model available in Möbius. Reward code expressions can leverage any of the state variables described above, and impulse rewards can accumulate upon the firing of any of the actions described earlier.

Metrics can assess the attack goals an adversary can achieve. Metrics can provide insight on common or likely targets, the preferred attack path of an adversary, or the criticality of certain components. Also, metrics can aid in impact assessment. Security analysts are able to capture a wide range of security-relevant measurements from ADVISE models.

F. Using Studies with ADVISE

Global variables may be used throughout the ADVISE model specification. Studies can be defined, providing values for those global variables in order to investigate different model parameters. It is important to remember that all state variables in an ADVISE model are integer types, and that the global variables that would initialize the state variables should be typed as integers.

Studies have been used to compare the relative security of different system configurations as well as the impact of different adversaries; for example, studies have examined the effects of including or not including a DMZ network in a networked system [6].

G. Möbius Solution Techniques

Möbius comes with a discrete event simulation solver and a nice array of specialized analytical solvers. The Möbius tool links together the previously built atomic model libraries, composed model libraries, reward model libraries, study libraries, and the simulation solver libraries to create a binary executable application that performs the complete model simulation. Observed values for each of the reward variables are gathered across many simulation iterations, and statistics are calculated, e.g., mean and variance.

In order to use the analytical solution techniques offered by Möbius, it is necessary to generate a state space, which is really a compact representation of a Markov chain which the analytical solver will execute. Not all models in Möbius can be transformed into Markov chains, so various restrictions are detailed in the Möbius manual [17], defining which classes of models are candidates for analytical solution. At this time, only ADVISE models which place restrictions on the Adversary utility functions are candidates for analytical solution, as outlined in [10]. This is because, in general, the adversary's decision is not solely dependent on the current state and, therefore, the Markov chain representations of ADVISE models cannot be generated.

More details on model composition, metric specification, experimental study definition, and solution methods in Möbius can be found in [17].

IV. Performance Analysis and Optimizations

In this section, we will explore the performance of the tool in the simulation phase. We identify key differences between simulation for ADVISE and typical simulation. By optimizing ADVISE's unique characteristics, we significantly improve performance.

A. Adversary Decision Computation

A typical simulation loop consists of identifying enabled transitions, selecting one probabilistically, and changing model state. By including the adversary decision in the simulation loop in ADVISE, we require that an optimization problem be solved for each state transition. The inclusion of the adversary decision poses a significant performance hurdle.

The time to complete the adversary decision algorithm corresponds directly to the size of the state lookahead tree (SLAT). For typical models, the SLAT size can be hard to compute, since the number of enabled attacks is not known a

priori. However, for a general model with A attack steps, each attack step has O or fewer outcomes, and the adversary has a look-ahead of L, the upper bound of the size of the SLAT is A^{O^L}. Solving for the adversary decision is exponential in the look-ahead.

The exponential algorithm is executed for each state the adversary enters, during thousands of simulation iterations. However, the adversary's decision is static with respect to the state. By caching the decision associated with a state, we avoid subsequent rebuilding of the SLAT, and the complexity of the decision algorithm is reduced from construction of the SLAT to a simple lookup. Using a cache allows us to quickly return the adversary's decision if the adversary revisits a state.

However, we can utilize the cache for more than a single simulation iteration. In order to return metrics that are within the specified confidence interval, we run a given experiment thousands of 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 100% cache hit rate after the first few simulation iterations, since only the attack outcomes are probabilistic.

B. SLAT Construction

The simplest method of executing the adversary decision algorithm is to construct the SLAT in a depth-first manner. That method leads to the A^{O^L} SLAT size mentioned above. An improvement is to use a graph-based approach, storing intermediate results of the SLAT and reusing them if the same state is encountered again at the same look-ahead level.

We accomplish this, not by constructing the entire graph and iterating over it, but rather by caching the cost, detection probability, and expected payoff of the sub-tree for each state and look-ahead level pair. In practice, that is an extension to the depth-first exploration of the SLAT, which significantly reduces the branching of the SLAT.

Because all intermediate values are cached, the size of the SLAT is bounded by the look-ahead, L, times the size of the state space $(L * 2^{|A||K||G|})$. 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.

C. Performance Data

In Table I, we present full execution timings for the base implementation (Base), when caching was used at the decision level (ADA Cache), when caching was used within the SLAT (SLAT Cache), and finally when both caches were used simultaneously (SLAT & ADA). The hyphen indicates that one simulation could not be included in the data because it took too long to execute. Those timings are generated from an easily extensible model, which is fully described in [7].

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 size of the SLAT, since two attack steps are always enabled, and each attack step has two outcomes. The size of the SLAT is 4^N, where N is the look-ahead.

Look-ahead	Base	SLAT Cache	ADA Cache	SLAT & ADA
3	4.79	5.23	0.57	0.60
4	17.80	8.96	0.57	0.57
5	69.12	13.89	0.57	0.58
6	289.81	19.78	0.58	0.58
7	1134.02	26.99	0.60	0.57
8	4303.11	34.63	0.65	0.57
9	18455.96	42.50	0.94	0.57
10	-	50.64	1.94	0.56

TABLE I
SIMULATION TIMING (IN SECONDS)

The base algorithm scales poorly with respect to the look-ahead. By caching within the SLAT, essentially converting the tree to a graph, we achieve significant improvements. However, those computations are repeated for each of the 100,000 runs. Caching the adversary's decision results in a single decision computation over all of the simulation iterations for each state. The times, which are shown in the ADA Cache column of Table I, approach the lowest timing bound, because of the simulation overhead. However, for large look-aheads, even a single decision calculation takes a significant amount of time unless it is optimized. When we combine the two methods, the execution time remains close to the simulation overhead threshold, even with large look-aheads.

V. RELATED TOOLS

The ADVISE formalism implementation in Möbius is not the only system security analysis tool.

Researchers at George Mason University developed the Topological Vulnerability Analysis (TVA) tool [18] to generate attack graphs based on input from a network vulnerability scanner. They developed a database of vulnerabilities that specifies a precondition and postcondition for each vulnerability. To create an attack graph, the system information from the scanner and the vulnerability information from the database are combined with information about the starting state and goal state of the attacker. The analysis of the attack graph finds a minimum set of conditions such that the attacker can reach the attack goal.

By adding visualization capabilities, the researchers at George Mason University developed a tool called Combinatorial Analysis Utilizing Logical Dependencies Residing on Networks (CAULDRON) [19], which is now available as a commercial product.

Researchers at MIT Lincoln Laboratory developed a system called NETwork Security and Planning Architecture (NetSPA) [20] to generate attack graphs. NetSPA receives input data from a network vulnerability scanner and vulnerability databases. NetSPA also computes reachability, while assuming monotonicity and independence of the time required to execute an attack.

To facilitate the analysis of attack graphs generated by NetSPA, an interactive visualization tool called Graphical At-

tack Graph and Reachability Network Evaluation Tool (GARNET) [21] was developed. Like CAULDRON, GARNET displays the network topology and attack steps on that topology. Further visualization improvements were implemented in the next iteration of the tool, called NAVIGATOR (Network Asset VIsualization: Graphs, ATtacks, Operational Recommendations) [22]. NAVIGATOR's main contribution is the overlaying of results on the network topology map.

Both CAULDRON and NAVIGATOR focus on the reachability of an attack goal. They only analyze live systems based upon scanner input, and assume that there is no defensive response. In contrast, ADVISE simulates adversary behavior over time using attractiveness functions, provides the ability to model arbitrary vulnerabilities, analyzes real or yet unrealized systems, incorporates business and operational models, and provides predictive probabilistic security metrics.

VI. Conclusion

The main contribution of this paper is a detailed discussion of the implementation of the ADVISE formalism within the Möbius tool, and performance enhancements to solving ADVISE models via simulation.

The ADVISE atomic model formalism implementation provides a graphical editor that modelers can use to construct ADVISE models. Because we implemented the ADVISE atomic model to conform with the standardized Möbius AFI, ADVISE models are fully integrated in the Möbius tool. The addition of ADVISE to Möbius allows users to easily model system security and adversary preferences, and quickly compute relevant quantitative security metrics.

VII. Acknowledgments

The work described in this paper was conducted, in part, with funding from the Department of Homeland Security under contract FA8750-09-C-0039 with the Air Force Research Laboratory and the Army Research Office under Award No. W911NF-09-1-0273. 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 would like to acknowledge the contributions of current and former members of the Möbius team and the work of outside contributors for the Möbius project. The authors would also like to thank Jenny Applequist for her editorial work.

References

[1] D. 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 Transactions on Software Engineering*, vol. 28, no. 10, pp. 956–969, Oct. 2002.

[2] J. M. Doyle, "Abstract model specification using the Möbius modeling tool," Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, January 2000.

[3] S. Derisavi, P. Kemper, W. H. Sanders, and T. Courtney, "The Möbius state-level abstract functional interface," *Perf. Eval.*, vol. 54, no. 2, pp. 105–128, Oct. 2003.

[4] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders, "Möbius: An extensible tool for performance and dependability modeling," in *Computer Performance Evaluation / TOOLS*, 2000, pp. 332–336.

[5] E. LeMay, W. Unkenholz, D. Parks, C. Muehrcke, K. Keefe, and W. H. Sanders, "Adversary-driven state-based system security evaluation," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics (MetriSec 2010)*, Bolzano-Bozen, Italy, Sept. 15, 2010.

[6] E. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke, "Model-based security metrics using ADversary VIew Security Evaluation (ADVISE)," in *Proceedings of the 8th International Conference on Quantitative Evaluation of SysTems (QEST 2011)*, Aachen, Germany, Sept. 5–8, 2011, pp. 191–200.

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

[8] D. Buckshaw, G. Parnell, W. Unkenholz, D. Parks, J. Wallner, and O. S. Saydjari, "Mission oriented risk and design analysis of critical information systems," *Military Operations Research*, vol. 10, no. 2, pp. 19–38, 2005.

[9] S. R. Watson and D. M. Buede, *Decision Synthesis: The Principles and Practice of Decision Analysis*. Cambridge University Press, 1987.

[10] M. D. Ford, P. Buchholz, and W. H. Sanders, "State-based analysis in advise," in *Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on*, sept. 2012, pp. 148 –157.

[11] J. McAffer, J.-M. Lemieux, and C. Aniszczyk, *Eclipse Rich Client Platform*, 2nd ed. Addison-Wesley Professional, 2010.

[12] S. Northover and M. Wilson, *SWT: The Standard Widget Toolkit, volume 1*, 1st ed. Addison-Wesley Professional, 2004.

[13] R. Harris, *The Definitive Guide to SWT and Jface*, 2nd ed. Berkeley, CA, USA: Apress, 2007.

[14] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[15] A. J. Stillman, "Model composition within the Möbius modeling framework," Master's thesis, University of Illinois, 1999.

[16] W. H. Sanders and J. F. Meyer, "Stochastic activity networks: Formal definitions and concepts," in *Lectures on Formal Methods and Performance Analysis*, ser. Lecture Notes in Computer Science, E. Brinksma, H. Hermanns, and J. P. Katoen, Eds., vol. 2090. Berg en Dal, The Netherlands: Springer, 2001, pp. 315–343.

[17] Möbius Team, *The Möbius Manual, www.mobius.illinois.edu*, University of Illinois at Urbana-Champaign, Urbana, IL, 2013. [Online]. Available: https://www.mobius.illinois.edu/manual/MobiusManual.pdf

[18] 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.

[19] S. O'Hare, S. Noel, and K. Prole, "A graph-theoretic visualization approach to network risk analysis," in *Proceedings of the 5th International Workshop on Visualization for Computer Security (VizSec 2008)*, J. Goodall, G. Conti, and K.-L. Ma, Eds. Berlin, Germany: Springer-Verlag, September 2008, pp. 60–67.

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

[21] L. Williams, R. Lippmann, and K. Ingols, "Garnet: A graphical attack graph and reachability network evaluation tool," in *Proceedings of the 5th International Workshop on Visualization for Computer Security (VizSec 2008)*, J. Goodall, G. Conti, and K.-L. Ma, Eds. Berlin, Germany: Springer-Verlag, 2008, pp. 44–59.

[22] M. Chu, K. Ingols, R. Lippmann, S. Webster, and S. Boyer, "Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR," in *Proceedings of the Seventh International Symposium on Visualization for Cyber Security*, 2010, pp. 22–23.