Modeling Humans: A General Agent Model for the Evaluation of Security

Michael Rausch[1], Ahmed Fawaz[1], Ken Keefe[1], and William H. Sanders[2]

[1] Information Trust Institute, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
{mjrausc2, afawaz2, kjkeefe }@illinois.edu
[2] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
whs@illinois.edu

Abstract. Careful planning is needed to design cyber infrastructures that can achieve mission objectives in the presence of deliberate attacks, including availability and reliability of service and confidentiality of data. Planning should be done with the aid of rigorous and sound security models. A security modeling formalism should be easy to learn and use, flexible enough to be used in different contexts, and should explicitly model the most significant parts of the system of interest. In particular, the research community is increasingly realizing the importance of human behavior in cyber security. However, security modeling formalisms often explicitly model only the adversary, or simplistic interactions between adversaries and defenders, or are tailored to specific use cases, or are difficult to use. We propose and define a novel security modeling formalism that explicitly models adversary, defender, and user behavior in an easy and general way, and illustrate its use with an example.

1 Introduction

Institutions and enterprises rely on properly functioning cyber infrastructures to fulfill their intended missions. Impairments to proper operation can come from both accidental and malicious sources. While a sound engineering approach exists for designing and assessing systems that must tolerate accidental faults, there is no such corresponding methodology for systems that must be resilient to malicious attacks. Given the increasing likelihood of such attacks in the modern world, the lack of such a methodology will lead to increased uncertainty regarding the resilience of systems, and may result in designs that are susceptible to catastrophic failure when attacked.

Unfortunately, as society comes to rely more on cyber infrastructure, it is becoming an increasingly attractive target. The cyber infrastructure must be protected from the increased threat of attacks to ensure that it remains functional. Successful cyber attacks against such infrastructures can be very impactful. A few prominent examples include the cyber-attack on Ukraine's power grid in 2015, which deprived 230,000 residents of power [12]; the Stuxnet worm,

which delayed Iran's nuclear program [11]; and data exfiltration attacks on U.S. Democratic National Committee emails [2]. Those historical examples show how influential such attacks may be.

Cyber defenses must be carefully planned from the earliest stages of design to mitigate threats and ensure that systems will achieve availability, integrity, and confidentiality objectives, even in the face of attack. Practitioners and academics alike have long known the importance of accurate risk assessment [3] [21], as it is key to designing effective security architectures. Risk assessment, while important, is difficult to perform correctly.

Formal computer security models help security experts overcome limitations, gain additional insight into a system, and confirm conjectures. Assumptions are made explicit in models, and the assumptions, input parameters, methodology, and results of a model may be audited by an outside party. The modeling formalism could also serve as a common language that would allow security experts and experts from other domains to more easily collaborate on a security model. Finally, the models would add additional mathematical and scientific rigor to risk analysis. All of these benefits speak to the need for security models.

We believe that, when constructing security models, it is necessary to consider not only the system to be defended, but also the humans that interact with that system: adversaries, defenders, and users. If the model does not consider these human entities, it is much less likely to be accurate. Unfortunately, many security modeling formalisms in use today fail to explicitly model all of the human entities in the system, or do so in an overly simplistic way. Models that ignore or improperly model human users are significantly less likely to be helpful to system architects.

To address this issue, we propose a new security modeling formalism, the *General Agent Model for the Evaluation of Security (GAMES)*, which allows a system engineer to explicitly model and study the adversaries, defenders, and users of a system, in addition to the system itself. These models are executed to generate security-relevant metrics to support design decisions. The formalism is used to easily build realistic models of a cyber system and the humans who interact with it. We define an agent to be a human who may perform some action in the cyber system: an adversary, a defender, or a user. The formalism enables the modular construction of individual state-based agent models of the three types. The formalism also allows the modeler to compose these individual agent models into one model so the interaction among the adversaries, defenders, and users may be studied. Once constructed, this composed model can be executed. During the execution, each individual adversary, defender, or user utilizes an algorithm or policy to decide on what action the agent will take to attempt to move the system to a state that is advantageous for that agent. The outcome of the action is then probabilistically determined, and the state updated. Modelers using GAMES have the flexibility to determine how the agents will behave: either optimally, or according to a modeler-defined policy. The model execution generates metrics that aid in risk assessment, and helps the security analyst

suggest appropriate defensive strategies. The formalism helps security architects make cost-effective, risk-aware decisions as they design new cyber systems.

2 Related Work

Many academics and practitioners have recognized the need for models for computer security. Many examples may be found in surveys, e.g., surveys of papers on game-theoretic security modeling approaches [14] [17], a survey of attack tree and attack-defense tree models [9], and a survey that includes a useful taxonomy of security models [22].

Such modeling approaches are a step in the right direction, but pose their own sets of limitations, especially in the ways they model the humans who interact with the cyber portion of the system. Some modeling approaches explicitly model only the adversary, e.g., the well-known attack tree formalism [18]. Other formalisms model the adversary and defender, but neglect the role and impact of users. For example, the attack-defense tree formalism [8] and related attack-defense graph formalism [10] extend the attack tree formalism to include one attacker/defender pair, but do not model multiple adversaries or multiple defenders, or any users, which limits the effectiveness of the models. There exist some approaches and tools for modeling multiple adversaries, defenders, and users in a system, e.g., Haruspex [1], and some agent-based simulation approaches [4] [23]. However, these approaches and tools are not in common use, for a number of reasons. Often, the models lack realism because of model oversimplification, are tailored to narrow use cases, produce results that are difficult to interpret, or are difficult to use, among other problems. Finally, security modeling formalisms, particularly those that take a game-theoretic approach, often assume that attackers and defenders are rational. Some examples include [7] [24] [5]. However, this assumption may not produce useful security models [22].

The GAMES approach is inspired, in part, by the ADVISE formalism [13]. In particular, we extend and generalize the ADVISE formalism's Attack Execution Graph. However, there are important differences between the two formalisms. Models constructed using the ADVISE formalism can explicitly model only an adversary. Defenders, users, and the interactions between actors cannot be explicitly modeled with ADVISE. The GAMES formalism recognizes the importance of considering defender and user actions when designing secure systems, and explicitly incorporates defenders and users as first class model elements.

3 A General Agent Model for Evaluation of Security

In this section, we give a comprehensive overview of the *General Agent Model for the Evaluation of Security* (GAMES) formalism. We will first explain in detail how the individual agent models are defined and composed together. Next, we will describe how the models are executed, including the algorithms or policies that the various adversaries, defenders, or users of the system may utilize to decide upon the best course of action. Finally, we will describe the kind of results

these models will produce, how they may be interpreted, and how the models may be used. One or more agent models may be joined in a composed model. The composed model may then be combined with a reward model to create a model that may be executed to obtain security-relevant metrics, which a security analyst may use to gain additional insight into the system being modeled.

3.1 Model Formalism Definition

A model defined using the GAMES formalism will consist of one or more agent submodels and a model composed of the agent submodels, that describes how the submodels relate to one another. An agent model represents one acting entity in the cyber system: an adversary, a defender, or a user. The framework allows a modeler to define many different kinds of agents. For example, a modeler may define a malicious insider adversary, a nation-state adversary, a network operator defender, and a customer user of the system. Once the individual agent models have been composed into a model that defines their relationships, the modeler can study, for example, 1) how two adversaries may cooperate to achieve a goal on the network, 2) the effectiveness of the network operator's defensive actions, and 3) how the actions of the adversaries and defender impact the user's experience and behavior. We shall first describe the agent models, and then how they may be composed together.

Agent Model: An *agent model* describes the state an agent may read or write, how this state is initialized, the set of actions the agent may utilize to change the state of the model, the payoff the agent receives given a particular state, and the agent decision algorithm that determines the action the agent will take given the state of the system. The agent model is composed of an *Action Execution Graph* (AEG) and an *agent profile*. The AEG may be thought of as an extension of the attack-tree formalism [18]. Unlike attack trees, an AEG contains state, and therefore shares some similarities with generalized stochastic Petri nets (GSPNs) [15] and stochastic activity networks (SANs) [16]. However, it is most similar to the Attack Execution Graphs of ADversary VIew Security Evaluation (ADVISE) models. [13]. Action Execution Graphs are more general than Attack Execution Graphs, since they can be used to model any agent type, whereas Attack Execution Graphs are used only to model adversaries. The agent profile defines how the state is initially defined; the payoff function, which assigns the payoff the agent achieves given a particular model state; and the agent decision algorithm the agent uses to decide what actions to take to change the state of the system.

An *Action Execution Graph* (AEG) is a labeled transition system defined by

$$< S, A, C >,$$

where S is some finite set of state variables that an agent may read or write, A is some finite set of actions an agent may take, and the relation C defines a set of directed connecting arcs from $p \in S$ to $a \in A$, and from $a \in A$ to $e \in S$, where p is a prerequisite state variable whose value directly affects the behavior of the

action a (e.g., whether the action may be attempted, how much it will cost, and how long it will take), and e is a state variable that may be changed when action a is performed. The states and actions together, $S \cup A$, are the vertices of the AEG, while the connecting arcs, C, are the edges.

Each state variable in S may have a single value or a finite sequence of values. Each value can be drawn from any countable subset of the real numbers. The state variables may be further subdivided, at the discretion of the modeler, into different classes. For example, state variables relating to an adversary agent may be divided into those that relate to access (like physical access to the system, network access, or administrator access to individual machines), knowledge (of the routing protocols used, company policies, encryption schemes, etc.), or skill (in decryption, social engineering, privilege escalation, and the like), as proposed by LeMay [13]. This subdivision of state variables is superficial, but may serve as a conceptual aid to the modeler.

An action, $a \in A$, may be used by an agent to attempt to change the state of the system. It is defined by the tuple

$$< B, T, C, O > .$$

First, let Q be the countable set of model states, where a model state (also known as a *marking*) is given by a mapping $\mu : S \to \mathbb{R}$.

$B : Q \to \{True, False\}$ is a Boolean precondition that indicates whether the action is currently enabled. An agent may not take an action that is not enabled.

$T : Q \to \mathbb{R}^{\geq 0}$ is the length of time needed to complete the action, and is a random variable. All of the action times in the model are mutually independent.

$C : Q \to \mathbb{R}^{\geq 0}$ is the cost to the agent for attempting the action.

O is the finite set of outcomes of the action (such as success or failure). Each outcome $o \in O$ is defined by the tuple

$$< Pr, E >$$

where

$Pr : Q \to [0, 1]$ is the probability that outcome $o \in O$ will occur.

$E : Q \to Q$ is the *effect* of the outcome, i.e., the function that transitions the system to a new state upon the selection of $o \in O$.

An agent profile is composed of three distinct components. The first specifies the initial value for each state variable in the Action Execution Graph. The second is a function that accepts as input the state of the model and outputs the payoff the agent accrues given that the model is in that particular state. While in theory a modeler could choose to assign a different payoff for every state in the AEG, that may be impractical in many cases because of state-space explosion.

The third and final component of the agent profile is the agent decision algorithm. The agent decision algorithm will take as input the state of the model, and output an action. In general, the agent decision algorithm will attempt to select an action that will maximize the agent's payoff. The modeler may choose to assign to the agent one of several predefined agent decision functions, or specify a

custom decision function. The various predefined decision functions may be based on well-studied techniques drawn from game theory and artificial intelligence, or novel algorithms that may be developed in the future. If, in the opinion of the modeler, none of the predefined decision functions realistically describe an agent's real behavior, the modeler may define a custom agent decision algorithm. We will explain the various adversary decision functions in greater detail in the section on model execution.

Model Composition: Agent models should be composed together to exploit the full power of the GAMES approach. Agent models are modular and independently functional, so one agent model could be executed by itself if the application warrants. For example, if the modeler is only interested in studying the adversary's behavior and is not interested in the defender's response or the impact on the user's behavior, he or she could build a standalone adversary agent model similar to an ADVISE model [13]. However, the chief aim of the GAMES formalism is to give those in charge of making security decisions the ability to easily model the interaction among adversaries, defenders, and users in cyber systems. The composed model will define how the agent models will be allowed to interact with one another. We utilize the state-sharing approach of the Replicate/Join formalism [19] to enable agents to interact with one another. This state-sharing approach allows an agent to read and write state in another agent model. Agents can pass messages to one another through shared state variables that serve as communication channels.

3.2 Model Execution

Overview: Model execution is accomplished via discrete-event simulation paired with the agent decision algorithms, as seen in Algorithm 1.

Algorithm 1 Model Execution

```
1: procedure EXECUTE MODEL
2:     Time ← 0
3:     State ← s_0
4:     Agents ← {agent_1, agent_2, ... agent_n}
5:     while Time < EndTime do
6:         for all agent ∈ Agents do
7:             agent_action ← AgentDecisionAlgorithm(State)
8:         Time ← time to complete earliest completed agent action
9:         Outcome ← o, where o is the randomly chosen outcome of the action
10:        State ← Effect(State, Outcome)
```

Decisions: The formalism allows the modeler to specify adversary, defender, and user behavior as realistically as possible to produce high-quality results that will support design and defense decisions. Today, security modeling formalisms are commonly built around one agent behavior specification approach. The agent

behavior specification is driven by the agent decision algorithm, which the agent uses to choose actions to perform. Often this agent behavior specification is inspired by and draws from some particular technique in game theory, artificial intelligence, or psychology. The GAMES formalism may be used to test and compare a wide variety of agent decision algorithms. The modeled entities may act cooperatively or competitively with others to achieve a goal, using well-established techniques from artificial intelligence research.

3.3 Metrics

The primary intended use of the GAMES formalism is to provide insight to support those charged with making decisions that affect system security. That insight comes in the form of quantitative metrics, which are results of the executed model. We leverage the theory of reward-model-based performance variables [20] to construct metrics for GAMES models. It provides a simple formulation that is very flexible, and thereby allows us a great deal of freedom and creativity in the variety of metrics we may specify. For a demonstration, see Section 4.

3.4 Limitations

We argue that quantitative model results can produce useful insights into the security of the system being considered, but we do not claim that the metrics produced by executing a GAMES model are infallible, or that they should be treated as such. The metrics may not accurately reflect reality; they may give misleading results if the model is constructed incorrectly or if the model input parameters are inaccurate. Security analysts must work closely with system architects and other experts to verify that the model is a good representation of the actual system. In addition, whenever possible, the results given by the model should be validated by experiments performed on the system itself. Experiments performed on the system can also be used to improve the model. The adversary, defender, and user behavior in the model may be checked against data from intrusion detection systems, analytics data, academic studies, and surveys results. The issue of inaccurate input parameters may be mitigated with a well-executed design exploration, which may be achieved by a sensitivity analysis.

Some are concerned that models that produce quantitative security metrics may be misused, to the detriment of the security of systems, in part because people may have more confidence in the model results than they should [22]. We believe, however, that the alternative of having no formal security model is much worse. Security analysts almost always carry informal mental models of their systems in their heads, and those models suffer from many of the same limitations as formal security models, while also suffering from their own additional limitations. The GAMES formalism gives an analyst the ability to turn a mental model into a formal, concrete, rigorous, auditable model. Indeed, the quantitative metrics produced by these models represent an important step towards the development of a science of security [6].

4 Example

We present an example model to illustrate how security practitioners could use the GAMES formalism to make security decisions. The goal of the model is to help an operator make an informed choice among several strategies that could be used to defend user accounts from being compromised by an adversary. We limit the size of the model for ease of explanation; we could obtain a more realistic model by expanding the size to include more details. We solved the example model using an implementation of the GAMES framework written in Python.

In the model, there are four agent types: adversary, operator/defender, user, and media. The operator's goal is to provide a service to the users in exchange for a fee. The operator must maintain a Web-accessible account for every user using the system. Each user has the goal of using the service (and consequently the account) with minimal effort; if the effort of using the service becomes too great, the user will switch to a different service. The adversary wants to obtain unauthorized access to as many accounts as possible. The last agent type, the media, will publicize a successful hack if it learns about it. The model contains one adversary instance, one defender instance, one media instance, and two hundred user instances. We model the two hundred users as two hundred copies of the user submodel with different state initializations.

The model can be used to help a security practitioner choose among different defensive strategies by comparing their effectiveness across a variety of metrics. Specifically, we show how it may be used to compare three different operator defensive policies (passive, aggressive, and balanced) with respect to net defender profit, the time to discover the initial breach in security, and the number of accounts compromised in the attack. The metrics calculated by the model would help the defender choose a policy to follow in an implemented system.

4.1 Composed Model

The composed model consists of submodel instances of four submodel types, one for each of the four agent types in the model. There are two hundred instances of the user submodel type, and one instance of each other submodel type. Each submodel instance shows a particular agent's view of the environment (expressed in the Action Execution Graph) and the decision algorithm the agent uses to choose an action at each stage of the simulation.

There are twelve state variables in this model, as follows.

- **Account Access:** The status of a user's account.
- **Noisiness:** The cumulated evidence the attacker left (observable by the defender) as a result of the actions the attacker took.
- **Password Complexity:** A user's password strength.
- **Password Reset Requested:** This Boolean-valued state variable indicates whether the defender has requested that a user change an account password. In effect, it serves as a communication channel that the defender utilizes to convey a request for a particular action.

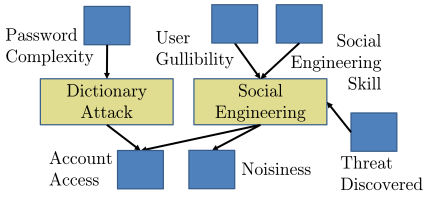

Fig. 1. A graphical representation of the adversary's Action Execution Graph.

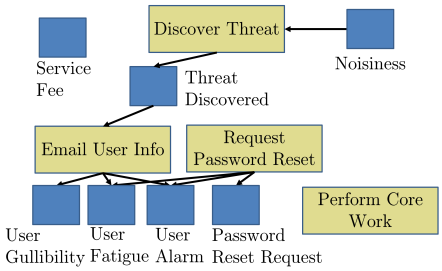

Fig. 2. A graphical representation of the defender's Action Execution Graph.

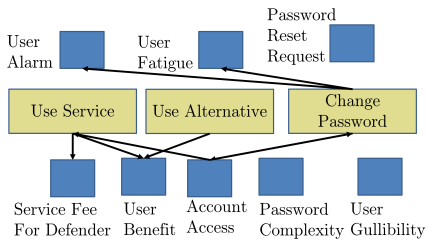

Fig. 3. A graphical representation of a user's Action Execution Graph.

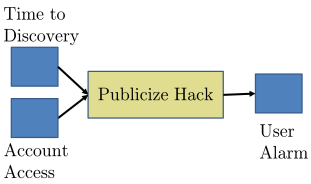

Fig. 4. A graphical representation of the media's Action Execution Graph.

- **Social Engineering Skill:** The adversary's level of skill in this attack.
- **Service Fee for Defender:** Payment made by the user for account access.
- **Threat Discovered:** Defender's knowledge of an attempted attack.
- **Time to Discovery:** Days until the media learn of a successful attack.
- **User Alarm:** User's fear of loss due to an account compromise.
- **User Benefit:** The value of the benefit the user accrues.
- **User Fatigue:** The user's level of fatigue from using the defender's service.
- **User Gullibility:** Susceptibility of a user to social engineering.

In the model, the *Noisiness*, *Password Reset Requested*, *Social Engineering Skill*, *Service Fee for Defender*, *Threat Discovered*, and *Time to Discovery* state variables each have a single value. The *Account Access*, *Password Complexity*, *User Alarm*, *User Benefit*, *User Fatigue*, and *User Gullibility* state variables have a sequence (or array) of values, with one value for each of the two hundred users in the model. The user submodels form an ordered list, such that the i^{th} user can only read from or write to the i^{th} value of those state variables that have a sequence of values. (Note that each of those state variables is included in the users' Action Execution Graphs.) That allows us to use one state variable, for example, to hold account status information for every individual user in the system, rather than two hundred individual copies of a state variable.

Every value of every state variable is initialized to zero, with three exceptions. The first exception, *Social Engineering Skill*, is initialized to 7/10 at the

beginning of the simulation. This indicates that the adversary has above-average skill in social engineering. The second and third exceptions are *Password Complexity* and *User Gullibility*. Some users have more cybersecurity knowledge, and motivation to act on that knowledge, than others; in this model, we have 80 sophisticated users and 120 average users, and these user types are reflected in the values assigned to these two state variables. We randomly selected 80 indices from 200 users to represent sophisticated users, and for each index in this set of 80 the corresponding values of *Password Complexity* and *User Gullibility* are set to 8/10 and 2/10, respectively. That indicates that the sophisticated users have strong passwords and are relatively unlikely to be susceptible to social engineering attacks. The remaining 120 values of *Password Complexity* and *User Gullibility* are initialized to 4/10 and 8/10, respectively.

4.2 Adversary Submodel

This submodel contains the attacker's Action Execution Graph (visually represented in Figure 1), and the agent decision algorithm.

Action Execution Graph: The attacker may choose either (1) to perform a dictionary attack on the database in an attempt to discover passwords via the *Dictionary Attack* action, or (2) to conduct a social engineering attack to trick users into revealing their passwords via the *Social Engineering Attack* action. These actions, along with their precondition and postcondition state variables, form the adversary's Action Execution Graph. The probability of a successful social engineering attack depends on the user's gullibility, the adversary's skill in social engineering, and whether or not the defender discovers the attack. If the attack is attempted, there is some probability that it will generate a small amount of noise. If successful, the attack will give the adversary access to the account. The probability of a successful dictionary attack depends directly on the complexity of the user's password.

Decision Algorithm: The attacker has two actions to choose from: (1) starting a dictionary-based brute-force attack to obtain the password, or (2) attempting a social engineering attack to trick users into giving the attacker access to their passwords. The adversary will attempt a dictionary attack on a password if (1) the minimum value in *Password Complexity* is less than 5, (2) *Social Engineering Skill* is less than 5, and (3) *User Gullibility* is greater than 5. If these conditions have not been satisfied, the adversary will choose to perform a social engineering attack.

4.3 Defender Submodel

This submodel contains the defender/operator's view of the overall model's state and the actions that the agent could take to change the state. For a visual depiction of the defender's Action Execution Graph, see Figure 2.

Defender Action Execution Graph: The defender can choose from four actions. The defender has the option of attempting to discover the adversary's activities (the *Discover Threat* action), sending an email to inform users of relevant threats or educate them about cybersecurity best practices (the *Email User Info* action), requesting that the user perform a password reset (the *Request Password Reset* action, or working on core business activities (the *Perform Core Work* action).

First, if a threat has not already been discovered, the defender can try to discover the adversary's attack. The probability of successfully uncovering the attack depends on the noisiness of the attack (the value of the *Noisiness* state variable). If the defender is successful, the attack is discovered (as indicated by *Threat Discovered*). Second, the defender can choose to send an email to the users with the *Email User Info* action to reduce their susceptibility to social engineering attacks. The email's effectiveness increases if the threat has been discovered. If *Threat Discovered* indicates that the threat has been discovered, the action will greatly reduce the user's gullibility; otherwise, the action will reduce it only slightly. Sending the email also slightly increases every user's fatigue, and, if the threat has been discovered, alarm. Third, the defender can also use the *Request Password Reset*, which sets the *Password Reset Requested* state variable to true. It also greatly increases the users' fatigue (because they have to create and memorize new passwords) and slightly increases the users' alarm (as tracked by the *User Fatigue* and *User Alarm* state variables, respectively). Fourth, and finally, at any time, the defender can attempt the *Perform Core Work* action. This action represents work that the operator can do that isn't directly related to cybersecurity. It exists as a sort of placeholder, because in reality, the operator is likely to spend most of his or her time on tasks that are unrelated to cybersecurity.

Decision Algorithm: We evaluate three different policies that a defender could choose to employ: aggressive, passive, and balanced.

First, the *aggressive policy* calls for defenders to take actions frequently to defend their networks and keep their users educated. Specifically, a defender will (1) email the users every ninety days to educate them about cyber security and how to avoid social engineering attacks, (2) request that the users reset the account password every ninety days (and whenever an ongoing attack has been detected), (3) attempt to discover threats by thoroughly analyzing security logs (from tools such as intrusion detection systems) every seven days, and (4) do work unrelated to cybersecurity (such as maintaining infrastructure, providing services to account users, etc.) on the remaining days.

If the defender employs the *passive policy*, he or she will take defensive actions infrequently. Specifically, the defender will send out a password reset request and an email educating users once every year and do work unrelated to cybersecurity on the remaining days.

Finally, the *balanced policy* can be used by a defender to strike a balance between the aggressive and passive approaches. The policy calls for the defender

(1) to send an email educating users every ninety days about how to avoid falling for social engineering attacks, (2) to request a password reset every year (and whenever a threat has been discovered), and (3) to attempt to discover threats by examining security logs every two weeks.

4.4 User Submodel

As loyal customers, the users wish to keep using the defender's service, because it meets their needs more effectively than competitor services do. As a result, the customers are cooperative and will reset their passwords if the defender asks them to. However, the defender can take actions that annoy the user (e.g., sending out emails too often), or the media could alarm the users if a compromise is publicized. Either case could drive the users away from the defender's service.

Action Execution Graph: Each user's Action Execution Graph consists of three actions and seven state variables. A graphical representation can be found in Figure 3. The three actions that the user could choose are *Use Service*, *Use Alternative Service*, and *Change Password*. First, the *Use Service* action is essentially the default action for this agent. It is enabled as long as *Account Access* does not indicate that the user has abandoned the account. *Service Fee for Defender* and *User Benefit* are incremented (by values of one and three, respectively) when *Use Service* is attempted. Second, the *Use Alternative Service* action can be taken by a user who is frustrated with the defender's service. The action is always enabled. It has only one postcondition state variable, *User Benefit*, which is incremented by two when the action is attempted. Third, the main effect of the *Change Password* action is to remove the adversary's access to the account (if it had previously been gained) by changing the password. The action is enabled as long as the user hasn't abandoned his or her account. The *Password Reset Request* state variable may signal to the user that the defender believes it would be beneficial to change the password. *User Alarm* is incremented by a small amount when the user changes the password, but *User Fatigue* is incremented by a much larger amount, to model the difficulty of creating and remembering a new password.

User Decision Algorithm: The user's policy is expressed in the following list: (1) If the defender requests that the user reset the password and the user has not abandoned the account, reset the password. (2) Otherwise, if the *User Alarm* and *User Fatigue* state variables both have values of less than 9, and the account has not been abandoned, use the defender's service. (3) Otherwise, if the user's fatigue is greater than or equal to 9, but the user's alarm is less than 9, and the account has not been abandoned, with some low probability switch to a competitor's service, otherwise continue to use the defender's service. (4) Otherwise, if the user's alarm is greater than or equal to 9, and the account has not been abandoned, with some high probability switch to a competitor's service, otherwise, continue using the defender's service. (5) Otherwise, use a competitor's service.

4.5 Media Submodel

The media submodel is the simplest agent in the model. Its one goal is to publicize a successful widespread attack against the defender's service. A graphical representation of the media's Action Execution Graph is given in Figure 4. The media's Action Execution Graph consists of only a single action, the *Publicize Hack* action, along with the state variables *Time to Discovery*, *Account Access*, and *User Alarm*. If at least 10% of user accounts have been compromised, the action will become enabled. Once the action is taken, it will complete at the end of the time indicated by *Time to Discovery*. If at least 10% of user accounts remain compromised at the end of that time, the actions will set the value of every user's *User Alarm* state variable to the maximum value, which could trigger abandonment of accounts by users. The media agent has a straightforward decision algorithm since the media have only one choice of action. The media will publicize the attack, increasing the users' alarm, six months after the first account has been compromised, if the defender doesn't quickly contain the problem.

4.6 Reward Model: Defining Metrics

The model gains its usefulness by supplying relevant metrics that will help the system architects and policy designers make good security choices. The model calculates seven metrics that give insight into the modeled system.

To begin, we track the defender's profit. The defender gains revenue from every user each day that he or she uses the service. The total revenue gained at the end of a 2-year period can be found by merely observing the value of the *Service Fee for Defender* state variable at the end of a 2-year simulation. The defender incurs costs by performing defensive actions. Every time a user takes action during a simulation, the cost to perform that action is calculated and stored in a running counter. At the end of the simulation, the value of that counter can be observed to determine the total direct cost of a particular policy. The profit is revenue minus costs.

Also, three metrics track the state of the 200 accounts in the defender's care at the end of the simulation: the mean number of uncompromised active accounts, the mean number of compromised accounts, and the mean number of abandoned but uncompromised accounts. The first of the three metrics gives the number of actively used, secure accounts (accounts that the adversary cannot access and have not been abandoned). The second of the three metrics tracks the number of accounts that are compromised by the adversary at the end of the simulation (whether or not the account has been abandoned by its user). The last of the three metrics gives the number of uncompromised accounts that have been abandoned by their users (because of alarm or fatigue) at the end of the simulation. All three of these metrics are calculated by observing the value held by each of the 200 instances of the *Account Access* state variable.

The final metric gives the time from the first successful account compromise to the time the defender discovers the threat. The first time the adversary compromises an account, the current simulation time is recorded. Similarly, the first

time the *Discover Attack* action successfully completes, the current simulation time is noted. The metric is the difference between the two times.

4.7 Model Execution and Results

	Passive	**Balanced**	**Aggressive**
Defender Stats			
Revenue	93770 +/− 540	145800 +/− 0	144029 +/− 111
Costs	2200 +/− 0	28300 +/− 0	59610 +/− 20
Profit	91570 +/− 536	117500 +/− 0	84418 +/− 122
Days to Detect	Never Detected	17 +/− 2	12 +/− 1
Account info			
# Uncompromised	19 +/− 1	200 +/− 0	182 +/− 1
# Compromised	120 +/− 0	0 +/− 0	0 +/− 0
# Abandoned	61 +/− 1	0 +/− 0	18 +/− 1

Table 1. Simulation Results

The complete model may be executed to calculate the defined metrics, as described in Sections 3.2 and 3.3. Every action in this particular model takes one day to complete (except the *Publicize Hack* action, which takes 180 days to complete). For that reason, the simulation step size is one day. We simulate the agents' behaviors over two years of simulation time to obtain the relevant metrics. All results are simulated with a 95% confidence interval.

The results are presented in Table 1. They show that the balanced policy has a clear advantage over the other two policies. The defender will earn the highest average net profit and have the most uncompromised accounts at the end of the two-year period if the balanced policy is used. When compared to the balanced policy, the aggressive policy brings in a similar amount of revenue, but incurs much higher costs (due to the frequency of defensive actions taken with this policy), so the defender earns a significantly lower profit. The defender, using an aggressive policy, successfully thwarts the adversary's attempts to compromise accounts, but the defender's frequent actions annoy some users, driving them to abandon the service. The aggressive policy leads to slightly earlier threat detection than the balanced policy. The passive policy costs the defender much less than the aggressive or balanced policies. However, revenue isn't as high, so average net profit is also low. The abandonment of so many user accounts explains the low revenue. We investigated why so many more users abandoned their accounts here than for the other two policies. We found that the media never publicize the hack if the defender uses the aggressive or balanced policies (because not enough accounts are compromised for a long enough time to be newsworthy). However, because the passive defender does so little to counter the

attacker, many accounts are compromised, which triggers the media publication of the hack, which alarms users and drives them to abandon their accounts.

5 Conclusion

In this paper we argue that cyber security models must explicitly incorporate all relevant agents to provide an accurate view of the overall system behavior, and that the agents must be modeled in a realistic manner. Formalisms that only model adversary behavior or simple adversary-defender conflict behavior do not accurately reflect the reality found in cyber systems that are manipulated and used by many different human entities. To solve this problem, we propose a new, easy-to-use modeling framework, the General Agent Model for the Evaluation of Security. This framework allows the modeler to construct different agent submodels, which may be composed together and executed to calculate metrics that give insight into system behavior. Each submodel consists of the agent's view of the state, the actions available to the agent, and the agent's customizable decision algorithm. We demonstrated the richness of the formalism with an example, which incorporated a number of different agents with different goals and policies. The GAMES formalism is a significant step forward in the quest to give cyber security analysts the ability to create realistic security models of cyber systems.

References

1. Baiardi, F., Corò, F., Tonelli, F., Bertolini, A., Bertolotti, R., Guidi, L.: Security stress: Evaluating ict robustness through a monte carlo method. In: Panayiotou, C.G., Ellinas, G., Kyriakides, E., Polycarpou, M.M. (eds.) Critical Information Infrastructures Security. pp. 222–227. Springer Int. Publishing (2016)
2. Buratowski, M.: The DNC server breach: who did it and what does it mean? Network Security 2016(10), 5–7 (2016)
3. Eloff, J., Labuschagne, L., Badenhorst, K.: A comparative framework for risk analysis methods. Computers & Security 12(6), 597 – 603 (1993)
4. Gorodetski, V., Kotenko, I., Karsaev, O.: Multi-agent technologies for computer network security: Attack simulation, intrusion detection and intrusion detection learning. Int. Journal of Comp. Systems Sci. & Eng. 18(4), 191–200 (2003)
5. Grossklags, J., Christin, N., Chuang, J.: Secure or insure? a game-theoretic analysis of information security games. In: Proceedings of the 17th International Conference on World Wide Web. pp. 209–218. WWW '08 (2008)
6. Herley, C., v. Oorschot, P.C.: Sok: Science, security and the elusive goal of security as a scientific pursuit. In: 2017 IEEE Symp. on Security & Privacy (SP). pp. 99–120 (May 2017)
7. Izmalkov, S., Micali, S., Lepinski, M.: Rational secure computation and ideal mechanism design. In: 46th Annual IEEE Symp. on Foundations of Comp. Science (FOCS'05). pp. 585–594 (Oct 2005)
8. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack–defense trees. Journal of Logic and Computation 24(1), 55–87 (2012)
9. Kordy, B., Pièetre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. CoRR abs/1303.7397 (2013), http://arxiv.org/abs/1303.7397

10. Kramer, J.: Attack-Defence Graphs: On the Formalisation of Security-Critical Systems. Master's thesis, Saarland University (2015), https://www-old.cs.uni-paderborn.de/uploads/tx_sibibtex/main_01.pdf
11. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy 9(3), 49–51 (2011)
12. Lee, R.M., Assante, M.J., Conway, T.: Analysis of the cyber attack on the Ukrainian power grid. SANS Industrial Control Systems (2016)
13. LeMay, E.: Adversary-Driven State-Based System Security Evaluation. Ph.D. thesis, U. of I. at Urbana-Champaign (2011), https://www.perform.illinois.edu/Papers/USAN_papers/11LEM02.pdf
14. Manshaei, M.H., Zhu, Q., Alpcan, T., Bacşar, T., Hubaux, J.P.: Game theory meets network security and privacy. ACM Comput. Surv. 45(3), 25:1–25:39 (2013)
15. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1994)
16. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: Structure, behavior, and application. In: Proceedings of the International Conference on Timed Petri Nets. pp. 106–115. Torino, Italy (July 1985)
17. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: System Sciences (HICSS), 2010 43rd Hawaii International Conference on. pp. 1–10. IEEE (2010)
18. Salter, C., Saydjari, O.S., Schneier, B., Wallner, J.: Toward a secure system engineering methodolgy. In: Proceedings of the 1998 Workshop on New Security Paradigms. pp. 2–10. NSPW '98 (1998)
19. Sanders, W.H., Meyer, J.F.: Reduced base model construction methods for stochastic activity networks. IEEE Journal on Selected Areas in Communication, special issue on Computer-Aided Modeling, Analysis, & Design of Communication Networks 9(1), 25–36 (Jan 1991)
20. Sanders, W.H., Meyer, J.: A unified approach for specifying measures of performance, dependability, and performability. In: Avizienis, A., Kopetz, H., Laprie, J. (eds.) Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems. pp. 215–237. Springer-Verlag (1991)
21. Straub, D.W., Welke, R.J.: Coping with systems risk: security planning models for management decision making. MIS quarterly pp. 441–469 (1998)
22. Verendel, V.: Quantified security is a weak hypothesis: A critical survey of results and assumptions. In: Proceedings of the 2009 Workshop on New Security Paradigms Workshop. pp. 37–50. NSPW '09 (2009)
23. Wagner, N., Lippmann, R., Winterrose, M., Riordan, J., Yu, T., Streilein, W.W.: Agent-based simulation for assessing network security risk due to unauthorized hardware. In: Proceedings of the Symposium on Agent-Directed Simulation. pp. 18–26. ADS '15, Society for Computer Simulation International, San Diego, CA, USA (2015)
24. You, X.Z., Shiyong, Z.: A kind of network security behavior model based on game theory. In: Parallel and Distributed Computing, Applications and Technologies. PDCAT'2003. Proc. of the 4th International Conf. on. pp. 950–954 (Aug 2003)