ADDRESSING CHALLENGES TO QUANTITATIVE SECURITY MODELING

BY

MICHAEL RAUSCH

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor William Sanders, Chair
Professor Klara Nahrstedt
Professor Carl Gunter
Dr. Carol Muehrcke

ABSTRACT

Quantitative state-based models can help those responsible for designing, maintaining, or insuring cyber systems make informed decisions. However, there are a number of difficulties that discourage the use of quantitative cybersecurity models in practice. We identify four significant challenges to quantitative security modeling, (1) cybersecurity models are difficult to build by hand, particularly for system architects that are not experts in cybersecurity, (2) it is challenging to model the complex interplay between the cyber system and the many human entities that interact with it with current modeling formalisms, (3) the uncertainty that comes from the model's input variables should be managed and explored with sensitivity analysis (SA) and uncertainty quantification (UQ), but many models run too slowly to complete traditional SA and UQ analyses, and (4) there is a lack of appropriate frameworks, guidance on metrics, and advice on common modeling issues with regards to quantitative cybersecurity models.

In this dissertation, we address each of the four challenges. To address the first challenge, we present an ontology-assisted automatic cybersecurity model generation approach that modelers can use to make cybersecurity models quickly and easily. Using this approach, a system architect would first create a system diagram of the components of the system and their relationships to one another. Then, a model generation algorithm would convert the system diagram (with the aid of an ontology) into a sophisticated cybersecurity model that can be executed to obtain metrics. We implemented the tool in Möbius and demonstrated its use with an AMI test case. To address the second challenge, we designed a new agent-based modeling formalism called GAMES that allows the modeler to explicitly model the system and all of the human entities that interact with the system in a modular and intuitive fashion, and show its strengths with a worked example. To address the third challenge, we proposed an indirect stacking-based metamodeling approach. Using the metamodeling approach, we are able to accomplish sensitivity analysis and uncertainty quantification hundreds to thousands of times faster than traditional approaches and with better accuracy than current metamodel approaches. We demonstrate the approach's efficacy with eight worked

examples. Finally, to address the fourth challenge, we present a high-level framework to guide the modeling process, give guidance on what metrics to calculate and how to calculate them, and share advice on common issues with cybersecurity modeling.

The theoretical and practical contributions presented in this dissertation will help make quantitative cybersecurity modeling easier to use and more useful, which will, in turn, help protect society's most critical and valuable infrastructure from cyber threats.

To my parents, my first and greatest teachers.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor William H. Sanders, for guiding me and supporting me on my academic journey over the past seven years. I sincerely appreciate the opportunity he gave me to do research in the Performability Engineering Research Group (PERFORM) and to study at Illinois. He is, in my opinion, the greatest champion of quantitative cybersecurity modeling in academia, and I was honored to have him as my advisor. I am also grateful to Professor Klara Nahrstedt, Professor Carl Gunter, and Dr. Carol Muehrcke for their valuable advice, and for serving on my committee.

I would like to thank Ken Keefe for helping me to learn the Möbius framework and the ADVISE modeling formalism, and for his leadership on the ontology project. I would also like to thank Dr. Brett Feddersen for the engaging discussions, encouragement and technical advice. I also greatly appreciate the editorial assistance given by Jenny Applequist, who greatly improved the quality of my writing.

I am grateful for the advice and friendship of the members of the PERFORM group: Varun Badrinath Krishna, Atul Bohara, Carmen Cheh, Ahmed Fawaz, David Huang, Mohammad Noureddine, Uttam Thakore, Benjamin Ujcich and Ronald Wright. I have learned a great deal on how to do high-quality research from the example of my fellow researchers in PERFORM.

I am so grateful for my loving family, especially my wonderful parents, Lowell and Katie. Thank you for always being there for me. Thank you also to my friends who enriched my graduate school years, particularly those I met at the Lincoln Green Foundation.

PUBLICATIONS

- Chapter 2 contains material previously published in [1] and [2]. The work was done in collaboration with Ken Keefe, Brett Feddersen, Carol Muehrcke, Ronald Wright, and under the supervision of Dr. William H. Sanders.

- Chapter 3 contains material previously published in [3]. The work was done in collab-

oration with Ahmed Fawaz, Ken Keefe, and under the supervision of Dr. William H. Sanders.

- Chapter 4 contains material previously published in [4], [5], and [6]. The work was done under the supervision of Dr. William H. Sanders.

The material in [1], [2], [3], [4], and [6] was reused with permission from Springer. The material in [5] was reused with permission from IEEE.

FUNDING AND DISCLAIMERS

The work described here was performed, in part, with funding from the Department of Homeland Security under contract HSHQDC-13-C-B0014, "Practical Metrics for Enterprise Security Engineering." Some material in this dissertation is based upon work supported by the Maryland Procurement Office under Contract No. H98230-18-D-0007. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the Department of Homeland Security or the Maryland Procurement Office.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Modern societies rely on functioning cyber systems, and will likely grow more reliant on them in the future. Every field of modern society has been touched by cyber systems, including education, communication, health care, industry, critical infrastructure, and defense. There is every reason to believe that cyber systems, from the largest cloud to the tiniest IoT device, will continue to improve and become better integrated in different facets of our modern life. These systems are incredibly valuable and will only become more valuable in the future. Their value makes them attractive targets to a variety of malicious actors, from skilled teams of hackers from huge nation states to small terrorist cells to individual disgruntled employees.

There are many examples that highlight both the importance of cyber infrastructure and its vulnerability. In 2015 a cyber attack on Ukraine's electric grid disrupted power for over two hundred thousand residents [7], the successful hack and exfiltration of emails belonging to the U.S. Democratic National Committee influenced the contentious U.S. Presidential race of 2016 [8], the Stuxnet worm delayed Iran's nuclear program [9], the 2017 Equifax breach led to the loss of private and very sensitive data for over one hundred forty million people [10], the Mirai botnet compromised hundreds of thousands of IoT devices and used them to launch distributed denial of service attacks that caused widespread disruptions, making it difficult to access popular sites such as Amazon, Netflix, and Twitter, among others [11], and the SolarWinds hack compromised the U.S. Department of Energy (responsible for managing the U.S. nuclear weapons program), the Pentagon, the U.S. Cybersecurity and Infrastructure Security Agency, and major U.S. companies such as Microsoft, Intel, and Cisco [12]. Many other cyber incidents are not as widely reported but still cumulatively have a significant societal and economic impact. Overall, the U.S. government's Council of Economic Advisors estimates that the United States lost between $57 billion and $109 billion due to malicious cyber activity in 2016 alone [13]. As a consequence, cyber systems should be designed, maintained, and insured with security in mind.

1.2 WHY DO QUANTITATIVE SECURITY MODELING?

Modeling can be difficult, time consuming, and expensive. One may question why quantitative modeling should be done in the cybersecurity domain. We identify three major uses cases: designing systems that are secure, maintaining systems so they stay secure, and determining how to insure systems to effectively manage risks. We shall consider each of these three use cases in turn. In this dissertation, we focus mainly on the design use case, though most of the methods, techniques, and arguments can be applied in a straightforward manner to the other two use cases.

1.2.1 Quantitative Modeling for Designing Secure Cyber Systems

Quantitative models can assist a system architect in making good design decisions. Every time the architect is faced with a choice when designing a system, the choice implicitly contains multiple design options: one design for each unique combination of options that could be chosen for the system. To determine the best choice (or, at least, a satisfactory choice) the system architect must consider a set of the possible designs and either rank them by some criteria, or determine whether they will satisfy some minimum efficacy threshold. When considering a particular design, the system architect always first creates a model and then examines it in an attempt to forecast how the particular design may function in the future if a real-world system is constructed using the design. The model could be an informal mental model, or a more formal model. The forecast obtained from the model must consist of information that supports the ranking of the design choices or the determination of whether it surpasses the minimum efficacy threshold, so that an appropriate selection may be made.

An informal mental model may be enough for a sophisticated system architect to make many decisions (particularly if the system is simple and small), but there are many drawbacks to relying exclusively on mental models. First, as the systems become larger, more heterogeneous, more tightly interconnected, and more complicated, it often becomes increasingly difficult to create and maintain a sufficiently accurate mental model. In addition, these large, complicated systems are usually more expensive to build and more valuable than smaller, simpler systems. It is therefore often the case that the negative consequences of forming an

inadequate mental model grows with the difficulty of forming an adequate mental model. Creating formal, documented, falsifiable models also moves the field towards the *Science of Security* paradigm, and all the benefits that the scientific method can provide [14]. A formal quantitative model much more easily facilitates collaboration among subject matter experts than informal mental models contained in the mind of a coordinator or manager. For these and other reasons it is advisable for the system architect to pursue formal quantitative cybersecurity modeling as a design aid.

1.2.2 Maintaining Secure Systems

Cyber infrastructure is under the constant threat of attack after it is constructed. System operators must be prepared to handle very frequent and ongoing attacks. Adversaries constantly probe the system defenses, looking for a weak link. Often adversaries make their way into the system, and the operators/defenders of the system must try to get the system to fulfill its mission objectives despite the presence of the adversary in the system (as in the case of *advanced persistent threats*, commonly abbreviated as APTs).

The operators/defenders of the system must make complex decisions very quickly when the system is under attack. It can be difficult to wargame different scenarios and understand the consequences of different possible actions in this stressful environment. Prebuilt quantitative models can help defenders make better decisions regarding defensive actions quickly. Such models can also help to record the knowledge of the primary system architects and defenders, so if they leave the company or are transfered to a different project, or are otherwise unreachable during an attack, others may use the models that codify their knowledge as a decision aid to help them develop appropriate responses while under attack.

1.2.3 Insurance

A major cyber breach can be financially devastating to a company. Depending on the circumstances, a company can lose revenue if the availability or integrity of their systems is compromised, and may face fines from regulatory agencies. A successful cyber attack can also negatively impact the reputation of the company, making it less likely that people

will do business with the company in the future. The probability of an individual company suffering a major successful cyber attack is relatively small, but the consequences are large. Companies are increasingly turning to cyber security insurance to help them manage cyber risk [16] [17]. The cyber insurance market generates $5 billion dollars of premiums per year, and is expected to continue to grow in the future [18].

Cyber security insurers need to form a quantitative model of the company to determine an appropriate premium. The goal of the for-profit insurance company is to ensure that the premium is greater than the customer's expected insured loss. If the premium is higher than that threshold the insurance company can expect to make a profit, otherwise, the insurance company will lose money. The argument makes clear the need for insurance companies to make good estimates of the expected loss of the company. It is reasonable to assume that an insurance company that understands the client, the cyber threats the client faces (e.g. the vulnerabilities in the system along with the potential adversaries, their motivations, and capabilities), and the defenses of the client's cyber infrastructure, and how these interact, will be able to offer better premiums than their competitors (either by offering a lesser premium to win the client from competitors, or by refusing to offer a premium that will likely be unprofitable that they otherwise would have offered). Formal quantitative modeling could give a competitive edge to cyber security insurance companies. Of course, potential clients of cyber security insurance companies also need to form some sort of quantitative risk assessment model to determine whether they should obtain the insurance given a particular premium. Both buyers and sellers of insurance could benefit from quantitative cybersecurity modeling.

1.3 OVERVIEW: PRINCIPLES, CHALLENGES, CONTRIBUTIONS

Many now recognize that security must be a design consideration from the beginning of the design process. There was once a tendency to design cyber systems in such a way that their performance would be maximized in an environment free of adversaries, and then "security" would be added at the end of the design process, or once the system was already built. However, adding "security" so late often resulted in a poor fit that left

the system insecure or needlessly compromised the system's performance. Instead, system architects should use models, like the Computer Aided Design (CAD) programs used in other engineering disciplines, to design secure systems. Though system architects may desire to use models to help them incorporate security considerations into the design process, in practice it appears that system architects tend to rely on experience, intuition, and tradition instead. We believe system architects are faced with a number of challenges that make modeling cyber system security a particularly onerous task compared to other disciplines that use quantitative modeling.

We believe that cyber security modeling must satisfy four requirements to be widely adopted.

1. It should be usable without special modeling expertise.
2. It should model the correct things.
3. It should be able to validate the model results.
4. It should follow appropriate frameworks, be guided by a sensible philosophy of metrics, and be aided by advice on overcoming common modeling issues.

Cybersecurity modeling is unlikely to become popular and widely used if it cannot satisfy these four requirements. Each of the four requirements, however, is paired with a corresponding challenge that, if left unresolved, reduces the practicality of cyber security modeling and discourages its use.

1. The first challenge is to construct cybersecurity models despite the fact that most cybersecurity domain experts are not expert modelers. Currently, many security modeling formalisms require a near-expert understanding of some combination of game theory, probability (to understand the assumptions and results), understanding of historical cyber attacks, the ability to forecast future cyber attacks and attackers, and general modeling principles, in addition to a deep understanding of the intended workings of the system to be modeled. It is uncommon for system architects to have such a deep understanding in these disparate fields.

2. The second challenge is to correctly model the complex interplay between the cyber system and the humans that interact with it. Most current security modeling formalisms only consider the adversary's perspective, or model scenarios as a game between the adversary and the defender. In reality, however, things are not so simple. Often other human parties have a huge influence on the security of the system. These parties include customers, third-party vendors, users, law enforcement, the media, and others. How should all of these disparate entities be modeled?

3. The third challenge is model validation given given uncertain input variables and long model execution times which make traditional sensitivity analysis and uncertainty quantification techniques unfeasible in many cases. It is clear to anyone who has tried to do cybersecurity modeling that there is often significant uncertainty about the values of a number of the input variables. Many factors contribute to this uncertainty, including a lack of knowledge of the identity, motivations, and abilities of potential attackers, and an incomplete understanding of how the intricate cyber system will actually behave and evolve over time in real-world conditions. The traditional way to help explore and manage this uncertainty is through sensitivity analysis (SA) and uncertainty quantification (UQ). However, traditional SA and UQ techniques require running the model many times, which is often unfeasible due to long model run times. How should modelers overcome this issue?

4. The fourth challenge is the lack of helpful modeling frameworks, guidance regarding appropriate security metrics, and advice on handling common modeling problems.

These, of course, are not the only challenges to cybersecurity modeling, but we believe that they are among the most pressing. Our contributions address each of the four challenges to the four requirements enumerated above. Taken together, our contributions point a clear path towards an end-to-end approach to cybersecurity modeling that is (1) relatively easy for those without modeling expertise to accomplish, (2) models humans appropriately, (3) produces results that can be validated and explored, and (4) follows appropriate frameworks, guidance regarding metrics, and is aided by advice on common modeling issues. The following summarize the contributions we have made to addressing the challenges.

1. To address the first challenge, we present a novel ontology-driven method to automatically generate complex security models from relatively simple system diagrams that can be easily constructed by system architects.

2. To address the second challenge, we developed the novel GAMES modeling formalism, which allows modelers to intuitively model *all* of the human agents in the cyber system, whether adversaries, defenders, users, third-party vendors, customers, or others. The formalism uses composable submodels for each agent which can then be unified into a comprehensive executable simulation model.

3. To address the third challenge, we created a novel metamodel-based approach to do sensitivity analysis (SA) and uncertainty quantification (UQ) indirectly. While the idea of using metamodels to indirectly perform SA and UQ is not new, it had not been applied to cybersecurity models before, and the ML process we used (called stacking) has, to the best of our knowledge, never been applied to the construction of metamodels before. Using the metamodel approach, we can perform SA and UQ thousands of times faster than using traditional methods, and with more accuracy than other metamodeling approaches.

4. To address the fourth challenge, we developed a high-level framework to guide the modeling process, give guidance on the philosophy, purpose, definition, and creation of cybersecurity metrics, and give advice on common cybersecurity modeling issues.

1.4 DISSERTATION ORGANIZATION

The following four chapters cover the contributions to addressing the four challenges, respectively. In Chapter 2, we explain our approach to ontology-assisted automatic model generation from high level system diagrams that dramatically reduces the modeling burden away from system architects. In Chapter 3, we define the GAMES agent-based modeling formalism, which allows for construction of modular agent submodels that can be composed to study the complex interactions of the various human entities (adversaries, defenders, customers, regulatory and law enforcement agencies, etc.) with one another and the cyber

system under investigation. We demonstrate its practicality with a worked example. In Chapter 4, we show how an indirect stacking-based metamodel approach can help a modeler perform sensitivity analysis and uncertainty quantification on complex slow-running cybersecurity models much faster than is possible with traditional direct methods, and with greater accuracy than other indirect metamodeling methods. We show the effectiveness of the approach by analyzing two cybersecurity models as test cases (and further show the generalizability of the approach with an analysis of 6 other quantitative models). Chapter 5 contains a framework we have developed to help guide modelers through the modeling process, a discussion on metrics, including an argument for utility as the key cybersecurity metric, and advice on common modeling issues. We conclude in Chapter 6 with a discussion on how to continue pushing the field of cybersecurity modeling forward.

CHAPTER 2: AUTOMATIC ONOTOLOGY-AIDED MODEL GENERATION

2.1 INTRODUCTION

Architects of new cyber-physical infrastructure must make many decisions when designing a system that will impact its overall security. There are two major approaches for evaluating the impact of different design choices on system security: (1) consultation with security experts, and (2) construction and analysis of rigorous security models.

The two approaches may be used jointly and complement one another, though both suffer limitations. Security experts may make assumptions that are not explicit, and thus their decisions are not easily auditable; further, experts often rely on experience and intuition to make decisions, rather than on rigorous scientific approaches. From a practical perspective, there is also a significant shortage in the number of qualified security experts, with some estimating that more than 200,000 cybersecurity-related positions are unfilled [19]. On the other hand, security models make assumptions explicit and are easily auditable and scientifically rigorous, but they are also difficult and time-consuming to construct, especially for domain experts who do not have a background in modeling or security.

We propose a novel approach that allows those with almost no security modeling background to construct simple and intuitive system models that may be used to automatically generate relevant, rigorous security models to support the evaluation of a system. Following this approach, a user would (1) use a tool with an intuitive graphical user interface to develop a system model, (2) select the types of adversaries that should be considered in the analysis, and (3) choose the metrics the security model should compute. Next, the complete user-defined model would be given to a model generator, which would leverage a predefined ontology to construct a relevant ADversary VIew Security Evaluation (ADVISE) model [20]. The security model would then be executed to calculate the metrics, and the results would be presented to the user to aid in the evaluation of designs. We have implemented this approach in the Möbius tool [21].

We explain the approach through a case study of a hypothetical utility that has an *Advanced Metering Infrastructure* (AMI) deployment and seeks to select the most cost-effective

intrusion detection system (IDS) approach for monitoring and defense. An AMI is a cyber-enhanced power grid that gives a utility a more fine-grained ability to observe its network, and more remote control over the network components. Different IDS deployment approaches, including both centralized and distributed ones, may be used to defend an AMI. We show how a system architect can easily and automatically create security models to calculate security estimates to be used to inform the design choices among the IDS options.

2.2 APPROACH

We present a novel approach to automatically generate security models from system models, which we have implemented in the Möbius modeling tool. At a high level, a modeler will use component types defined in an ontology to construct a simple system model, choose adversaries of concern, and select security metrics to be calculated. The user-defined system model is used as input to the model generation algorithm. This algorithm relies on the ontology to provide mappings between system model elements and security model elements. The algorithm will output a security model, which may be executed to obtain the user-defined security metrics.

Our approach for automatically generating security models is novel, though the concept has been considered previously. Groundbreaking work in this area was presented in [22], though that approach was found to scale poorly. An approach for scalable security model generation is given in [23], but it is intended for analysis of existing systems and does not support analysis of systems in the design stage, in contrast to the approach described in this chapter. The approach shown in [24] is perhaps closest to our method, though that method does not use an ontology and it is not immediately clear how it could be extended to support state-based security models, like ADVISE.

2.2.1 Ontology

The ontology is the key to the model generation approach. All of the component types and relationships between them, in both the system model and the security model, are drawn from the ontology.

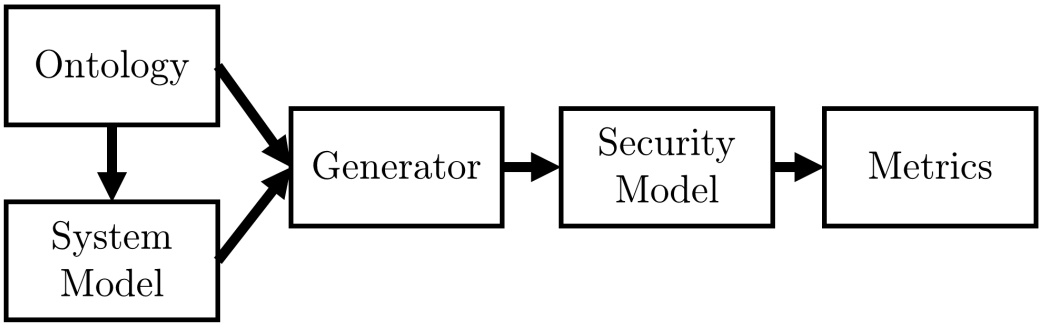


Figure 2.1: An end-to-end illustration of the approach to calculating security metrics from security models automatically generated from an ontology and a hand-built system model.

Formally, the ontology can be defined as the tuple $< C, R, F, M, I, A, S >$, where:

- C is the set of all possible system model component types.
- R is the set of all possible pairwise relationships, $r(C_d, C_r)$, from components of types from C_d to components of types from C_r, with $C_d, C_r \subset C$.
- F is the set of security model fragments, $fragment_x \in F$. Since the generated security models conform to the ADVISE formalism, the model fragments may include attacks along with access, knowledge, skill, and system state variables, and the precondition/effect relationships between them. For more on ADVISE, see Section 2.2.4.
- M is the set of mappings that can be made based on a component's type to infer a security model fragment, $m(c_u, fragment_x) \in M$. For example, if there is a physical device in the system and the adversary gains physical access to it, he or she may try to damage it. This is expressed as
  $fragment_1 = \{pAttck, pAccss, dmg, precond(pAttck, pAccss), effect(pAttck, dmg)\}$
  $\in F$,
  $physicalDevice \in C$ and $m(physicalDevice, fragment_1)$.
- I is the *inheritance* relationship between $c_u, c_v \in C$. If $i(c_u, c_v) \in I$, then we say that component type c_u *inherits* from c_v, i.e., c_u is a specialized type of c_v. For example, if the *smartmeter* component type inherits from the *meter* type, $i(smartmeter, meter) \in$

I. Inheritance is transitive and multiple inheritance is supported. Attributes and mappings are inherited.

- A is the set of all adversary profiles.
- S is the set of security metrics to be calculated.

The ontology should be constructed by experts who have experience with the system domain, modeling, and security; the typical user will not have to learn the details of the ontology or modify it. Once defined, the ontology may be used to construct models of many different systems and system configurations.

2.2.2 General System Model

To begin, a user builds a model of the system using a graphical user interface tool. The model resembles a UML-style diagram. The user drags and drops predefined system component types from a menu onto a canvas to define instances of the component type, and then forms connections between the components to describe the relationships between the components. Smart meters, DCUs, servers, and networks are examples of components, while *NetworkConnection* is an example of a relationship. Components can contain attributes, which may be editable by the user. At this stage, the user also selects the types of adversaries of concern, and customizes the base adversary profiles. Finally, the user selects the security metrics of interest that should be calculated by the security model (e.g., damage to the system as the result of an attack, or the probability of detecting the adversary).

2.2.3 Generator

The purpose of the generator is to construct a security model, which may be executed to calculate security-relevant metrics from the user-defined system model along with the corresponding ontology. At its core, the generation algorithm is simple. First, the type of each instance defined in the system model is determined, and then the mapping function is applied to find the corresponding security model fragments.

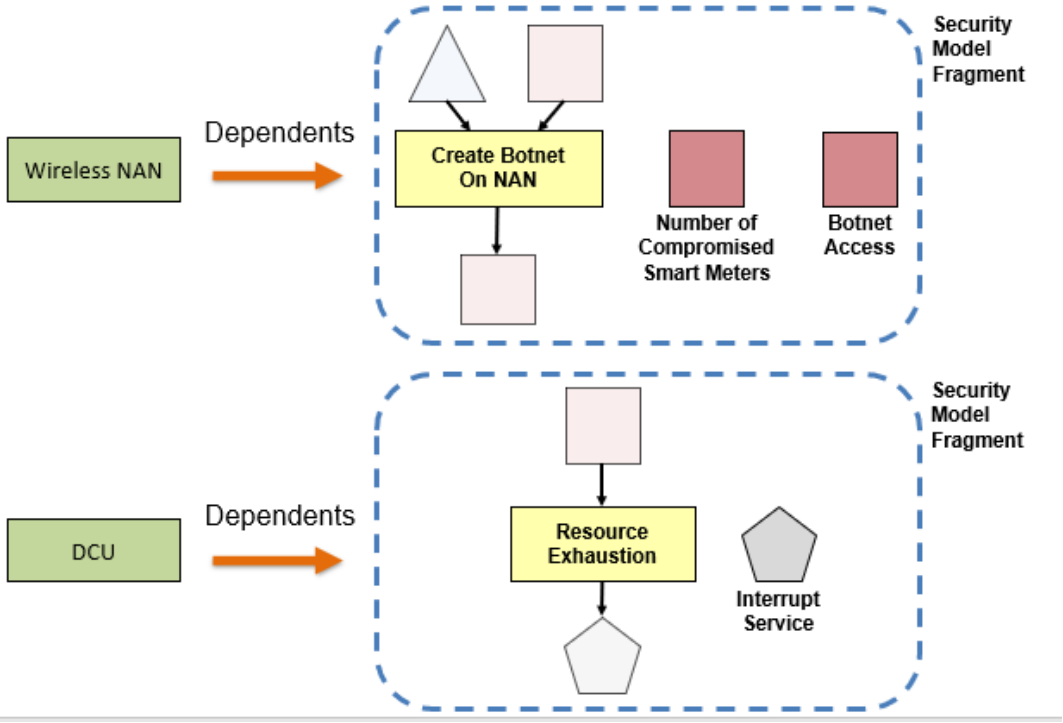


Figure 2.2: An illustration of system diagram components and their corresponding security model fragments.

Next, once found, the security model fragments are stitched together to form a whole, and the model elements are pruned if doing so does not affect the results obtained by executing the model. During fragment stitching, each state variable in a model fragment is compared to state variables in other fragments. If it is found that the state variables are equivalent, they may be combined, joining the fragments. As an example, consider two fragments, f_1 and f_2. Now, f_1 consists of an attack $InstallWirelessJammerOnNAN1$ that has an effect on a state variable called $WirelessJammerAccessOnNAN1$, while f_2 consists of an attack $JamWirelessSignalOnNAN1$ with a precondition state variable called $WirelessJammerAccessOnNAN1$. Both fragments contain that state variable, and so may be linked together to form an attack chain that expresses the idea that a wireless jammer must be installed before a wireless jamming attack can be accomplished. Similarly, security model elements (i.e., attack steps and state variables) may be pruned without affecting the model execution results if a path from the fragment to a goal does not exist. The pruning step can substantially reduce the computation needed to automatically generate the model. Future work should investigate other opportunities for pruning.

2.2.4 Security Model

We developed the generator and ontologies to support the creation of ADVISE security models, though in theory it should be possible to use this approach to generate security models in other modeling formalisms.

Each model in ADVISE consists of an *Attack Execution Graph* (AEG) and an *adversary profile*, which together form one atomic submodel in Möbius. The full resulting model may then be executed to obtain the security results. The formalism is explained briefly here; for more details, consult [25].

Formally, an AEG is defined by the tuple $< A, S, C >$, where A is a set of attack steps, S is a set of state variables, and the relation C defines the set of directed connecting arcs from $p \in S$ to $a \in A$, and from $a \in A$ to $e \in S$, where p and e are precondition and effect state variables, respectively.

An attack step $a \in A$ is defined by the tuple $< B, T, P, O >$, where B is a Boolean precondition that must be satisfied before the adversary may attempt the attack, T is the time to complete an attack, P is the cost the adversary incurs for attempting the attack, and O is the set of outcomes that may occur if the attack is attempted. Each outcome has a probability of occurrence, and, if selected, a specified effect on the system state.

The attacker model describes the initial state of the attack execution graph (the starting value of every state variable), as well as some further adversary characteristics, such as the payoff the adversary receives for achieving a goal, the cost the adversary incurs if it is detected, and the adversary's ability to forecast the future (modeled by defining an upper limit on the longest chain of attack steps the adversary may consider when forming attack plans).

The model is executed once the AEG and adversary profile have been defined. During model execution, the adversary first uses a competitive Markov decision process to select the optimal attack, given the system state, that will maximize the accumulated net profit (as described in [26]); then, one of the attack's outcomes is randomly selected according to its probability of occurrence, and its effect is applied to the model state. This process repeats until the end of the simulation, at which time the results are presented to the user. These results may be used to inform security-related decisions regarding system design.

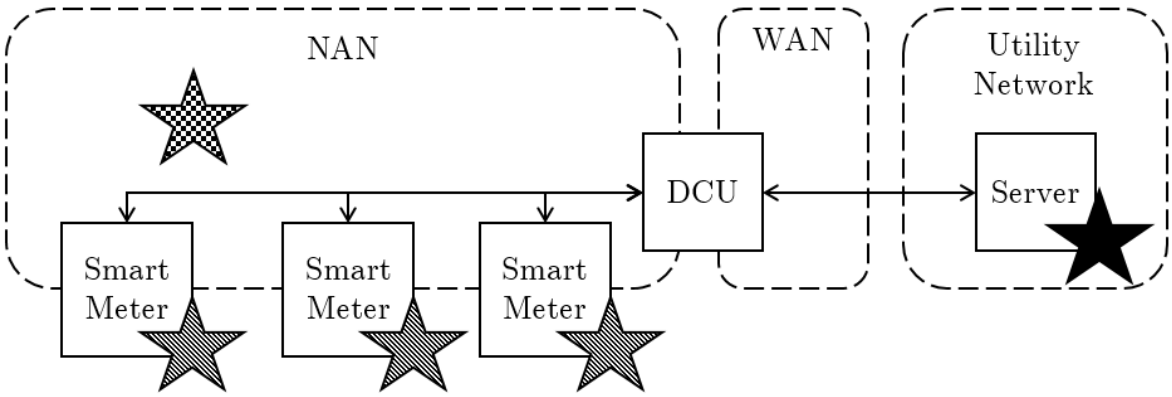


Figure 2.3: The architecture of the AMI deployment under consideration by the hypothetical utility. The placement of each IDS is shown by a star. The solid black star indicates the location of the centralized IDS. The locations of the dedicated device IDS and embedded IDS approaches are denoted by checkered and striped stars, respectively.

2.3 DEMONSTRATION OF APPROACH

Utilities are increasingly upgrading the power grid by incorporating smart meters and other computerized components to form AMI deployments that enhance the ability to monitor and remotely control the grid. These upgrades will allow utilities to be more efficient and cost-effective; the Electric Power Research Institute estimates that utilities will gain billions of dollars in benefits by constructing smart grids [27], while the U.S. International Trade Commission cited market observers who forecasted that the global smart meter market would approximately quintuple between 2011 and 2018, from $4 billion to about $20 billion [28].

While many network topologies may be used, we shall limit our discussion in this chapter to one representative case. In this deployment, the smart meters are connected to one another and to a *data concentrator unit* (DCU) via a *neighborhood area network* (NAN), which is a wireless mesh network. The DCU collects readings from the smart meters and sends them via the *wide area network* (WAN) to a centralized server on the utility's back-end network, where they may be observed by network operators. The utility may also send control commands in the reverse direction to the smart meters or to other AMI components.

The AMI deployments are critical infrastructure that must be effectively protected to ensure that people continue to receive power. However, the increasing networking and com-

puterization of the power grid are potentially introducing new vulnerabilities [29] [30] [31]. Attacks that exploit the new vulnerabilities may include denial of service attacks, Byzantine attacks, and various kinds of routing attacks, among others [29]. In addition, the utility must handle the threat of new types of attacks that customers may employ to steal electricity from the utility [32] [30] [31]. A utility may choose to defend its AMI deployment by employing defenses, e.g., IDSes, commonly used to protect more traditional cyber networks.

In general, an intrusion detection system monitors and logs network traffic and processes on machines and alerts the network operators if it detects suspicious behavior. Different IDS deployment approaches will give different levels of benefit in terms of coverage and have different costs, and these must be carefully weighed before an appropriate choice may be made. In this chapter, we consider two IDS deployment approaches — centralized and distributed — in addition to the trivial case of having no IDS, which shall serve as a baseline. The centralized approach places the intrusion detection system in the utility network, so that network traffic going to and from the centralized server may be monitored. This approach is relatively cheap, but suffers from the limitation that it cannot observe any of the traffic between the smart meters at the NAN level. The distributed approach is to embed the IDS directly into each smart meter. The approach provides greater traffic coverage than the centralized approach and thus may be able to detect more attacks, or alert at an earlier stage of an attack. In addition, it enables direct detection of attacks on the smart meters themselves. However, it will also cost more, since the per-unit cost of each IDS-enabled smart meter will be higher. The utility must carefully choose among the three IDS options, and should use security models to help make a decision.

2.3.1 Motivation

To show the effectiveness of the novel model generation approach, we present a case study of a hypothetical utility that is deciding whether to select a centralized or distributed IDS approach to protect its AMI, or to do without any IDS at all. The utility is primarily concerned with attacks from malicious customers, insiders, and terrorist organizations. The utility wishes to estimate the amount of damage to the system and the probability of detect-

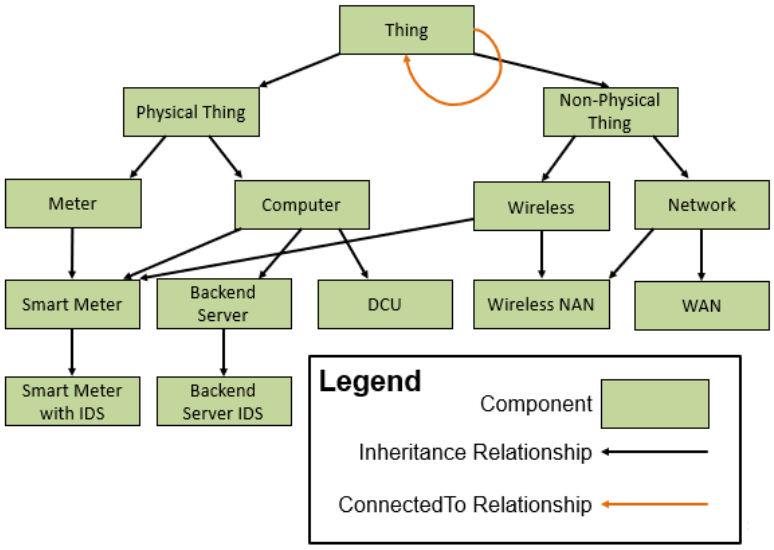


Figure 2.4: The system side of the AMI ontology used to help autogenerate the security model from the system diagram.

ing the adversary for each possible pairing of adversary and IDS approach. The estimate will help the utility evaluate the cost-effectiveness of each IDS approach. However, the employees of the utility do not possess prior knowledge of security modeling and are not capable of easily constructing a security model by hand. We show how an effective security model that can calculate the metrics of interest may be automatically constructed from a simple system model that a domain expert would be comfortable building by hand.

2.3.2 AMI Ontology

The AMI-focused ontology we created for this case study defines the system model elements, the security model elements, and the mapping between them. Since the ontology provides the type definitions allowable in the system model, it must be created before the system model. An illustration of the system side of the ontology may be found in Figure 2.4.

First, we defined the type information used to create the system model. The base component type object in the ontology, *Thing*, is the parent of both *PhysicalThing* and *Non-PhysicalThing*. Both the *Computer* and *Meter* types are children of the *PhysicalThing* type. The *Computer* type is the parent of the *DCU* type and the *BackendServer* type, which

is in turn the parent of the *BackendServerIDS* type. The *Network* type and the *Wireless* type are children of the *NonPhysicalThing* type. The *WAN* type is a child of the *Network* type, while the *WirelessNAN* type is a child of both the *Network* and *Wireless* types. The *DedicatedDeviceIDS* is the child of both the *Computer* and *Wireless* types. Finally, the *SmartMeter* type is the child of three parents (the *Computer* type, the *Meter* type, and the *Wireless* type), while the *SmartMeterWithIDS* type is the child of the *SmartMeter* type. The *NetworkConnection* relationship is the only relationship defined, and signifies that a direct network link exists between the two items that share this relationship.

Next, we defined the AMI-focused attack steps and state variables that are used to help generate the security model. We primarily followed [29] and [33] to come up with the relevant list of attack steps and state variables. There are 17 attack steps in our ontology, and, at a high level, they may be broken into several major categories: (1) attack steps the adversary may attempt in order to gain control of a smart meter, through either a physical exploit or a remote exploit; (2) attack steps the adversary may perform to gain the ability to route traffic in the AMI, which in turn could be used to perform routing attacks and Byzantine attacks; (3) attack steps to form a botnet and perform resource exhaustion attacks; (4) attack steps meant to jam wireless communication in the NAN; and (5) low-tech physical attacks on the AMI infrastructure. Relevant AEG state variables, such as skill in various attacks, access to various parts of the AMI, and knowledge of how things operate internally, are also defined.

Following the definition of system model and security model elements, the appropriate relationships between ontology elements were defined. The AEG state variables were given roles as prerequisites or effects of various attacks, e.g., the *BotnetOnNAN* access is the effect of the *CreateBotnet* attack and the prerequisite of the *ResourceExhaustion* attack. Similarly, each attack and AEG state variable was associated with at least one system model component to which it may be applied, e.g., the *ResourceExhaustion* attack was applied to the *DCU* system model component type, and each one of that attack's prerequisite and effect state variables was associated with either the *DCU* component type itself or one of its ancestors, or was defined to be global. Finally, relevant adversary profiles and metrics were defined in the ontology. With the ontology thus completed, any system modeler can use the component types defined in the ontology to create instances of those types and relationships between

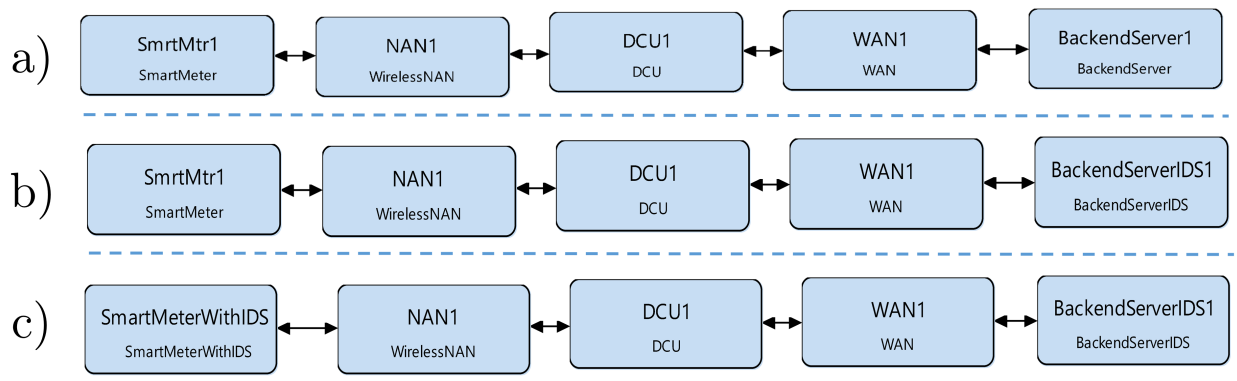


Figure 2.5: Graphical representations of the system models in Möbius. The models (a), (b), and (c) represent the system with no IDS, a centralized IDS, and a distributed IDS, respectively.

them in a hand-built system model.

2.3.3 System Model

Next, we constructed a separate system model of the AMI for each IDS approach. Graphical representations of the system models are shown in Figure 2.5. We modeled the AMI at a high level, both for ease of explanation of the approach and because the metrics we were interested in calculating did not require a model with more detail. Additional detail could be added if required for the accurate calculation of different security metrics, if that level of detail is supported by the ontology. We note that multiple smart meters are present in the modeled AMI, though only one is shown graphically. Multiple smart meters are defined by setting the *NumMeters* attribute of the smart meter instance represented in the graphical model to the desired number (100 in our model).

After we created the system model, we completed the remaining steps. First, we defined goals that the adversaries could attempt to accomplish: remain undetected, steal electricity, damage equipment, or interrupt service. Second, we applied the baseline adversary profiles defined in the ontology to the model and customized them. Third, we applied two metrics defined in the ontology, the *Undetected* metric and the *MonetaryDamage* metric, which calculate (1) the probability that the utility will detect the adversary during the attack and (2) the amount of damage and lost revenue the utility would suffer as a result of the attack, respectively. Finally, we paired the system model with each defined adversary, along

with the metrics of interest, to create a complete configuration. From a configuration, we automatically generated an ADVISE security model consisting of an AEG and an adversary profile. The end-to-end process of automatically generating an ADVISE security model took on the order of minutes, as opposed to the hours it would have taken to create an ADVISE model of comparable size and complexity by hand.

2.3.4 ADVISE Security Models

For each of the 9 configurations (one for each pairing of IDS approach and adversary) we generated a separate ADVISE security model. We manually verified that each automatically generated ADVISE model was correct given the definitions in the ontology and hand-built system model. An example of an AEG that was automatically generated is given in Figure 2.6. Since the ontology is based in part on the attack step and state variable definitions found in [26] and [33], the AEGs that we automatically generated closely resemble the AEGs presented in those publications. It is beyond the scope of this work to explain the security model in detail; consult [26] and [33] for specifics on AMI-focused ADVISE models.

2.3.5 Results

We executed each of the models to obtain estimates of the monetary value of the amount of damage and lost revenue the utility could expect at the end of an attack and the probability of detecting the adversary as the attack was in progress. These results are presented in Table 2.2. The results are similar to those presented in [26] and [33], which is not surprising since the ontology that was used to generate these security models was informed by the security models in those publications. We could have calculated additional security metrics (e.g., the costs and payoffs of the adversary given the chosen sequence of attack steps), but chose not to do so since a full analysis of the security model is outside the scope of this chapter. The calculated results give insight that will aid decision-makers in choosing the most cost-effective IDS. Supplied with the estimates this model provides of the various IDSes, along with cost information provided by the IDS vendors, a system architect can make an informed choice in selecting the most cost-effective IDS.

Table 2.1: Labels and their corresponding names for attack steps or adversary state variables from Figure 2.6.

Label	Attack Step Name	Label	State Variable Name
1	NAN1_InstallLongRangeJammer	A1	NAN1_PhysicalAccess
2	NAN1_InstallShortRangeJammer	A2	NAN1_NumCompromisedMeters
3	NAN1_InstallMaliciousSmartMeter	A3	NAN1_LongRangeJammerAccess
4	SmrtMtr_PhysicalSmrtMtrExploit	A4	NAN1_ShortRangeJammerAccess
5	SmrtMtr_MassMtrCompromise	A5	NAN1_RoutingCapability
6	SmrtMtr_RemoteSmrtMtrExploit	A6	NAN1_BotnetAccess
7	NAN1_CollectCryptoKeys	G1	UndetectedGoal
8	NAN1_CreateBotnet	G2	DamageEquipmentGoal
9	NAN1_AnalyzeTraffic	G3	InterruptServiceGoal
10	NAN1_GainRoutingCapability	K1	NAN1_CryptoKeys
11	NAN1_MajorJammingAttack	K2	NAN1_TrafficKnowledge
12	NAN1_MinorJammingAttack	S1	NodeInstallationSkill
13	NAN1_PhysicalAttack	S2	SmartMeterInstallationSkill
14	NAN1_MinorRoutingAttack	S3	PhysicalSmartMeterExploitSkill
15	NAN1_MajorRoutingAttack	S4	RemoteSmartMeterExploitSkill
16	NAN1_ByzantineAttack	S5	BotnetShepherdSkill
17	DCU1_ResourceExhaustionAttack	S6	TrafficAnalysisSkill
		S7	RoutingAttackSkill
		S8	ByzantineAttackSkill

Table 2.2: Cost incurred by the utility as a result of the actions of each adversary and the probability of detecting an adversary during an attack, given a particular IDS. Italicized results indicate that the adversaries did no damage because they did not attempt to attack the system.

IDS	Adversary	Monetary Damage	Damage Error	Prob. of Detect. Adv.	Detection Error
None	Insider	\$11.6M	+/− \$440K	0.62	+/− 0.03
	Customer	\$380	+/− \$5	0.05	+/− 0.01
	Terrorist	\$1.2M	+/− \$120K	1	+/− 0
Centralized	Insider	\$3M	+/− \$100K	0.62	+/− 0.03
	Customer	*\$0*	+/− *\$0*	*0*	+/− *0*
	Terrorist	\$1.2M	+/− \$120K	1	+/− 0
Distributed	*Insider*	*\$0*	+/− *\$0*	*0*	+/− *0*
	Customer	*\$0*	+/− *\$0*	*0*	+/− *0*
	Terrorist	\$1.2M	+/− \$120K	1	+/− 0

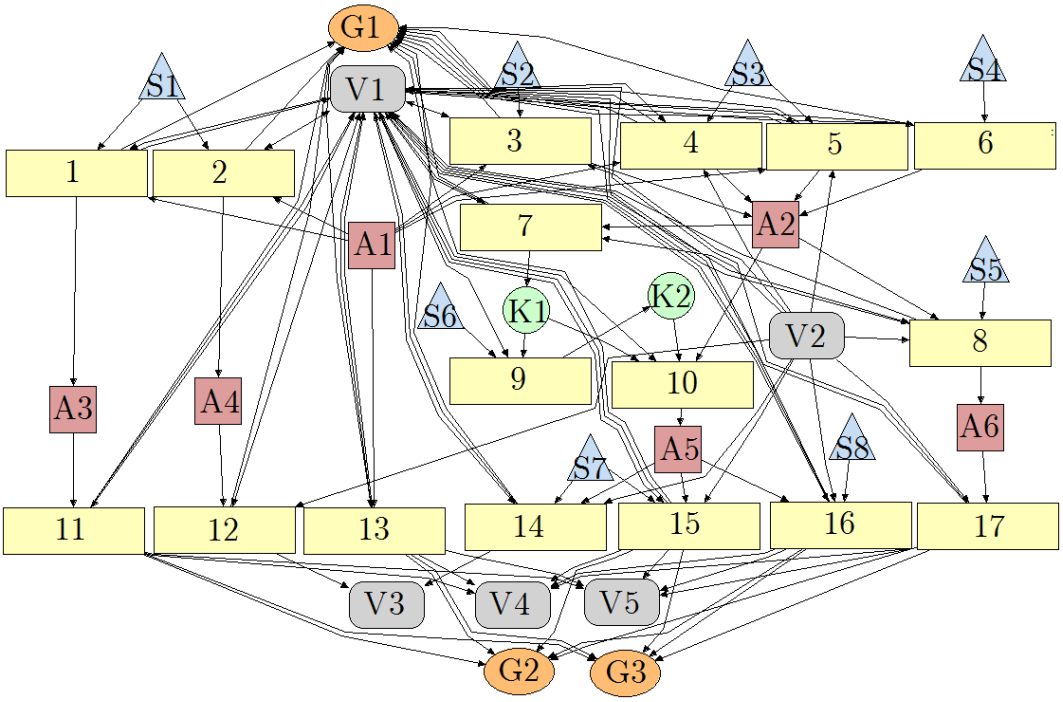


Figure 2.6: The AEG generated from the *AMI with a Centralized IDS* system model paired with the Insider adversary. The gray blocks labeled $V1$, $V2$, $V3$, $V4$, and $V5$ are the $Undetected$, $Centralized\_IDS$, $StealElectricity$, $InterruptService$, and $DamageEquipment$ state variables, respectively. The names corresponding to the other labels may be found in Table 2.1.

2.4 CONCLUSION

In this chapter we presented a novel approach a modeler may employ to automatically create security models from hand-built system models that are used as input to an ontology-assisted model generator. These security models may then be executed to obtain results that may be used to gain insight into the system and support design decisions. We demonstrated the effectiveness of this approach with a case study of a utility that has an AMI deployment and is deciding whether to protect its infrastructure with a centralized IDS or a distributed IDS, or to forgo an IDS. We found that the process of automatically building a security model by first hand-building a system model was much easier and faster than hand-building a security model of similar complexity and size. This is a promising result that may enable security modeling at scale and with a minimal learning curve by those who use the approach.

CHAPTER 3: THE GAMES FORMALISM

3.1 INTRODUCTION

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 [34] [35], as it is key to designing effective security architectures. Risk assessment, while important, is difficult to perform correctly.

Unfortunately, expert-driven, informal risk assessment approaches have several significant limitations, including the difficulty of auditing the expert decisions, the challenge of making assumptions explicit to facilitate collaboration among the experts, and the lack of rigor and scientific basis to give confidence in the forecasts. 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, users, customers, law enforcement, etc. 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, for example. The formalism enables the modular construction of individual state-based agent models. The formalism also allows the modeler to compose these individual agent models into one model so the interaction among the adversaries, defenders, users, and other humans may be studied. Once constructed, this composed model can be executed. During the execution, each individual agent 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.

3.2 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 [36] [37], a survey of attack tree and attack-defense tree models [38], and a survey that includes a useful taxonomy of security models [39].

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 [40]. Other formalisms model the adversary and defender, but neglect the role and impact of users. For example, the attack-defense tree formalism [41] and related attack-defense graph formalism [42] 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 [43], and some agent-based simulation approaches [44] [45]. 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 [46] [47] [48]. However, this assumption may not produce useful security models [39].

The GAMES approach is inspired, in part, by the ADVISE formalism [25]. 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 the actions of humans other than the adversary (e.g. defender and user actions) when designing secure systems, and explicitly incorporates them as first class model elements.

3.3 FORMALISM

In this chapter, we give a comprehensive overview of the *General Agent Model for the Evaluation of Security* (GAMES) formalism, along with a worked example to demonstrate the approach. 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 the various agents 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. Figure 3.1 gives an overview of the individual components of the formalism. One or more agent models may be joined in a composed model. This 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.

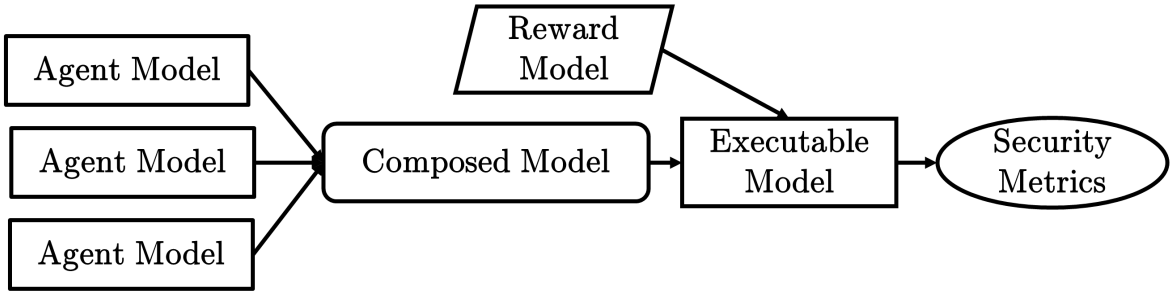


Figure 3.1: Overview of the General Agent Model for Evaluation of Security flow.

3.3.1 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, which will describe how the submodels relate to one another. An agent model represents one acting entity in the cyber system (e.g. 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 defined they may be composed into a model that defines the relationships between the agents. The modeler can then study 1) how the 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 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 [40]. Unlike attack trees, an AEG contains state, and therefore it shares some similarities with generalized stochastic Petri nets (GSPNs) [49] and stochastic activity networks (SANs) [50]. However, it is most similar to the Attack Execution Graphs of ADversary VIew Security Evaluation (ADVISE) models [25].

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 only used 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 decision function the agent uses to decide what action to take to change the state of the system.

An *Action Execution Graph* (AEG) is defined by the tuple $< S, A, C >$, 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, 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 variables 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 [25]. 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 to a more advantageous state for the agent. An action 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 or not the action is currently enabled. An action may only be taken by an agent if it is 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.

$C : Q \rightarrow \mathbb{R}^{\geq 0}$ 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

$Pr : Q \rightarrow [0, 1]$ is the probability that outcome $o \in O$ will occur. Naturally, $\Sigma_{o \in O} Pr(s) = 1$ for all s.

$E : Q \rightarrow Q$ is 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 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 the AEG could be in, that may be impractical in many cases because of state-space explosion. We anticipate that some state variables will be designated as *goal* variables, all of which are initialized to a value of zero, and that the agent will obtain a payoff for changing the model state such that every goal state variable has a positive value.

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 [25]. 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.

In this research, we shall extend the state-sharing approach of the Replicate/Join formalism [51] to enable agents to interact with one another. This state-sharing approach will allow an agent to read and write state in another agent model. For example, agent models of defenders may pass messages to one another through shared state variables that serve as communication channels to coordinate defenses.

3.4 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.

We chose to create our own simulator for the speed of development and ease of customization. We believe other existing simulation frameworks, such as FLAME GPU[1], can also support the modeling formalism and speed up model execution by utilizing GPUs.

We will begin by discussing the composed model. Next, we will examine each of the individual agent submodels that are composed together to form the composed model, looking particularly at the two principal elements of each submodel: the Action Execution Graph and the agent decision algorithm. We will then explain how each of the metrics is constructed to form a reward model. Finally, we will present and interpret results from the executed model.

3.4.1 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. A value of 0 denotes that the account has not been hacked by the adversary and is being actively used by the user. A value of 1 signifies that the account is compromised and can be accessed by the adversary. A value of 2 indicates that the account has been abandoned by the user but is not accessible by the adversary. This state variable is present in every submodel,

[1] http://www.flamegpu.com/

because every agent could take at least one action that either reads from or writes to the state variable.

- **Noisiness:** The cumulated evidence the attacker left (observable by the defender) as a result of the actions the attacker took. It can take values in a range from 0 (stealthiest) to 10 (noisiest). Its value is incremented when the adversary takes actions that leave evidence observable by the defender. The adversary can take actions that write to this state variable, and the defender can take actions that read from the state variable.

- **Password Complexity:** A user's password strength. The state variable can take values between 0 and 10. A weak password such as "password" or "12345" might have a value of 0, while a longer password with upper case letters, lower case letters, numbers, and symbols would have a much higher value. The state variable is present in the adversary's and users' Action Execution Graphs.

- **Password Reset Requested:** This Boolean-valued state variable indicates whether the defender has requested that a user change an account password. It is a boolean variable - a value of 0 signifies that the defender isn't currently asking the user to change the password, while a value of 1 signifies that the defender wants the user to change the password. In effect, it serves as a communication channel that the defender utilizes to convey a request for a particular action. This state variable exists in both the defender and users' Action Execution Graphs.

- **Social Engineering Skill:** The adversary's level of skill in this attack.

- **Service Fee for Defender:** Payment made by the user for account access. The payment may not be directly monetary - it could represent something valuable like the user's willingness to observe advertisements served by the operator. The state variable is found in the operator-view and users' Action Execution Graphs.

- **Threat Discovered:** Defender's knowledge of an attempted attack. It can contain one of three value: 0 (indicating that the threat has not been discovered), 1 (indicating

that the threat has been discovered by not yet communicated to the user), and 2 (indicating that the threat has been uncovered and the user informed). The state variable is contained in the defender's and adversary's Action Execution Graphs.

- **Time to Discovery:** Days until the media learn of a successful attack. It is only present in the Media's Action Execution Graph.

- **User Alarm:** User's fear of loss due to an account compromise. Every agent (except the adversary) has this state variable in their Action Execution Graph.

- **User Benefit:** The value of the benefit the user accrues. The role it plays for the user is similar to the role played by the *Service Fee for Defender* state variable for the defender. It is found solely in the users' Action Execution Graphs.

- **User Fatigue:** The user's level of fatigue from using the defender's service. It can be found in the defender's and users' Action Execution Graphs.

- **User Gullibility:** Susceptibility of a user to social engineering. Every agent (except the media) has this state variable in their Action Execution Graph.

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.

In addition to the state variables mentioned above, the four submodels contain ten actions. The attacker can choose to perform either a *Dictionary Attack* or a *Social Engineering Attack*. The defender may select one of four actions: *Discover Threat*, *Email User Info*, *Request Password Reset*, and *Perform Core Work*. *Use Service*, *Use Alternate Service*, and *Change Password* are actions that the user may choose to perform. The media has only one option of action - it may publicize a hack it discovers by performing the *Publicize Hack* action. Each action takes one day to complete, except for the *Publicize Hack* action, which takes 180 days to complete. The details of the ten actions, including their prerequisites and effects, are explained in the descriptions of the submodels that follow below.

3.4.2 Adversary Submodel

This submodel contains the attacker's Action Execution Graph (visually represented in Figure 3.2). In addition, it includes the adversary's agent decision algorithm. In this case study, we assume that the adversary has already acquired a database of hashed passwords, either through a prior compromise or on the black market. First, we will consider the agent's Action Execution Graph, then the 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. It follows that the *User Gullibility*, *Threat Discovered*, and *Social Engineering Skill* state variables are preconditions for the attack and that the *Noisiness* and *Account Access* state variables are effects of the attack. The probability of a successful dictionary attack depends directly on the complexity of the user's password. Therefore, the *Password Complexity* state variable is a prerequisite of the attack, and *Account Access* state variable is an effect of the attack.

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.

3.4.3 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 3.3.

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

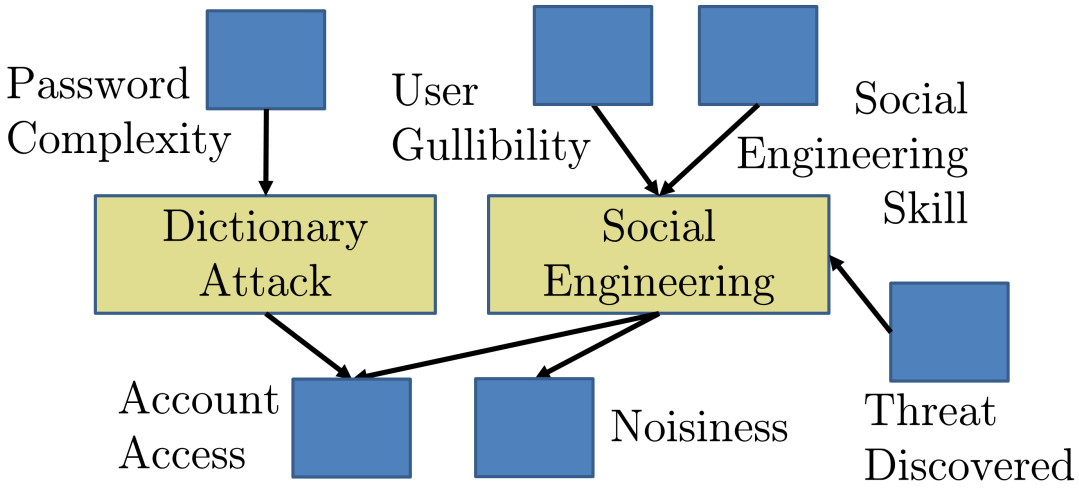


Figure 3.2: A graphical representation of the adversary's Action Execution Graph.

a password reset (the *Request Password Reset* action, or working on core business activities (the *Perform Core Work* action). The defender Action Execution Graph includes these actions, their precondition and postcondition state variables, and one other state variable (*Service Fee for Defender*) that is neither a precondition nor a postcondition for any action in the submodel. The state variable is included in the Action Execution Graph because the defender wishes to maximize the fees they collect from the users for providing the service.

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.

The aggressive policy may allow a defender to detect an adversary more quickly and lead to a more educated and cautious user-base but is expensive to maintain and may exhaust or annoy the users. The passive policy is easy and cheap for the defender and places a minimal burden on the user, but may allow an adversary to go undetected for quite some time. The balanced policy seeks the benefits of each policy while limiting the drawbacks. But does it strike the right balance?

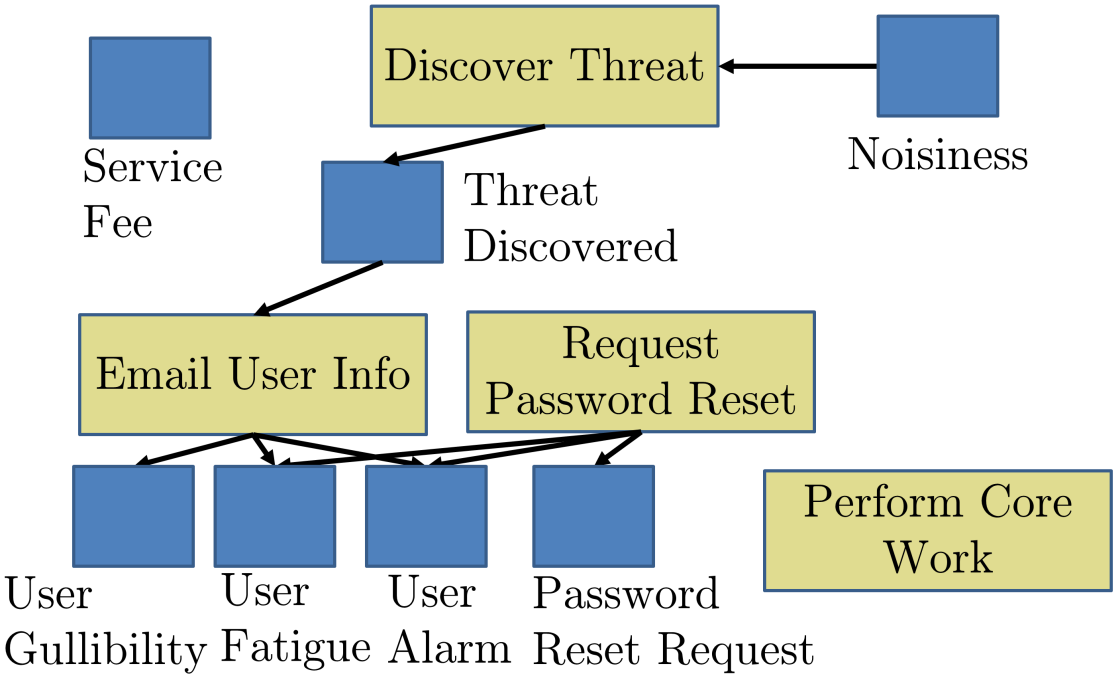


Figure 3.3: A graphical representation of the defender's Action Execution Graph.

3.4.4 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 3.4. 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: A user has three options: reset the password, use the defender's service, or use an alternate service. 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.

3.4.5 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.

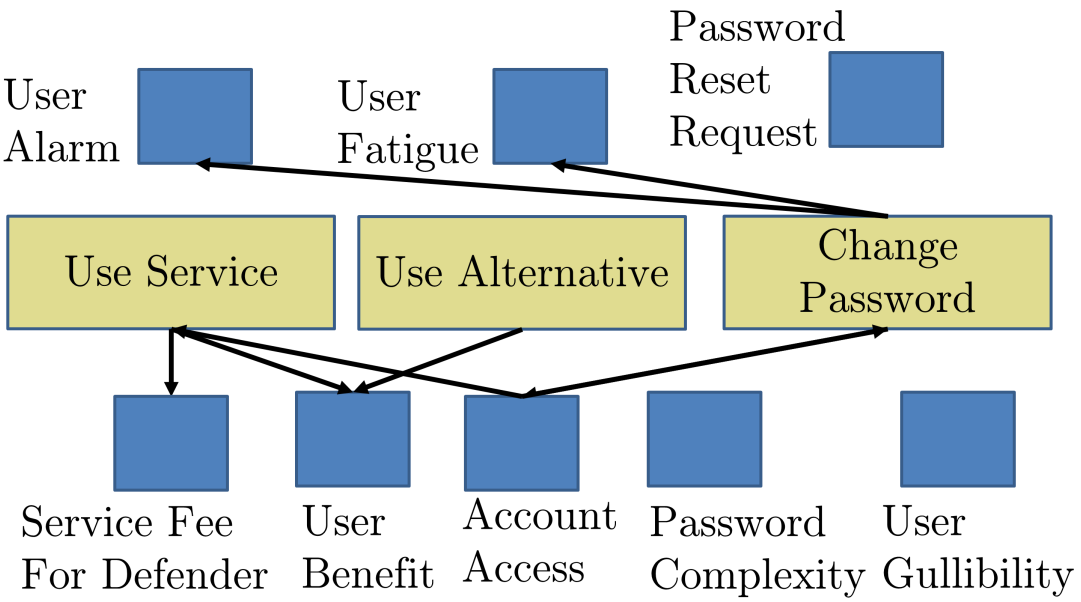


Figure 3.4: A graphical representation of a user's Action Execution Graph.

Action Execution Graph A graphical representation of the media's Action Execution Graph is given in Figure 3.5. 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.

Media Decision Algorithm 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.

3.4.6 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.

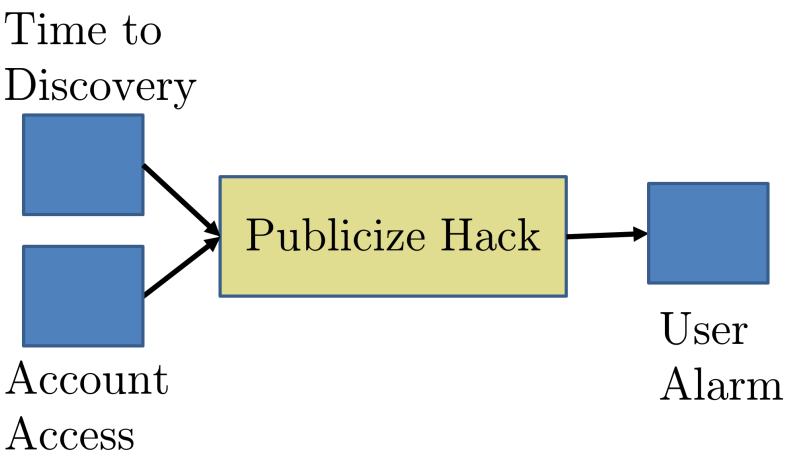


Figure 3.5: A graphical representation of the media's Action Execution Graph.

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. In general, payoffs and costs do not have to be in the same units in GAMES models. However, in this particular model, they are expressed in the same units. This allows the modeler to easily determine the profit earned by the operator.

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

Table 3.1: Results from simulation experiments.

	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

metrics are calculated by observing the value held by each of the 200 instances of the *Account Access* state variable – as explained previously, a value of zero indicates that the account is uncompromised, a value of one indicates that the account is compromised, and a value of two indicates that the account is abandoned.

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.

The suite of metrics will give modelers insight into metrics that are important to the defender/operator. Operators can use the calculated metrics to find the particular policy in a set of policies that will maximize profit, minimize detection time, and maintain the integrity of user accounts.

3.4.7 Model Execution and Results

The complete model may be executed to calculate the defined metrics. At every step of the simulation, the simulation clock is updated and all enabled actions which have completed their execution time are executed in a non-deterministic order — one outcome is probabilistically selected from each action, and the outcome's effect is applied to the model state. Then each agent may choose one action using their decision algorithm, considering the new

state of the system. After this process, the simulation proceeds to the next step and the process repeats. 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 3.1. 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.

3.5 CONCLUSION

In this chapter 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.

CHAPTER 4: STACKED METAMODELS

Many state-based discrete-event simulation models of real-world systems are complex, large, and contain uncertain input parameters. This is especially true of cybersecurity models. It is challenging to make realistic quantitative models smaller and simpler (and thus faster to execute) because the world is large and complex. It is also very difficult to remove uncertainty in the model input values. Obtaining precise, certain input values in many domains may be prohibitively expensive or even impossible. Special approaches must be developed to make effective use of such models, given the issues of long run times and uncertain input values.

The traditional way of handling uncertain input parameter values is to perform (a) sensitivity analysis (SA) to determine the most sensitive inputs, and (b) uncertainty quantification (UQ) to determine how the uncertainty in the inputs propagates to uncertainty in the model output. Both SA and UQ typically require that models be solved many times, with the input variable values being varied each time. If the calculation of the model's metrics could be done quickly, comprehensive SA and UQ could be accomplished with a reasonable computation and time budget. If the model input values were known with certainty, it would be unnecessary to execute the model multiple times to perform SA and UQ, so the time and computation required to obtain a single model solution would be less of a concern. However, the twin issues of long solution times and uncertain model input values in complex state-based models present a significant challenge to modelers.

We propose the use of *metamodels* (also known as *emulators* or *surrogate models*) to address the two issues. Metamodels are models of the original base model that attempt to approximate the relationship between the base model's inputs and outputs, and can generally be executed much more quickly than the base model. With an acceptably accurate metamodel, fast and comprehensive sensitivity analysis and uncertainty quantification can be performed on the metamodel in place of the original base model. While metamodels can be constructed by hand, often they are automatically constructed using machine learning techniques (e.g. Gaussian process regressors, multilayer perceptrons, and random forests).

A chief concern with metamodeling is the choice of an appropriate machine learning tech-

nique, as each has its own strengths and weaknesses. While most related work arbitrarily chooses a particular machine learning technique, or evaluates a small handful of different techniques and chooses the strongest, in this work we use an ensemble of heterogeneous regressors in an effort to benefit from the strengths of each approach while mitigating the weaknesses. We structure the ensemble using a custom stacking approach. Stacking is a cutting-edge technique used by the winners of some recent machine learning competitions [52], but we adapt it for use on state-based discrete-event simulation models.

To the best of our knowledge, we are the first to propose and demonstrate a metamodeling-based approach to the analysis of complex quantitative security models, and the first to use a stacking-based approach to perform SA and UQ on real-world quantitative models. We show that our stacked metamodels are several orders of magnitude faster than the original models, are more accurate than traditional metamodels, and are amenable to SA and UQ that could not be performed on the original base model within a reasonable time budget. We use preexisting models used in previously published papers, namely, a SAN botnet model [53], an ADVISE AMI model [26], and six PRISM models as test cases for the metamodeling approach.

4.1 APPROACH

In our context, a *metamodel*, also known as a *model surrogate* or *emulator*, is a model of a model (which we will refer to in this work as the *base model*) that attempts, given a particular vector of input variables (which we shall call an *input*), to produce an output that matches as closely as possible the output that the base model would produce given the same input, for all inputs. Metamodels can rarely achieve perfect accuracy in emulating the base model, but they often run much faster than the base models. The long time needed to run the base model, together with the need to run the base model many times to conduct sensitivity analysis, uncertainty quantification, and optimization, provides the motivation to find fast metamodels that can emulate the base model with acceptable accuracy.

While high-quality metamodels can be constructed manually by an expert familiar with the base model, it is often easier and faster to build the metamodel automatically. At a high

level, the metamodeling process is conducted in three stages:

1. Data for training and testing are acquired by generating a number of different model inputs and running the base model with those inputs to observe the resulting outputs.

2. The training data are used by a machine learning algorithm to train a metamodel.

3. The test data are used to assess the quality of the trained metamodel.

To begin, in the first stage, data for training and testing must be acquired. Time and computation constraints restrict the maximum number of inputs that can be run on the base model. We can imagine that an n-dimensional input vector describes a point in the n-dimensional input space.[1] The metamodel will benefit from high-quality training data that gives the most complete view of the input space possible, given the limited number of samples. For example, if all the training inputs are clustered closely together in the input space, the trained metamodel may be accurate only in that limited region of the input space. The input space should be explored as efficiently as possible. The most important decision at this first stage is the choice of strategy for input selection. We consider three input selection strategies in this chapter: random sampling, Latin hypercube sampling, and Sobol sequence sampling.

In the second stage, the training data gathered in the previous stage are used to train a metamodel: a model of the original base model that attempts to produce the same output the base model would produce if it were given the same input. One can choose from a variety of machine learning regressors, including, e.g., kriging (Gaussian process regressors), random forest regressors, support vector machine regressors, and k-nearest neighbors (KNN) regressors. The most important decision at this stage is the selection of the machine learning technique that will be able to produce the most accurate metamodel given the training data collected in the first stage. Instead of selecting one regressor, we use the predictions from multiple heterogeneous regressors through stacking.

In the third stage, the test data are used to evaluate the accuracy of the trained metamodel. Given each input in the test data, the metamodel will produce an output, and the

[1] We limit ourselves in this analysis to one-dimensional input variables.

metamodel's output is compared to the base model's output. The absolute value of the difference between the two outputs quantifies the error of the metamodel. If test inputs are generated randomly and independently, one can use the standard statistical methods to determine the average error and associated confidence interval.

The remainder of this section will be devoted to explaining the procedures for obtaining the training data and constructing the metamodel.

4.1.1 Sampling

Acquiring appropriate data for training and testing is vitally important for constructing metamodels and evaluating how well they emulate the base model. The general idea is to choose an input (in other words, a vector that contains a specific value for each input variable), execute the base model with that input, observe the resulting model output or metric, and then record the input and output by adding them to the list of previously observed input-output pairs. This process is repeated multiple times until some stopping criterion is reached, such as the exhaustion of the allocated time budgeted for acquiring training data. The choice of input at each iterative stage is very important. If all the inputs in the training data are "close" to one another in the input space, the metamodel trained on that data may not be able to accurately emulate the base model in other regions. In this chapter, we consider three ways of exploring the input space: random sampling, Latin hypercube sampling, and Sobol sequences. The differences between the three methods are illustrated in the two-dimensional case in Figures 4.1, 4.2, and 4.3.

Random Sampling

Random sampling is the simplest sampling strategy that we consider. Random sampling is conducted by selecting a value for each input variable from the range of possible values at random, independently of any prior sample. Random sampling has a chance of exploring any region in the input space, unlike deterministic sampling, but random sampling can have the unfortunate side effect that samples may cluster in regions of the input space that have already been well explored while ignoring regions that have not been explored at all. The

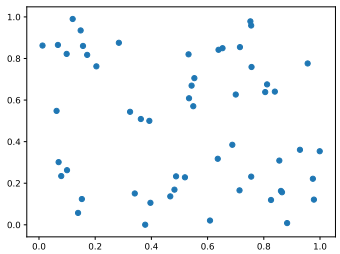


Figure 4.1: 55 Random samples.

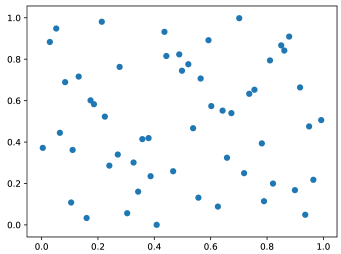


Figure 4.2: 55 Latin hypercube samples.

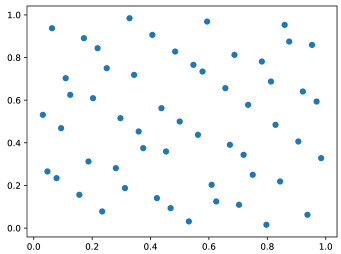


Figure 4.3: 55 Sobol samples. Notice the uniformity.

clustering effect can easily be observed in the two dimensional case shown in Figure 4.1.

Latin Hybercube

The Latin hypercube sampling (LHS) [54][55] strategy attempts to explore the space more evenly than random sampling. It is inspired by the Latin square puzzle, which consists of an n by n array of n unique symbols such that a particular symbol will appear exactly once in every row and every column.[2] In the two-dimensional case, Latin hypercube sampling generates N samples by dividing the range in the x and y dimensions into N equally likely intervals, forming a grid of N rows and N columns, and choosing a sample such that each row and each column is sampled exactly once.[3] Similarly, in the case of a finite number of inputs, LHS generates N samples by dividing each input variable into N equally probable intervals and choosing exactly one sample from each interval. LHS provides stronger guarantees of coverage than random sampling, because each interval is sampled once, while a particular interval may not be sampled at all with random sampling. However, LHS may not evenly cover the space very well. Indeed, there exist pathological cases in which Latin hypercube sampling leaves large regions of the input space totally unexplored.[4] An illustration of 55 samples chosen via Latin Hypercube sampling is shown in Figure 4.2. Notice that the sampling approach produces some pairs that are close to one another in the input space.

[2]A solved Sudoku puzzle is an example of a Latin square.

[3]This is similar to the classic "8 rooks" problem in chess.

[4]For example, taking a sample at every cell along the diagonal is a valid Latin hypercube sample sequence in two dimensions.

Sobol Sequences

The Sobol method, in contrast to random sampling and LHS, generates low-discrepancy sequences. Informally, a low-discrepancy sequence will sample the input region relatively evenly. For a formal definition of discrepancy and technical descriptions of the Sobol generation procedure, please see [56]. An illustration in the 2-dimensional case can be seen in Figure 4.3. Notice how evenly the Sobol method samples the region of interest compared to the random sampling method and LHS.

4.1.2 Metamodeling

A machine learning model can be trained to emulate the base model, producing outputs that are as close as possible to the base model outputs, given the same input. If the base model output is qualitative, a machine learning classifier can be trained as the metamodel. If, on the other hand, the base model output is a quantitative metric, the metamodel will be a machine learning regressor. A wide variety of regressors exist, each with its own strengths. Unfortunately, in general, it is not possible to know a priori which particular machine learning technique will produce the most accurate metamodel for a given black box base model.

We consider a number of regressors in our analysis. Each regressor has its own strengths, weaknesses, and assumptions about the underlying data. We consider seven major types of regressors: random forest (RF) regressors, multilayer perceptrons (MLP), gradient-boosting machines (GBM), the RidgeCV regressor, k-nearest neighbors (KNN) regressors, Gaussian process (kriging) regressors, and stochastic gradient descent (SGD) regressors. Many of the regressors have hyperparameters that change their behavior. For example, the number of neighbors used in the k-nearest neighbors regressor can be any positive integer; different solvers and activation functions can be used by the multilayer perceptron; and the loss function used by the gradient-boosting machine can be changed. In this work we consider twenty-five different regressors of the seven types mentioned above: one random forest, seven different multilayer perceptrons, four different gradient-boosting machines, one Ridge regressor, ten different k-nearest neighbor regressors, one Gaussian process regressor, and one stochastic gradient descent regressor. The collection of regressors we have chosen include

Table 4.1: The 25 regressors and their hyperparameters considered in the analysis.

ID	Regressor Type	Hyperparameters
1	Random Forest	num_estimators=100, criterion='mse'
2	Multilayer Perceptron	solver=adam, activation_function='logistic'
3		solver=adam, activation_function='tanh'
4		solver=adam, activation_function='relu'
5		solver=sgd, activation_function='logistic'
6		solver=sgd, activation_function='tanh'
7		solver=lbfgs, activation_function='logistic'
8		solver=lbfgs, activation_function='tanh'
9	Gradient Boosting Regressor	loss_function='least squares'
10		loss_function='least absolute deviation'
11		loss_function='huber'
12		loss_function='quantile'
13	RidgeCV	
14	K Nearest Neighbors	n_neighbors=1, weight='uniform'
15		n_neighbors=2, weight='uniform'
16		n_neighbors=4, weight='uniform'
17		n_neighbors=8, weight='uniform'
18		n_neighbors=16, weight='uniform'
19		n_neighbors=1, weight='distance'
20		n_neighbors=2, weight='distance'
21		n_neighbors=4, weight='distance'
22		n_neighbors=8, weight='distance'
23		n_neighbors=16, weight='distance'
24	Gaussian Process Regressor	
25	Stochastic Gradient Descent	

some of the most popular regressors currently in use.

One could simply choose a particular regressor to serve as the metamodel (based on intuition or subject matter expertise), or one could train several different candidate regressors, compare their levels of accuracy, and select the one with the best performance to serve as the metamodel. However, the best regressor alone may not perform as well as several regressors working together. The simplest way to use multiple regressors together is to establish a voting committee of regressors, where the regressor predictions are averaged to produce a combined prediction. A drawback of that approach is that poorly performing, inaccurate regressors have the same vote as the most accurate regressor. It would be beneficial to weight the votes, such that the more accurate regressors have more say in the final prediction than

the less accurate regressors. Stacking is one way to accomplish such weighting.

Stacking is a machine learning technique that accomplishes the weighting by training another regressor (or committee of regressors) on the original training data plus the predictions that the original regressors made [57]. We shall refer to the regressors trained solely on the training data as *layer-1 regressors*, and the regressors that were trained on the training data combined with the predictions from the layer-1 regressors as *layer-2 regressors*. We do not know a priori which regressor will perform the best in the second layer, so we train all twenty-five regressors as layer-2 regressors. We then take the average prediction of the layer-2 regressors as the final metamodel prediction.

In practice, we find that some of the layer-1 regressors produce predictions that are so inaccurate that they significantly degrade the performance of some of the layer-2 regressors. Therefore, we filter the layer-2 training data so they include only the predictions from the most accurate, best-performing layer-1 regressors. Similarly, some of the layer-2 regressors are so inaccurate that their predictions would significantly affect the quality of the final vote, so we filter the predictions of those layer-2 regressors as well. The filtering threshold is set to remove the predictions of any regressor whose average prediction error is more than 25% worse than that of the most accurate regressor at that layer. We found that this filtering strategy was effective in the evaluation of our case study model, but the threshold may have to be adjusted for other models, or a completely different strategy could be used (e.g., using the best k regressors out of the set of n, where $k < n$).

Once trained, the stacked ensemble of regressors can form an accurate metamodel of the base model. Input analysis (such as sensitivity analysis and uncertainty quantification) techniques can then be applied to the metamodel in place of the base model.

4.2 ANALYSIS TECHNIQUES

Uncertainty quantification and sensitivity analysis can help a modeler gain a deeper understanding of the model's input variables. Uncertainty quantification determines the likelihood of different outcomes given the uncertainty in the values of the input variables, and can be conducted in a straightforward manner using a Monte Carlo method [58]. Unfortunately,

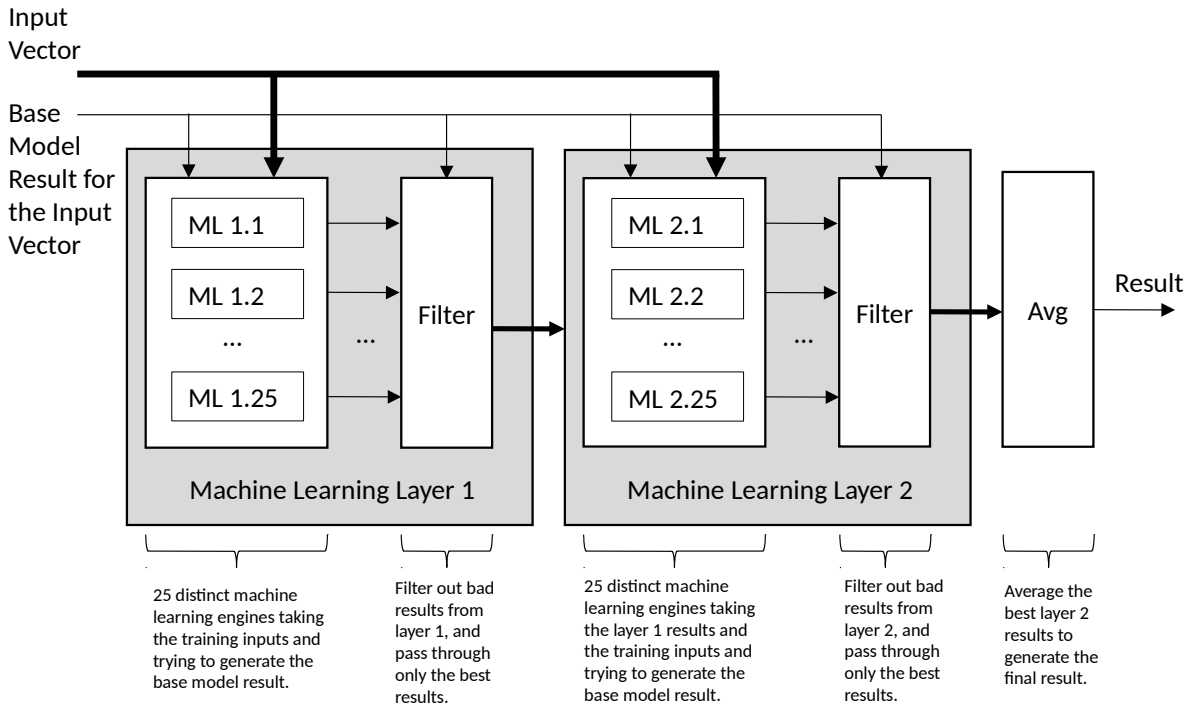


Figure 4.4: Overview of the metamodel construction approach.

sensitivity analysis techniques are more complicated, and the different techniques may give differing answers. Sensitivity analysis attempts to determine the degree to which an input variable influences the output. A number of different techniques can be used to perform sensitivity analysis. We briefly describe three that we use in this work: Sobol sensitivity analysis, the Morris method, and the feature importance method. Sensitivity analysis can be used in a variety of ways. For example, a modeler may be able to dedicate only a limited amount of time and resources to reducing the uncertainty in particular model inputs (e.g., by performing experiments or seeking the opinions of experts). If that is the case, the modeler would like to know the input variables whose values have the greatest effect on the model output, so the uncertainty-reduction efforts can be focused on those variables.

Sobol Sensitivity Analysis: Sobol sensitivity analysis [59] is a method for performing global sensitivity analysis. The method calculates the total order index for each value, which measures its total contribution to the variance in the output. The indices can be used to rank the input variables to determine the most and least sensitive inputs.

Morris Method: The Morris method is used to perform global sensitivity analysis; for details, consult [60][61]. The method varies one input variable at a time. First, it selects a point in the input space and runs the model with that input to determine the resulting output, which is used as a baseline. Next, each one of the input variables is assigned a new value, one at a time, while the others are held at the original baseline value, and the model is run to determine the magnitude of the difference between the resulting output and the baseline output. Once every input variable has been modified in turn, a completely new input is chosen (i.e., all the values for the input variables are changed at once), and the process repeats several times. The Morris method can use the collected data to perform a global sensitivity analysis, calculating a value μ^* for each input variable that quantifies its impact on the model output. By comparing the μ^* values, one can order the input variables based on their impact on the model output.

Feature Importance Method: The feature importance method [62][63] can be used as a way to rank the influence each input has on the model output. We shall describe the method at a high level. First, a metamodel is trained on the training data, and its baseline accuracy is recorded. Then, all the values of one input variable in the training data are perturbed through addition of noise. The metamodel is retrained with the modified training data, and the model's accuracy is compared with the accuracy of the baseline metamodel. If the input value has a large impact on the output value, we would expect a relatively large decrease in the accuracy of the new metamodel. If the input value has no impact on the output value, we would expect no appreciable drop in accuracy. The values of each input variable are perturbed in turn in the same way. The technique can be used to rank the input variables from most impactful to least impactful.

4.3 BOTNET TEST CASE

Our botnet test case considers a stochastic model of the growth of botnets in different conditions. A full description of the model may be found in [53]. A botnet is a collection of computers that have been compromised and hijacked by a malicious actor. A botnet can grow or shrink in size. The model gives an estimate of the size of the botnet at the end of one

week, given a number of assumptions, including the rate at which bots are removed from the botnet by defenders. We imagine that defenders may have a choice among several different methods for removing the bots from the botnet. Methods that remove the bots faster may cost more or have deleterious side effects on the system's performance. In this imagined scenario, the defender would like to know the slowest rate at which the bots may be removed while ensuring that the botnet does not grow above a certain fixed size. The defenders must take into account their lack of knowledge of the precise values of all the input variables. In addition, the defenders would like to know which input variables have the largest effects on the model output, so that they may obtain the most accurate value estimates possible for the sensitive input variables.

We will give a brief overview of the pertinent details. The eleven input variables used in the model are listed in Table 4.2. In [53], all the input variables were assigned baseline values based on suggestions from subject matter experts. In our analysis, we assume that all the inputs are uncertain, and the uncertainty range is $\pm 50\%$ of the baseline values given in the original paper, with the following exceptions: $ProbConnectToPeers$ and $Prob2ndInjctnSuccessful$ are probabilities and thus may not be greater than 1, so we assign $[0.25, 1]$ as a reasonable range of uncertainty; and we increased the range of uncertainty for $RateConnectBotToPeers$, $RateOfAttack$, and $RateSecondaryInjection$ so that the rates fell between once every 10 seconds to once every hour, which we believed to be realistic assumptions.

We used Python to write our sampling, metamodeling, and analysis scripts. The construction of metamodels was accomplished with the aid of the scikit-learn Python package [64], and the SALib package was used to perform Sobol and Morris method sensitivity analysis [65]. All the experiments were run on a machine with an Intel i7-5829K processor and 32 GB of RAM.

4.3.1 Speed Comparison

All reported times were rounded to the nearest tenth of a second. The base model was run one thousand times with random inputs, and it took 7,246.1 seconds (over 2 hours) to obtain

Table 4.2: List of inputs used in the botnet test case.

Variable Name	**Domain**
ProbConnectToPeers	[0.25, 1]
ProbPropagationBot	[0.05, 0.15]
ProbInstallInitialInfection	[0.05, 0.15]
Prob2ndInjctnSuccessful	[0.25, 1]
RateConnectBotToPeers	[0.0166, 6]
RateOfAttack	[0.0166, 6]
RateSecondaryInjection	[0.0166, 6]
RateBotSleeps	[0.05, 0.15]
RateBotWakens	[0.0005, 0.0015]
RateActiveBotRemoved	[0.05, 0.15]
RateInactiveBotRemoved	[0.00005, 0.00015]

the corresponding one thousand model outputs. That works out to just over 7 seconds per input, with a standard deviation of 24.8 seconds. The longest and shortest times it took to calculate an individual output were 510.6 and 0.8 seconds, respectively. The metamodel was also run one thousand times with random inputs, and it took a total of 1.9 seconds to obtain the corresponding one thousand model outputs. It follows that the metamodel can run several thousand times faster than the base model. Therefore, the input space can be searched more thoroughly using the metamodel approach compared to the traditional approach.

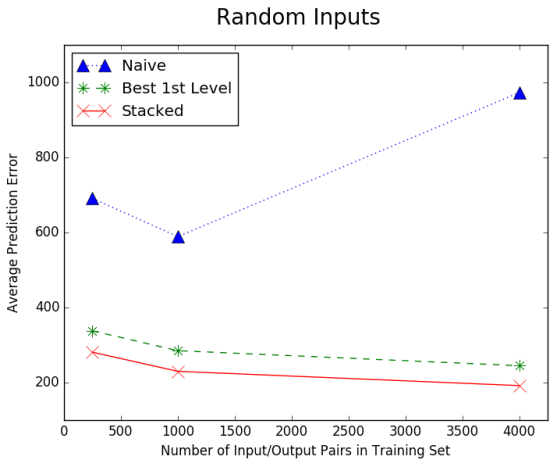
We found that the time needed to train the metamodel was correlated with the size of the dataset. Training of the stacked metamodel took 5 minutes and 19.1 seconds with the dataset that contained 4,000 random inputs, 1 minute 36.0 seconds with the dataset that contained 1,000 random inputs, and 32.0 seconds with the dataset that contained 250 random inputs. The time it takes to collect the training data is significantly greater than the time needed to train the metamodel. Recall that in our approach, only a limited number of samples can be collected and used to train the metamodel, because it takes such a long time to execute the base model. The training datasets will therefore be relatively small, leading to relatively fast training times.

4.3.2 Metamodel Accuracy

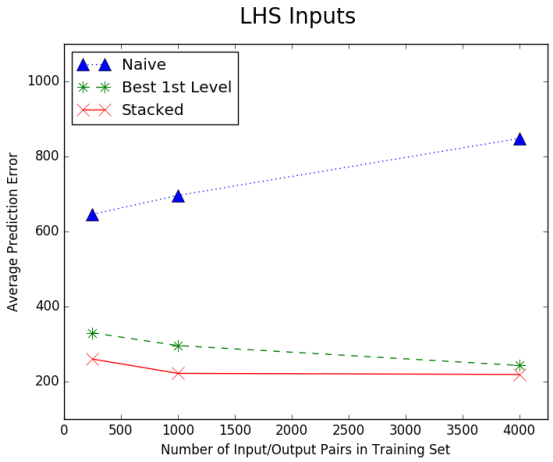
Having established the speed with which the metamodel can be executed, we turn to an evaluation of the metamodel's accuracy. Recall that the metamodel attempts to produce an estimate of the final size of the botnet after one week. The estimation error is the absolute value of the difference between the metamodel's estimate and the base model's estimate, given a particular input. The errors reported are the average absolute difference between the metamodel's predictions and the corresponding base model's outputs across all the data in the test set. 2,500 randomly generated inputs (and associated base model outputs) comprised the test set. The minimum and maximum botnet sizes recorded in the test set were 0 and 37,143, respectively.

First, we consider the accuracy of the metamodel given the three different input sampling strategies and three different training dataset sizes. The results can be seen in Figure 4.6. Unsurprisingly, we found that the metamodel error decreased when the training set contained more data. We also found that the Sobol sequence was the best-performing sampling strategy as the number of samples grew: as the number of samples grows, the effect of the low-discrepancy attribute of the Sobol sampling sequence becomes more obvious. Encouragingly, it appears that the metamodel can perform well even with a relatively low number of training samples: the metamodel trained with 250 random samples, the least accurate of the nine shown in Figure 4.6, had an average estimation error that was less than 1% of the range found in the test data.

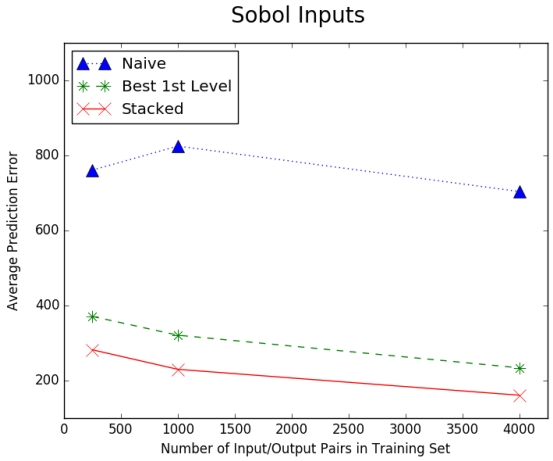
Next, we show that using our stacking approach is better than simply using the predictions from the best-performing of the twenty-five regressors we consider (which we call the *Best of Many* metamodel), and that the predictions of both methods perform better than a naive metamodel. Our naive metamodel calculates the average value of the outputs in the training dataset, and gives that average as its prediction regardless of the model input. Therefore, any well-performing regressor should produce more accurate predictions than the naive metamodel. We compare the errors of the naive metamodel, the most accurate single regressor (the Best of Many metamodel), and the stacked metamodel, given different training data. The results can be seen in Table 4.3. We see that the Best of Many metamodel has about half the average prediction error as the naive metamodel. In the worst case, the



(a) Accuracy Comparison with Random Samples



(b) Accuracy Comparison with LHS Samples



(c) Accuracy Comparison with Sobol Samples

Figure 4.5: Accuracy comparison given random, LHS, and Sobol samples.

Table 4.3: Average metamodel prediction error (lower is better).

Training Data	**Naive Metamodel Error**	**Best of Many Metamodel Error**	**Stacked Metamodel Error**	**Error Reduction Stacked vs. Best of Many**
Random250	691	338	281	17%
Random1000	589	285	230	20%
Random4000	973	245	192	22%
LHS250	646	330	260	21%
LHS1000	696	296	222	25%
LHS4000	848	243	219	10%
Sobol250	761	371	282	24%
Sobol1000	825	321	230	28%
Sobol4000	704	234	161	32%

metamodel composed of stacked regressors has a 10% average error reduction compared to the best single regressor, and in the best case, it achieves a 32% average error reduction. This analysis shows that regressor stacking can lead to a significantly more accurate metamodel for our cybersecurity model compared to simply using a single regressor that is the best among many candidate regressors.

4.3.3 Uncertainty Quantification: Determination of Optimal Removal Rate

Assume that a defender has the ability to remove nodes from the botnet at a specific rate, but a faster rate costs the defender more than a slower rate. That may be the case when the defender can implement more effective but more expensive countermeasures to respond to the attack. The defender may wish to know how quickly the botnet can be expected to grow given different removal rates, and uncertainty in the other input parameters.

We conducted an experiment in which we used our stacked metamodel trained on the dataset consisting of four thousand Sobol samples. For each experiment, we fixed the value of the *RateActiveBotRemoved* variable. We then generated ten thousand Sobol samples for the other input variables in the ranges given by Table 4.2, and ran the regressor with those inputs to observe the predicted botnet size given the conditions described by the input variables.

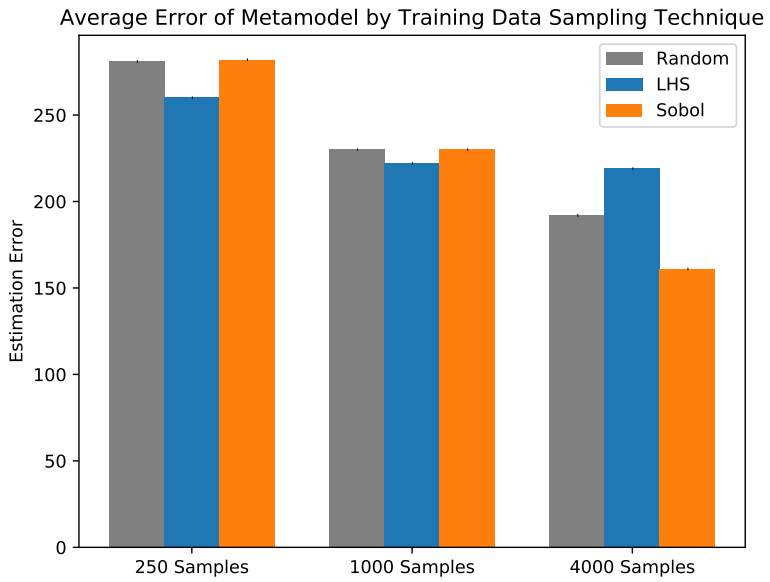


Figure 4.6: Comparison of regressors trained on data obtained from random sampling, Latin hypercube sampling, and Sobol sequence sampling, respectively.

For each value of *RateActiveBotRemoved* we tested, we found the average botnet size, the 95$^{\text{th}}$ percentile, the 99$^{\text{th}}$ percentile, and the largest recorded botnet. That information can help a defender determine the slowest permissible removal rate given uncertainty in the input parameters. The results of our analysis can be found in Table 4.4.

4.3.4 Sensitivity Analysis

We performed a sensitivity analysis to determine the degree to which model inputs impact the value of the output. Traditional SA techniques often cannot be applied directly to the base model because of the long model run times. To overcome that issue, we used several different SA techniques and compared the results to determine the highly sensitive and highly insensitive model inputs. We conducted sensitivity analysis through the Morris method, the feature importance method, and the Sobol method.

The results of the analysis can be found in Table 4.5. Each of the three methods has two columns; the left column gives the score calculated by the method (M_{μ}^{*} for the Morris method, increase in metamodel error rate for the feature importance method, and the total

Table 4.4: Estimated botnet size given removal rate.

Removal Rate	**Average**	**95%**	**99%**	**Largest**
0.05	170	614	2436	9277
0.04	264	984	3861	11568
0.03	493	1970	6330	30004
0.02	1000	4229	10033	31364
0.01	1661	7206	27029	35593
0.001	2163	8594	27573	48152
0.0001	2207	8650	27965	48152

Table 4.5: Input variables ranked from most to least sensitive by taking the average of the rankings provided by the Morris, feature importance, and Sobol methods.

Input Name	**Morris**		**F. I.**		**Sobol**		**Combined**	
	μ^*	**Rank**	**Error**	**Rank**	**Total Ord. Ind.**	**Rank**	**Avg. Rank**	**Std. Dev.**
ROfAttack	3620	1	355	1	3.54	1	1	0
RActiveBRemoved	2648	3	303	3	3.53	2	2.67	0.58
PPropagationB	2492	4	307	2	1.25	5	3.67	1.53
PInstall1stInfection	2675	2	74	7	1.20	6	5	2.65
RBWakens	1311	8	120	4	1.69	4	5.33	2.31
PConnectToPeers	2041	6	13	9	1.83	3	6	3
RInactiveBRemoved	2373	5	77	6	1.09	10	7	2.65
RBSleeps	0	11	103	5	1.13	7	7.67	3.06
PSecondaryInjection	1979	7	74	8	1.09	9	8	1
RSecondaryInjection	1113	9	-20	11	1.13	8	9.33	1.53
RConnectBToPeers	1073	10	-19	10	1.06	11	10.33	0.58

order indices for the Sobol method).[5] Finally, in the rightmost two columns, we show the average ranking across the three methods for each input variable and the standard deviation. We will comment on the results of each SA method in turn, and then discuss how they may be interpreted when taken together.

First, the Morris method returns a μ^* value for each parameter, and since higher values indicate higher sensitivities, we can rank the input parameters relative to one another by using this value. The Morris method indicates that the model output is most sensitive to

[5]The μ^* values and the feature importance errors were rounded to the nearest integer, and all other values in the table were rounded to the nearest hundredth.

the *RateOfAttack* input variable, and least sensitive to the *RateBotSleeps* variable.

Second, the feature importance method determines how much the average error of the regressor's prediction increases (or conversely, how much its accuracy decreases) when a particular input variable's value is distorted with noise. The regressor is least accurate when the *RateOfAttack* input variable's value is corrupted with noise, indicating that variable's importance. The negative error calculated for the *RateSecondaryInjection* and *RateConnectBotToPeers* input variables indicates that the regressor does not make much use of these values when performing the regression. A modeler may therefore choose to ignore the uncertainty in the values of these variables.

Third, the Sobol method calculates total-order indices for the input variables, with higher values indicating more sensitivity. We can see that the Sobol method is in agreement with the other two methods in finding that the model output is most sensitive to the value of the *RateOfAttack* variable. The output is also quite sensitive to the *RateActiveBotRemoved* variable.

Finally, when they are taken together, it can be seen that there is broad agreement among the three methods in their ranking of the inputs. In Table 4.8, we show the combined average rank for each input, which we calculated by summing the input variable's sensitivity ranks as determined by the three SA methods and dividing by the number of SA methods. We also calculated the standard deviation of the three rankings; a low standard deviation shows that the methods are in agreement about the sensitivity rank of an input variable, while a high standard deviation shows disagreement. For example, each method ranked the *RateOfAttack* variable as the most sensitive, so it is given an average rank of $(1+1+1)/3 = 1$ and a standard deviation of 0, which shows perfect agreement. On the other hand, the three methods disagreed the most on the relative sensitivity ranking of the *RateBotSleeps* variable. The Morris method ranked it as the least sensitive, while the feature importance method ranked it as the fifth most sensitive, and the Sobol method ranked it as the seventh most sensitive, for a combined average rank of $(11 + 5 + 7)/3 = 7.67$, and a standard deviation of 3.06. In the absence of ground truth from the base model, it is encouraging to find that multiple SA methods largely agree on the rankings. Furthermore, multiple SA methods can be quickly used on a metamodel once built, so this approach is feasible.

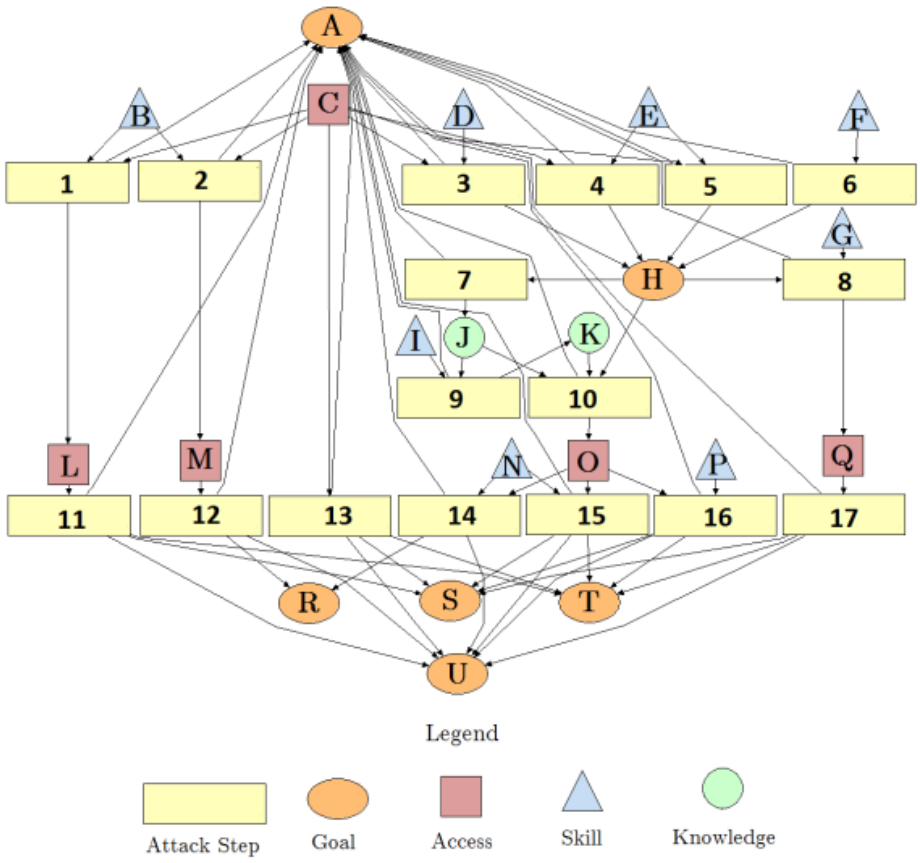


Figure 4.7: Attack Execution Graph of the ADVISE AMI IDS test case model.

4.4 AMI TEST CASE

The next test case we use to demonstrate the approach is built around a model constructed using the ADVISE formalism [25] to compare the effectiveness of different intrusion detection systems (IDSes) in an advanced metering infrastructure (AMI) deployment. The ADVISE model is composed of an adversary profile and an Attack Execution Graph (AEG) (see Figure 4.7), which contains a number of attack steps that the adversary may chain together to perform a full attack on the system, and the system variables that serve as preconditions and postconditions of the attack step. The model's output metric is the monetary damage done to the utility as a result of the adversary's attack, given the presence of a particular IDS in the system. A system architect can use the model to help him or her make an informed choice among the available IDS options. The description of the model was first published in [33], while [26] provides a more in-depth look at the model. Please see those publications for full details.

Table 4.6: List of inputs used in the ADVISE AMI IDS case study.

Variable Name	**Domain**
Adversary	{Insider, Cstmr., Nat. St., Terrorist}
IDS	{None, Central, Dedicated, Embedded}
IDS Multiplier	[0.0005, 0.5]
Payoff Scalers (5 total)	[0.2, 2]
Cost Scalers (10 total)	[0.2, 2]

We consider 18 model input variables in our analysis. The variables are listed in Table 4.6. The first input variable determines the adversary type: the adversary may be a disgruntled insider employee, a customer who attempts to steal power, a stealthy and well-funded nation-state, or a publicity-seeking terrorist organization. The second input variable indicates the IDS present in the AMI: either no IDS is present, or a centralized, dedicated, or embedded IDS is present. An IDS can lessen the probability that an attacker will successfully complete a particular attack step in the model, or increase the cost of attempting that attack step. The third input variable determines by how much the probability of successfully completing the attack step is reduced by the IDS. The value is not known precisely, so we consider a range of values in our evaluation. The values of the next five input variables scale the payoff of each of the adversary's five goals. We do not know precisely how much an adversary values achieving a particular goal, so in our evaluation we establish a baseline value for each goal, and each of these values may be scaled by its respective payoff scaler. Similarly, we use ten cost scalers in the evaluation to scale the baseline estimated cost of attempting an attack step. The baseline values and value ranges for all the input variables were drawn from [26].

In this analysis, we are primarily interested in answering two questions. First, we would like to use sensitivity analysis to rank the contribution of each input variable to the output uncertainty. Armed with that knowledge, a modeler can focus his or her limited time and resources on reducing the uncertainty of the most impactful input variables. Second, we would like to know the degree to which the system architect's choice of IDS is sensitive to the uncertainty in the IDS effectiveness multiplier, the particular adversary goal payoff estimates, and the attack step cost estimates in the model. SA and UQ will help us answer those questions.

4.4.1 Variation: One-Hot Encoding vs. Splitting

The stacked metamodel approach we described earlier in this chapter implicitly assumes that all the input variables will have quantitative values. However, the AMI model in our test case has two qualitative (categorical) inputs: the IDS type and the adversary type. We had to carefully consider how to use the qualitative inputs, which required us to adapt the approach. We chose to evaluate two methods to handle the case of a mix of quantitative and qualitative inputs.

The first, the *one-hot encoding* approach, was inspired by a technique commonly used in ML applications for representing categorical data. With one-hot encoding, the categorical input is removed, and a new binary variable for each category value is added. Thus, using the test case model as an example, the input variable representing the IDS type (which can take one of four values: *None*, *Centralized*, *Dedicated*, or *Embedded*) is removed and replaced with four new binary input variables, one for each value the old input could take. Only one of the four newly-introduced binary input variables will be true at any one time. The metamodel is trained once the dataset has been processed by the one-hot encoding technique.

The second approach, which we call the *split* approach, is to split the training dataset such that there is a separate partition for each possible combination of qualitative values, and then to train a metamodel for each separate partition. With the split approach, the training data from the AMI test case model we consider would be split into 16 partitions, because there are two qualitative inputs that can each take one of four values $IDS \times ADV$, where

$$IDS = \{None, Centralized, Dedicated, Embedded\} \tag{4.1}$$

and

$$ADV = \{Customer, Insider, NationState, Terrorist\} \tag{4.2}$$

Then a new metamodel would be trained for each dataset, for a total of 16 different metamodels. During testing or use, inputs would be similarly divided and processed by the corresponding metamodel.

The two approaches have different strengths. The one-hot encoding stacked approach uses all of the available data to train the metamodel, while the split-stacked approach can use

only a fraction of the data (about one-sixteenth of the data for the AMI test case model) to train any one of the split metamodels. The ability to utilize all of the data is a strength for the one-hot encoding stacked approach, especially in situations like ours in which it is difficult and time-consuming to obtain additional training data. On the other hand, training a separate metamodel for each data partition may simplify the regression problem. Some machine learning techniques may perform better if trained on a smaller dataset that contains no categorical data. It is not clear *a priori* which of the two approaches will produce the most accurate results, so we compare them in our evaluation.

4.4.2 Analysis

To be effective, a metamodel must (a) accurately emulate the behavior of the base model, (b) be significantly faster than the base model, and (c) be amenable to SA and UQ techniques. In this section, we will show that the metamodel we have constructed has these properties. In addition, we shall show that the Stacked metamodel emulates the base model much more closely than the Best of Many technique.

We used the scikit-learn Python package [64] to build the regressors, and the SALib Python package to perform Sobol sensitivity analysis [65]. All experiments were performed on a computer with an Intel i7-5829K processor and 32 GB of RAM.

4.4.3 Accuracy

Recall that each metamodel is trying to predict, as closely as possible, what the base model would output given the same input. The base model, in turn, gives a forecast of the monetary damage a particular adversary would inflict on an AMI protected by a particular kind of IDS. The *average error* in the metamodel represents the average difference between the metamodel's prediction and the base model's actual output. We had a test set consisting of 1000 randomly generated inputs (in the range given in Table 4.6) and the corresponding outputs from the base model. The smallest value among the outputs was 0, and the largest was 11,608,170. Table 4.7 shows the average error of the different metamodels we trained. The three metamodel types we trained were the Naive metamodel, the Best of Many meta-

Table 4.7: Average Prediction Error Given Training Data and Regressor Type.

Training Data Num. Samples	**Naive Metamodel**	**Best of Many**		**Stacked Many**	
		One-Hot	**Split**	**One-Hot**	**Split**
250	1,594,770	243,531	71,798	249,163	69577
1,000	1,369,770	187,585	60,614	113,093	46,364
4,000	1,379,670	108,962	60,121	79,006	45,675

model, and the Stacked metamodel. We had two versions each of the Best of Many and Stacked metamodels: one trained with the dataset processed by one-hot encoding, and one trained with the split dataset, as described in Section 4.4.1. We trained three metamodels of each type with datasets containing 250, 1000, and 4000 samples, respectively.

As expected, the Naive metamodels perform poorly (all of the other, more sophisticated metamodels are substantially more accurate) and demonstrate little improvement with additional training samples. While the Stacked metamodels trained with 250 samples perform about the same as the Best of Many metamodels trained with 250 samples, the other Stacked metamodels are more accurate than their corresponding Best of Many metamodels. That result suggests that the ensembles composed of stacked regressors are no worse, and, if given enough training samples, are substantially better than the best of the collection of regressors by itself, demonstrating the utility of the stacking/filtering approach given in [4]. Finally, the splitting approach yielded substantially more accurate metamodels than one-hot encoding for our AMI model. All of the split stacked metamodels had an average error that was less than 1% of the observed range of outputs (previously found to be 11,608,170), which should be sufficiently accurate for our sensitivity analysis and uncertainty quantification.

4.4.4 Speed

We rounded all reported times to the nearest second. We ran the base model one thousand times with random inputs, which took 6733 seconds (a little under 2 hours). We also ran the split stacked metamodel one thousand times with the same random inputs, and that took a total of 38 seconds. The metamodel is thus more than 100 times faster than the base

model. Training of the split stacked metamodel with a training dataset consisting of 4000 random samples took 14 minutes and 31 seconds. The code was not parallelized, but both the metamodel training and the execution could easily be parallelized for additional gains in speed. For example, each regressor at a particular level can be trained independently of the others at the same level, and each regressor at a particular level can be executed to obtain its prediction independently of the others at the same level.

4.4.5 Sensitivity Analysis

We conducted a Sobol sensitivity analysis in an effort to determine how much the uncertainty in each input variable contributes to the uncertainty of the output. We performed the sensitivity analysis on both the base model and the split stacked metamodel trained with the dataset containing 4000 random samples. The Sobol sensitivity analysis used 48000 samples. It took over 90 hours to complete on the base model, and approximately 30 minutes to complete on the trained metamodel.

The results of the sensitivity analysis may be found in Table 4.8. The table shows the total order index calculated by the Sobol sensitivity analysis for each input variable. That index measures the contribution of the uncertainty of the input variable on the output. We also show the ranking of the importance of each input variable from most sensitive to least as measured by the total order index. We show the total order index and ranking for both the metamodel and the base model. We also show the *rank error*: the absolute value of the difference in SA ranking between the base model and the metamodel. Only one input variable (*CostScalar6*) has a rank error of more than 2. On average the metamodel's ranking differed from the base model's ranking by only 1.25 places. The metamodel also correctly determined the three most sensitive and three least sensitive input variables.

The results show that the metamodel is a very good surrogate for the base model, as (a) it is much faster, and (b) a sensitivity analysis applied to the metamodel produces results that are very similar to the results obtained by a sensitivity analysis applied directly to the base model.

A modeler may use the rankings provided by the SA to focus his or her attention on

Table 4.8: AMI Sensitivity Analysis.

Input Name	**Metamodel Sobol SA Method**		**Base Model Sobol SA Method**		**Rank Error**
	Total Ord. Ind.	**Rank**	**Total Ord. Ind.**	**Rank**	
CostScalar1	1.069	1	1.221	1	0
CostScalar4	1.055	2	1.189	2	0
CostScalar9	1.047	3	1.180	3	0
PayoffScalar5	0.949	5	1.174	4	1
CostScalar5	0.937	6	1.163	5	1
PayoffScalar1	0.913	7	1.149	6	1
PayoffScalar3	0.910	8	1.140	7	1
CostScalar7	0.874	10	1.102	8	2
PayoffScalar2	0.848	11	1.083	9	2
CostScalar10	0.890	9	1.075	10	1
PayoffScalar4	0.841	12	1.036	11	1
CostScalar8	0.838	13	1.033	12	1
CostScalar6	1.030	4	1.0294	13	9
IDSMultiplier	0.672	14	0.6933	14	0
CostScalar2	0.563	15	0.609	15	0
CostScalar3	0.242	16	0.332	16	0

reducing the uncertainty of the most sensitive input variables (Cost Scalars 1, 4, and 9) while spending less time on the least sensitive input variables (the IDSMultiplier, and Cost Scalars 2 and 3).

4.4.6 Uncertainty Quantification

Recall that we are interested in determining whether the choice of IDS is sensitive to the uncertainty in the model input variables. To help answer this question, we ran 80,000 random samples through the metamodel (20,000 for each IDS option) to thoroughly explore the input space. The results of our uncertainty quantification analysis are summarized in Table 4.9, which shows the average, 95th percentile, and 99th percentile damage calculated by the metamodel for each IDS option. The results confirm that any IDS provides substantially better protection than none, even when the uncertainty in the input variables is factored in. The modeler can use these results to help select the correct IDS for an AMI, given the modeler's risk tolerance and the costs of acquisition and maintenance.

Table 4.9: Estimated Monetary Damage Given IDS.

IDS Type	Average	95%	99%
None	$3,076,859	$11,608,170	$11,608,169
Centralized	$447,718	$3,036,137	$3,040,767
Distributed	$255,162	$1,019,546	$1,019,546
Embedded	$325,785	$1,019,546	$1,019,546

4.5 PRISM TEST CASES

In this section, we evaluate seven different test cases to evaluate how well the stacking technique generalizes. We used the botnet model as a base case. In addition, we used six published PRISM [66] models which, to the best of our knowledge, have never been used as subjects of metamodeling. We examined the publicly available PRISM case studies[6], and from that list we selected those that had many model input variables and were non-trivial. Since we are especially interested in the use of metamodeling for sensitivity analysis, uncertainty quantification, and optimization, which are commonly used when the model has many uncertain input variables, we selected case studies that had many model input variables. We chose to use published models, rather than create synthetic models, to help evaluate the real-world effectiveness of the metamodeling techniques. We are also pleased that these models cover a variety of domains: cybsecurity, reliability, chemistry, and biology. We shall briefly describe each test case in turn.

Botnet The Botnet model is a Mobius model described in detail in [53], and was used as the sole test case in [4]. The model can be used to study the growth of a botnet over the course of a week given certain conditions, such as the probability of an uninfected computer becoming infected and the rate of removal of bots from the net. The values of eleven input variables are uncertain in this model. The input variables, and the range of values that they can take in our evaluation, can be found in Table 4.2.

Circadian The Circadian Clock model [67], based on the abstract model found in [68] and [69], is a CTMC PRISM model. Given rates for transcription, translation, binding,

[6]Available here: https://www.prismmodelchecker.org/casestudies/

Table 4.10: List of inputs used in the Circadian test case and the corresponding ranges of values.

Variable Name	**Domain**
T	[0, 200]
transc_da	[45, 55]
transc_da_a	[450, 550]
transc_dr	[0.009, 0.011]
transc_dr_a	[45, 55]
transl_a	[45, 55]
transl_r	[4.5, 5.5]
bind_a	[0.9, 1.1]
bind_r	[0.9, 1.1]
deactivate	[1.8, 2.2]
rel_a	[45, 55]
rel_r	[90, 110]
deg_a	[0.9, 1.1]
deg_c	[0.9, 1.1]
deg_r	[0.18, 0.22]
deg_ma	[9, 11]
deg_mr	[0.45, 0.55]

release, and degradation, for activator and repressor gene and mRNA the model calculates the amount of activator protein at a given time. There are 17 input variables in this model. The input variables, and the range of values that they can take in our evaluation, can be found in Table 4.10.

Cluster The Workstation Cluster model[70] is a PRISM model from [71]. It can be used to calculate the quality of service (QoS) of a workstation cluster arranged in a star topology. The workstations are grouped into sub-clusters connected by a switch, the switches are connected by a central backbone. The components of the cluster fail and are repaired at specific rates. The values of thirteen input variables are uncertain in this model. The majority of these uncertain input variables are various failure and repair rates. The input variables, and the range of values that they can take can be found in Table 4.11.

Table 4.11: List of inputs used in the Cluster test case and the corresponding ranges of values.

Variable Name	**Domain**
ws_fail	[0.0002, 0.02]
switch_fail	[0.000025, 0.0025]
line_fail	[0.00002, 0.002]
startLeft	[5, 15]
startRight	[5, 15]
startToLeft	[5, 15]
startToRight	[5, 15]
startLine	[5, 15]
repairLeft	[1, 3]
repairRight	[1, 3]
repairToLeft	[0.125, 0.375]
repairToRight	[0.125, 0.375]
repairLine	[0.0625, 0.1875]

Cyclin The Cyclin model [72] is a CTMC PRISM model based on a formal specification from [73]. It models cell cycle control in eukaryotes, given a specific quantity of various molecules and various base reaction rates. This model contains 14 input variables. The input variables, and the range of values that they can take in our evaluation, can be found in Table 4.12.

Embedded The Embedded System model [74] is a CTMC PRISM model of an embedded control system based on a model description from [75]. The embedded system consists of three sensors, a sensor input processor, a main processor, an output processor, two actuators, and a bus that connects the processors. The components have a probability of failing, either permenantly or in a transient manner. Some failures can be repaired by a system reboot. This model can be used to calculate reliability and availability metrics. There are six input variables in this model. The input variables, and the range of values that they can take, can be found in Table 4.13.

Kanban The Kanban Manufacturing System model [76] is a CTMC PRISM model based

Table 4.12: List of inputs used in the Cyclin test case and the corresponding ranges of values.

Variable Name	**Domain**
N	[2, 4]
t	[0, 60]
k	[0, 4]
R1	[0.0045, 0.055]
R2	[0.0009, 0.0011]
R3	[0.0027, 0.0033]
R4	[0.45, 0.55]
R5	[0.27, 0.33]
R6	[0.0045, 0.0055]
R7	[0.0081, 0.0099]
R8	[0.0081, 0.0099]
R9	[0.009, 0.011]
R10	[0.0153, 0.0187]
R11	[0.018, 0.022]

Table 4.13: List of inputs used in the Embedded test case and the corresponding ranges of values. The values form intervals of time. The value 1 represents once a second. For example, *lambda_p* ranges from once every two years to once every month, and *tau* ranges from once every 90 seconds to once every 30 seconds.

Variable Name	**Domain**
lambda_p	[1/(2*365*24*60*60), 1/(30*24*60*60)]
lambda_s	[1/(90*24*60*60), 1/(7*24*60*60)]
lambda_a	[1/(4*30*24*60*60), 1/(7*24*60*60)]
tau	[1/90, 1/30]
delta_f	[1/(2*24*60), 1/(8*60*60)]
delta_r	[1/(5*60), 1]

Table 4.14: List of inputs used in the Kanban test case and the corresponding ranges of values.

Variable Name	**Domain**
t	[1, 5]
in1	[0.5, 1.5]
out4	[0.45, 1.35]
synch123	[0.2, 0.6]
synch234	[0.25, 0.75]
back	[0.15, 0.45]
redo1	[0.18, 0.54]
redo2	[0.21, 0.63]
redo3	[0.195, 0.585]
redo4	[0.165, 0.495]
ok1	[0.42, 1.26]
ok2	[0.46, 1.38]
ok3	[0.455, 1.365]
ok4	[0.385, 1.155]

on a model found in [77]. The model can be used to estimate the throughput of the manufacturing system. There are fourteen input variables in this model. The input variables, and the range of values that they can take in our evaluation, can be found in Table 4.14.

Molecules The Simple Molecular Reactions model [78] is a CTMC PRISM model. Given a particular number of Na, Cl, and K molecules, various reaction rates, and length of time, it can calculate the expected percentage of Na/K molecules. There are nine input variables in this model. The input variables, and the range of values that they can take in our evaluation, can be found in Table 4.15.

4.5.1 Approach

In addition to evaluating the generalizability of the stacking approach, in this section we want to evaluate different committees and filter types to determine the ones that produce the most accurate metamodels. Recall that in prior sections we evaluated only one type of

Table 4.15: List of inputs used in the Molecules test case and the corresponding ranges of values.

Variable Name	**Domain**
T	[0, 0.003]
i	[0, 10]
N1, N2, N3	[10, 100]
e1rate	[90, 110]
e2rate	[9, 11]
e3rate	[27, 33]
e4rate	[18, 22]

committee and one type of filter. There are two main design decisions in metamodeling: committee composition and the filter approach. In general, one may choose any set of regressors as members of the two committees. However, one must of course choose a specific set of regressors. This is an important choice, since the overall accuracy of the metamodel heavily depends on which regressors are members of the two committees. Only one kind of committee was evaluated earlier in the chapter, we now evaluate five different committees in this section with a variety of properties.

Similarly, in general, one may use the predictions of each regressor in the committee, regardless of the accuracy of the regressor. However, we have experimentally found that it can be useful to filter predictions from inaccurate regressors. Only one kind of filter was evaluated earlier in this chapter, however, filtering can be accomplished in a variety of ways. We now evaluate variants on two different types of filters.

Committees We evaluate five different committees in this section. We hypothesize that larger committees would outperform smaller committees, and committees with more heterogeneity would outperform those with less, and wish to test the hypothesis. To that end, we evaluate committees of different sizes, and with varying degrees of heterogeneity in membership. We evaluate a large committee with many heterogeneous regressors (Committee 0), a couple of small committees with relatively homogeneous members (Committees 1 and 2, which only contain different kinds of random forests and multilayer perceptrons, respectively,

as members), a medium-sized committee with heterogeneous members (Committee 3), and a small committee with heterogeneous members (Committee 4). What follows is a description of each committee. A summary of the properties of the committees can be found in Table 4.16.

0. Committee 0: The regressors in this committee are the same as the regressors used in the two earlier test cases that we previously described in this chapter. It is also the largest committee we evaluate, with 25 members. The members are 1 random forest regressor, 8 different multilayer perceptrons (each with a different combination of solvers and activation functions), 4 different Gradient Boosting Regressors (each with a different loss function), a RidgeCV regressor, 10 different KNN regressors (each vary by the number of neighbors and whether the neighbor votes are weighted uniformly or by distance), a Gaussian Process regressor, and a stochastic gradient descent regressor.

1. Committee 1: All of the regressors in this committee are variants of the random forest architecture with different hyperparameters. There are 4 regressors in this committee, and each varies by the number of trees in the forest (either 10 or 100) and by the criterion used (either *mean squared error* or *mean absolute error*).

2. Committee 2: All of the regressors in this committee are variants of the multilayer perceptron architecture with different hyperparameters. There are 6 regressors in this committee, and each has a unique combination of solver (either *adam*, *sgd*, or *lbfgs*) and activation function (either *logistic* or *tanh*).

3. Committee 3: This committee is our representative example of a committee with a moderate number of regressors. There are six regressors in this committee: one random forest, one multilayer perceptron, one support vector machine, one Gaussian process regressor, one KNN regressor, one Gradient Boosting regressor, and one stochastic gradient descent regressor.

4. Committee 4: This committee is our representative example of a small committee of just three regressors, each having a different architecture. This committee contains a random forest, a multilayer perceptron, and an Gaussian process regressor.

Table 4.16: Properties of committees.

Committee Index	**Regressor Mix (Heterogeneous or Homogeneous)**	**Size of Committee (# members)**
0	Heterogeneous	Largest (25)
1	Heterogeneous	Small (4)
2	Homogeneous	Moderate (6)
3	Homogeneous	Moderate (6)
4	Heterogeneous	Small (3)

Filters We evaluate two different types of filters in this section: *Top k* and *Within n Percent*. The *Top k* filter only allows the predictions from the k most accurate regressors to pass through. We consider $k = \{1, 2, 3\}$. The *Within n Percent* filter will allow predictions from all regressors through as long as the regressor's error is no worse than n percent worse than the best performing regressor. We consider $n = \{10\%, 25\%, 50\%, 100\%\}$. With this filter, it is not known *a priori* how many regressors will be filtered - theoretically, all of the regressor predictions could be filtered (except the predictions from the best regressor), or all of them could pass through the filter. As a control, we also construct metamodels with no filters.

4.5.2 Results

In this evaluation, we first determine the most effective combination of committee and filter to use for the metamodel, and rank the committees and filter types by their effectiveness. Then, we calculate the accuracy of the metamodel on each of the test cases. We also investigate by how much the accuracy of the metamodels increase with more training data. Finally, we evaluate speed of the metamodel compared to the base models and the time it takes to train the metamodels.

Before we begin the evaluation, it is important to explain how the accuracy of the metamodels is calculated. For each of the test case base models we created a dataset of one thousand randomly generated input vectors and executed the base model with those vectors

Table 4.17: Most accurate committee/filter combination for each test case. *Note: Properties of committees by given index found in Table 4.16.*

Base Model	**Committee Index**	**Filter Index**
Botnet	0	*Within n Percent*, n=50%
Circadian	1	No filter
Cluster	0	*Within n Percent*, n=25%
Cyclin	1	*Within n Percent*, n=25%
Embedded	0	*Within n Percent*, n=10%
Kanban	0	*Within n Percent*, n=50%
Molecules	0	*Within n Percent*, n=10%

to obtain the corresponding outputs to create a "ground truth" test dataset. The test dataset was distinct from the training dataset. The trained metamodel is executed with all of the input vectors from the test dataset, and the metamodel prediction for each input vector is recorded. We then calculate the absolute error for each vector by taking the absolute value of the difference between the metamodel's prediction and the ground truth model output. We sum all of the absolute errors and then divide by the number of input vectors in the test dataset to obtain the mean absolute error. Finally, to facilitate comparison between different the different test cases (whose outputs have very different ranges of values), we normalize the errors by dividing the mean absolute error by the range of the base model's outputs in the test dataset.

4.5.3 Accuracy Given Different Committee Compositions and Filters

We evaluate five different committee types and eight different filters, so there are a total of $8 \times 5 = 40$ different combinations of committee and filter. The most accurate stacked metamodel committee/filter combination for each test case we consider is given in Table 4.17. We see that Committee 0, the largest and most diverse of the committees, is the best performing committee for five of the seven test cases, and that Committee 1, which is composed of four different random forest regressors, is the best performing committee for

the remaining two test cases. The best performing filters were all variants of the *Within n Percent* filter, with the exception of the Circadian test case, where not having a filter outperformed all of the other filters.

It would be useful to know which committee/filter combination works best in general. Table 4.17 shows that the best committee/filter combination for one base model will not be the best combination for another base model. One may of course try all forty combinations and select the best for their particular model. It does not take long to train a metamodel, so it is feasible to try many different variants. However, it is illuminating to know which committee and filter combinations work well across many different kinds of models. To determine this, we ranked each committee/filter combination for each of our seven test cases from 1 to 40, from most accurate to least. We then calculated the *average rank* for each committee/filter combination by summing all of the individual ranks and then dividing by the number of test cases. The committee/filter combination with the lowest average rank would be the most accurate metamodel across the test cases. When we performed that calculation, we found that Committee 0 (the largest and most heterogeneous committee) paired with the *Within n Percent, n=10%* filter was the most accurate combination across all of the test case models.

It would also be illuminative to rank the committees (each paired with the filter that maximizes its performance) and filters (each paired with the committee that maximizes its performance) across the test models, respectively. We shall describe the process to determine the rank of a committee, a similar process can be used to rank a filter. To rank the committees, we first calculate the average rank of each combination of the 40 combinations of committee/filter as described in the paragraph above, and then sort this list from the smallest value to the largest. We then start at the top of this list and if we see an entry with a committee we have seen before, we delete that entry. When we reach the end of the list we will have a ranked list of committees. By ranking in this way, we compare each committee when paired with the filter that maximizes its performance.

The ranking of the committees and filters can be found in Table 4.18. From the table it is clear that the stacking approach benefits from having committees with many heterogeneous members. Committees with more homogeneity (e.g. Committee 1 and 2) and small com-

Table 4.18: Committees and filters ranked by accuracy.

Rank	Committee Index	Committee Description
1	0	Large
2	3	Medium
3	1	4 RF
4	4	MLP, RF, GPR
5	2	6 MLP

Rank	Filter Description
1	*Within n Percent*, n=10%
2	*Within n Percent*, n=25%
3	*Within n Percent*, n=50%
4	No filter
5	*Within n Percent*, n=100%
6	*Top k*, k=3
7	*Top k*, k=2
8	*Top k*, k=1

mittees (e.g. Committee 4) do not perform as well in general (though, as we saw previously in Table 4.17, they may perform better than the alternatives on specific models). The results validate the intuition that an ensemble with many diverse members will outperform an smaller ensemble with fewer, more homogeneous members.

Table 4.18 also provides striking results for the filter. Clearly, the *Within n Percent* filter dramatically outperformed the *Top k* filter. In fact, the *Top k* filter was worse than having no filter at all. It is interesting that of the filters we considered, the more restrictive *Within n Percent* filters outperformed the less restrictive filters of the same type, while the less restrictive *Top k* filters outperformed the more restrictive. We hypothesize that the *Top k* filters were too restrictive to allow for the diversity necessary for a well-performing ensemble.

4.5.4 Metamodel Accuracy: Naive vs. Best of Many vs. Stacked

Up to this point, we have only evaluated the accuracy of different variants of the stacked metamodel relative to one another. It is also important to know (a) the accuracy of the stacked metamodel compared to other metamodels, and (b) the absolute accuracy of the model.

First, it is important to know whether sophisticated ML techniques perform better than very naive methods. To help make this comparison, we created a *Naive metamodel*, which

consists of a regressor that predicts that the output will be the average of the outputs in the training data, regardless of the input (as we described earlier in the chapter). More sophisticated metamodels must show that they are substantially better than this naive metamodel to demonstrate utility.

Table 4.19 reports the errors of the Naive, Best of Many, and Stacked metamodels. The errors we report are the mean absolute error normalized by the range of observed test values. We normalize the errors to make it possible to compare the accuracy of the metamodels across test cases. We chose to use the stacked metamodel architecture that was most suited for each individual test case (e.g. the metamodel variants listed in Table 4.17), rather than use the same committee/filter combination for every test case. All metamodels were trained with a datasets that contained 1000 random inputs each.

By examining the table we see that the Best of Many and Stacked metamodels always substantially outperform the Naive metamodel, as we had hoped. Further, for five of the seven test cases, the stacked metamodel was more than 5% more accurate than the Best of Many metamodel, and it was never less accurate. The stacked metamodels were, on average, 8.2% more accurate than the base models. These numbers demonstrate the effectiveness of the stacked approach compared to the current state-of-the-practice Best of Many approach, and validates its general applicability.

A metamodel must be accurate enough for it to be a practical tool for modelers. However, it is difficult to know how accurate a metamodel must be to be a good emulator. One of the primary challenges in determining an acceptable accuracy threshold is that a modeler may use metamodels for different reasons, and different use cases may require more accuracy than others. We leave to future work an investigation to determine the acceptable accuracy threshold for a variety of common applications of metamodels.

4.5.5 Accuracy Given Different Training Sample Dataset Sizes

It can be time consuming to collect the training data because the base model runs slowly. To be time efficient it would be best to collect as little training data as possible while maintaining reasonable metamodel accuracy. For this reason we investigated the impact of

Table 4.19: Average metamodel prediction error. The error is the mean absolute error normalized by the range of test values.

Metamodel Type	**Naive Metamodel Error**	**Best of Many Metamodel Error**	**Stacked Metamodel Error**	**Error Reduction Stacked vs. Best of Many**
Botnet	0.00161	0.00111	0.00100	10.4%
Circadian	0.08461	0.05658	0.05648	0.2%
Cluster	0.03322	0.01181	0.01141	3.4%
Cyclin	0.14129	0.02369	0.02111	10.9%
Embedded	0.03726	0.00459	0.00432	5.9%
Kanban	0.17832	0.02798	0.02508	10.3%
Molecules	0.06610	0.03899	0.03262	16.3%

the size of the training set on the accuracy of the metamodel. We trained stacked metamodels with Committee 0 (the largest and most heterogeneous committee) and the *Within n percent, n=10%* filter with datasets that contained 250, 500, 750, and 1000 input vectors, respectively, and evaluated their accuracy. The results of our investigation are contained in Table 4.20. As expected, the metamodels trained with more training data are more accurate than those that were trained with less. However, it is encouraging to see how that even those metamodels that were trained with just 250 training samples are often reasonably accurate compared to the metamodels trained with 1000 training samples.

4.5.6 Speed Comparison

We performed these speed experiments on an Ubuntu VM running on a laptop with an Intel i7-7500U processor and 8 GB of RAM. For each test case, both the base model and the metamodel were run with the same 200 randomly generated inputs, and the time it took to execute with those inputs was recorded. In addition, we recorded the time it took to train the metamodel for each case. We used the same committee/filter combination for each metamodel in these experiments to make it easier to compare across test cases. Committee 0 paired with the *Within n Percent, n=10%* filter was the combination we chose, for two reasons. First, we had previously determined that that combination was, on average, the

Table 4.20: Mean absolute error normalized by range of stacked metamodel trained with training datasets of different sizes.

Metamodel Type	**250 Training Samples**	**500 Training Samples**	**750 Training Samples**	**1000 Training Samples**
Botnet	0.03817	0.03227	0.01816	0.01626
Circadian	0.07011	0.064872	0.06065	0.05707
Cluster	0.02151	0.01631	0.01562	0.01349
Cyclin	0.05285	0.04281	0.03048	0.02606
Embedded	0.02445	0.01547	0.01271	0.01126
Kanban	0.03956	0.03222	0.02956	0.02555
Molecules	0.05672	0.05688	0.05072	0.04750

most accurate across all the test cases. Second, Committee 0 is the largest committee, so it will likely to take longer to train compared to the other committees, so it is a sort of "worst-case" training time.

Table 4.21 shows that the metamodel runs faster than the base model by several orders of magnitude, thousands of times faster. On average it took an hour and twenty minutes to run the base model with the dataset containing 200 inputs, while it took the metamodels on average half a second to run the same dataset. The metamodels are almost ten thousand times faster than their corresponding base model on average. The table also shows that training the metamodel can be done reasonably quickly, in about two minutes.

4.5.7 Discussion

Our analysis can help modelers who are interested in using metamodeling for their own models. At a high level, it appears that reasonably accurate metamodels can be created for a variety of different kinds of models using a variety of different machine learning algorithms. Even relatively slow running complicated ensemble methods run thousands of times faster than the base model, making feasible analyses that would otherwise be unfeasible if performed directly on the base model. We recommend that modelers consider the use of metamodels to help with analyses involving slow-running models that have many uncertain

Table 4.21: Base model vs. metamodel execution speed comparison and metamodel training time (all in seconds).

Name	**Base Model Execution (seconds)**	**Metamodel Execution (seconds)**	**Metamodel Training (seconds)**
Botnet	1115.5	0.3	137.4
Circadian	18291.6	0.5	204.7
Cluster	2034.9	0.6	66.4
Cyclin	2668.2	0.5	76.9
Embedded	684.7	0.2	60.6
Kanban	5737.6	0.3	64.5
Molecules	3714.5	1.1	67.1

input variables.

A stacking-based ensemble approach appears to produce more accurate metamodels than approaches that use a single regressor, even if that regressor is the best among many candidate regressors. Our analysis of the seven test cases shows that the best stacking metamodels were never worse than those metamodels produced by using the Best of Many approach, and were often significantly better. The accuracy of the metamodel was impacted by the size and heterogeneity of the constituent committees: more accurate metamodels had larger and more heterogeneous committees, while less accurate metamodels had smaller and more homogeneous committees. Judicious use of filters can increase the accuracy the metamodel, with relatively restrictive versions of the *Within n Percent* filter being the best we evaluated. Our evaluation shows that it does not take long to train a metamodel if one already has the training dataset, so it is possible to try a number of different metamodel variants to see which would work best for that particular dataset. We recommend that modelers using metamodeling should (1) use stacking rather than the more common Best of Many approach, (2) use committees that are large and heterogeneous in their stacked metamodels, (3) use a filter to remove predictions of under performing regressors, and (4) take advantage of how quickly metamodels can be trained by trying a number of different metamodel variants to find one that works particularly well for the dataset.

We found that the accuracy of the metamodel depends in part on the size of the training dataset: the larger the training dataset, the more accurate the metamodel. However, the stacked metamodels were reasonably accurate even with little training data. This is encouraging, because our original motivation for using metamodels was as a replacement for models that run slowly. Collecting training data can be the most time consuming stage of the ML-based metamodeling, so it is encouraging that metamodels do not require onerously large training datasets to be accurate. We recommend that modelers obtain as much training data as is practical, but to not be discouraged only a small training dataset can be obtained.

A modeler should be aware of the limitation of the approach. If the model being considered runs too slowly it may not be feasible to collect enough training data to build a reasonably accurate metamodel. However, if the model being considered runs quickly, it would be better to do the analysis (whether SA, UQ, optimization, or something else) directly on the model instead of doing the analysis indirectly with the use of a metamodel, because metamodels usually don't perfectly emulate the base model and can introduce errors into the analysis. For this reason, in some cases it may be beneficial to do a hybrid approach in which a broad metamodel-based analysis is paired with (and perhaps even guides) a more limited and narrowly focused analysis on the base model.

4.6 CONCLUSION

We have evaluated the effectiveness of metamodeling on seven real-world published models. We showed that the metamodels we created can be quite accurate in emulating the behavior of the base model, and run almost ten thousand times faster on average. Since the metamodels are so fast relative to the base model, analyses that could take too long to complete on the base model directly (such as sensitivity analysis, uncertainty quantification, and optimization) can be done indirectly with the aid of the metamodel instead in a reasonable amount of time. Metamodels can be constructed automatically with the aid of machine learning, minimizing the manual efforts of the modeler. We evaluated a variety of metamodels, and found that stacked metamodels performed better than the more commonly used Best of Many metamodels. Of the stacked metamodels, in general metamodels that

had larger and more diverse committees were more accurate than those that had smaller and more homogeneous committees. We believe that ML metamodels, like those shown in this work, are an underused and relatively-unknown tool in the modeler's toolbox. We hope that our descriptions and evaluations will encourage their future use in exploring slow-running state-based quantitative models with many uncertain input variables.

CHAPTER 5: THE MODELING PROCESS

We have shown how to use an ontology-assisted automatic model generation approach to ease the burden of making models, we have presented a new formalism called GAMES to help modelers explicitly model all of the agents that can impact the security of a system, and we have shown how to use metamodeling to do faster sensitivity analysis and uncertainty quantification. Each contribution addresses a particular specific modeling need. Not every modeling effort will require any one of the three contributions. Some models should be built by hand, not generated automatically with the aid of an ontology. Some models will only require modeling the adversary or adversary/defender pair, so the GAMES formalism may not be the right choice for every circumstance. Some models may run fast enough to do direct sensitivity analysis and uncertainty quantification, making metamodeling unnecessary. In this chapter, we give more general, higher-level advice that can guide any cybersecurity modeling effort.

It can be difficult to build realistic models when reality itself can be so complicated. Recognizing the need for a model is only the first step. Going from a desire for a model to a useful working model is usually not easy or straightforward. However, there are frameworks and advice that can make the modeling process smoother that we have learned over years of building models. Of particular interest, we (1) created a framework that can help direct the modeling effort, (2) developed a philosophy for cybersecurity metrics that can help modelers get the most out of their models, and (3) gained experience that can help us to give advice on how to handle commonly encountered issues in the modeling process.

The chapter is divided into three sections. In the first section, we present a high-level framework we developed that can guide the cybersecurity modeling process from beginning to end. In the second section, we discuss one of the most important topics in quantitative modeling: metrics. We present an overarching philosophy of cybersecurity metrics, show how the theory of reward-model-based performance variables can be used to construct metrics, and discuss a variety of practical cybersecurity metrics that can be calculated to aid decision makers. In the third section, we give practical advice on topics of interesting in the modeling process. Topics include: collaboration, choosing the right level of abstraction, minimizing

the effects of uncertainty in the values of the input variables, and increasing confidence in the model outputs.

5.1 A MODELING FRAMEWORK

We believe that modeling frameworks can make quantitative cyber security modeling easier and more effective. We have developed our own modeling framework, inspired by years of experience building models, and informed by the methodologies and approaches of others. Prominent examples of existing frameworks include the methodology given by Saydjari [79], the failure mode and effects analysis (FMEA) methodology [80] [81], the Cyber Capability Maturity Model (C2M2) [82], Hierarchical Holographic Modeling approach (HMM) [83], and the SAHARA approach [84]. We do not intend this framework to be used as a rigid checklist, but as an aid that a modeling team may use to inform their efforts to build a model that can support decision making. What follows is our multistage modeling framework that modelers can use as a to guide the modeling process.

1. Form a team to construct the model, including modeling experts and subject matter experts. Subject matter experts should be especially involved in aspects of the modeling that pertain to their expertise, but all team members should seek to learn about and critique the model in its entirety, to the degree that they are able. Subject matter experts should not ignore the aspects of the model that fall outside their domain of expertise. If subject matter experts broaden their scope outside their field of expertise they will both be able help correct mistakes that others make and deepen their own understanding of how the system works as a whole. Ontology-based automatic modeling approaches, like the one presented in Chapter 2, can help modelers to leverage modeling and subject matter expertise in a scalable and repeatable way, since the modeling expertise is encoded in an ontology that can be reused and shared.

2. Clearly define the decision the model is supposed to help make and the corresponding metric or metrics that will help support the decision-making process. When selecting appropriate metrics the modeler should keep in mind the primary goal(s) of the system and seek to define metrics that will give insight into how design decisions will impact

the system's goal(s). We shall discuss the issue of metrics in greater depth later in the chapter.

3. Conduct a review to determine what existing data sets, experiments, models of previous systems, or prior research could be used to help construct the model.

4. With the help of all the experts, develop a model of the system, and all of the entities that may interact with the system (adversary, defender(s), customers, third-party partners, etc.). The abilities, motivations, and decision-making process of each entity that could impact the security of the system should be incorporated into the model. It may be helpful to make composable submodels, one for each human entity or major system component, as in the GAMES modeling formalism [3]. Composable submodels allow domain experts to focus on their own submodel and can make it easier to upgrade or correct submodels in a clean way. We discussed the GAMES formalism that can be used to easily model human players in the cybersecurity game in detail in Chapter 3.

5. Obtain any values needed for the input variables of the model, using prior research, experiments, or data when possible. Otherwise, use reasonable estimates from subject matter experts for the values. The modeling team should do a sensitivity analysis to determine the relative importance of the various inputs (i.e. if the model is not sensitive to the input variable it is less important to obtain a precise value for it). If the model is sensitive to particular input values and the modeler is not confident in the values of the input variables then custom experiments or data gathering efforts to obtain better estimates for the values may be beneficial. If the model runs too slowly for traditional sensitivity analysis techniques, consider using a faster metamodel-based approach, like the one shown in Chapter 4, which can make possible analyses that are otherwise infeasible using standard methods.

6. The modelers should develop a hypothesis (based on their own mental model) of what results the quantitative model may produce before attempting to obtain metrics from the model. The hypothesis may be falsified or supported by the model's execution. Forming a falsifiable hypothesis is one way to make the cybersecurity modeling process

more scientific [85].

7. Obtain metric(s) from the model, and validate the metrics. There are a number of ways the validation can be accomplished. Choose the appropriate validation method or combination of methods for the task at hand. Some of the common validation techniques include:

   (a) Check to see if the model result matches the modeler's previously developed hypothesis. If the two do not match, at least one is wrong. Determine why either the hypothesis was wrong or the model was wrong. If the hypothesis is wrong and the model is right, it gives the modeler new insight and corrects the modeler's mental model which led to the erroneous hypothesis.

   (b) Check against prior data, experiments, or research results, if available.

   (c) Check against another model built in a different formalism (possibly by the same modeling team).

   (d) Check against models built by other independent teams. If the results do not match, determine the cause.

   (e) Check against experiments carried out in emulation.

   (f) Check against experiments carried out on the actual implemented system. Red teaming is a good example of this kind of experiment.

   If the model fails a validation check, the modeler should return to the model and determine why the model is incorrect. The modeler should then fix the model and rerun the validation check.

8. Once validated, the modeler can use the model results to help make a decision.

It is important when following this method that the perfect not be allowed to become the enemy of the good. In particular, in many cases it may be challenging to obtain accurate values for all of the input variables needed for a particular model, and to validate the model as well as the modelers would wish. However, it is our belief that it is better to attempt to create a formal quantitative security model with a clear understanding of its limitations

and weaknesses than to depend solely on informal mental models a decision maker may use instead.

5.2 METRICS THAT HELP DECISION MAKERS

Our framework helps modelers to fulfill the purpose of quantitative modeling, which is to produce metrics that help decision makers make better decisions. The metrics themselves must be chosen with great care. Everything about the design and production of the model is ordered towards producing high-quality metrics. However, it is possible for novices and even experts to take counterproductive approaches to metrics. We aim to address issues related to metrics in this section.

We begin this section on metrics with a discussion on a philosophy of cybersecurity metrics that focuses on the maximization of the expected utility of the system, then a review of the theory of performance variables based on the *reward model* formalism that we leverage to construct metrics, and then finally a presentation of a number of example cybersecurity metrics.

5.2.1 A Philosophy of Metrics

At first glance, it may appear reasonable that quantitative cybersecurity modeling presupposes that it is possible to calculate a metric that quantifies the security of the system. Given such a metric, different system designs could be ranked from *most secure* to *least secure*, which could help a system architect to make appropriate design decisions.

However, an examination of the academic literature and state-of-practice clearly shows that there is no commonly accepted overarching cybersecurity metric (in either theory or practice) that a system architect can use to rank systems (or designs) from most secure to least secure. Does this doom cybersecurity modeling? Consider the issue from a broader perspective.

Classical decision theory can help inform quantitative security modeling in its goal of helping decision makers to make better decisions by helping to define what a "better decision" means. The decision maker has a variety of preferences. We assume that the system architect

has certain preferences for the system. Preferences may include high availability, reliability, survivability, performability, security, etc. and low costs. The system's utility (from the perspective of the system architect) is a function of his or her preferences. At a high level, a common assumption in decision theory is that a rational agent has a utility function and acts as if it is trying maximize the expected value of this utility function when deciding which action to take.

Therefore, a rational system architect is expected to seek to maximize the expected utility of the system. The abstract concept of "increasing security" is not the priority for the system architect, nor should it be. Recognition and acceptance of this fact clarifies the role of cybersecurity modeling. The role of the cybersecurity modeling should not be to "increase the security" of the system (which is a poorly defined goal), but instead to help the system architect to maximize the expected utility of the system. Expected utility therefore obviates the need for an overarching cybersecurity metric that can rank designs from most secure to least secure.

Some may object to the preceding statement that the cybersecurity modeler's primary goal is to increase the expected utility of the system, rather to increase the security of the system. It may help to illustrate the point with an example. Say that a system architect must decide between two designs, A and B. A system constructed according to Design A has minimal cyber defenses and is predicted to produce revenue of \$1,000,000 per year, and suffer an expected loss of \$50,000 per year due to cyber security attacks. A system constructed according to design B has sophisticated and powerful cyber defenses, which are predicted to prevent all cyber attacks and monetary loss due to cyber attacks, but will only produce a revenue of \$900,000 per year, because the enhanced security detracts from the customer's experience and lessens the productivity of employees (e.g. annoyance of using two-factor authentication and requiring frequent password resets, increased network latency due to deep packet inspection, and resources invested in cyber defenses reduce investments that could have been used to improve other aspects of the business all potentially contribute to lessening the profitability of the system). In these circumstances, the owner/operator of the system will rationally select Design A, because it maximizes profit, even though Design A is less secure than Design B by every common sense definition of the word. A modeler specializing

in quantitative cybersecurity modeling should take care not to waste effort creating a secure design like Design B in the thought experiment, when the effort would be better spent on creating something like Design A that maximizes the utility of the system.

Careful consideration of expected utility resolves the apparent tension between system performance on the one hand and cybersecurity on the other. A cyber system that is airgapped, powered off and physically inaccessible is, in essence, both perfectly secure and perfectly useless. On the other hand, a cyber system with no defenses may have great performance, until an adversary attacks and easily commandeers or destroys the system. However, appropriate defenses will maximize the expected utility of the system. In fact, "appropriate defenses" in this context can be defined as the measures designed to prevent, detect, and respond to attacks in the way that will maximize the expected utility of the system.

Expected utility also informs a proper view of risk in cybersecurity. Risk can be defined as

$$r = l * i \tag{5.1}$$

where r is the risk, l is the expected probability of the occurrence, and i is the expected impact of the occurrence. A similar definition is

$$r = f * i \tag{5.2}$$

where r is the risk, f is the expected frequency of the occurrence, and i is the expected impact of the occurrence.

Since risk can be quantified, some may view it as the chief cybersecurity metric, and seek to mitigate it whenever they find it. However, it is almost always necessary to accept some risk to maximize the system's expected utility. Just as an investment portfolio cannot maximize expected returns without accepting some risks, similarly the utility of a system can rarely be maximized without accepting some risks. The process of modeling may reveal opportunities to reduce risk, but it may also reveal opportunities to accept more risk for outsize gains in utility. Cybersecurity modelers should not overly focus on risk, but recognize that is just one component of the unavoidable costs of building and maintaining the system. In short, cybersecurity experts should focus on risk management rather than risk mitigation.

Auxiliary Metrics

While system architects focus on maximizing expected utility, it can be useful to calculate a variety of other metrics if those metrics can help a system architect to create a system with higher expected utility. See Subsection 5.2.3 for example security metrics that can be calculated by a quantitative model built using a cybersecurity formalism like GAMES.

5.2.2 Metric Theory

Those interested in quantitative security modeling may leverage the theory of performance variables based on the reward-model formalism [86] to construct metrics for quantitative models. Performance variables accumulate reward, and the amount of reward accumulated may be examined to gain insight into some phenomenon of the model. Performance variables may be divided into two distinct classes based on how they accumulate reward: rate-based rewards and impulse-based rewards. The rate-based performance variables collect reward when the model is in a particular state, while the impulse-based performance variables collect reward when the model makes a particular type of state transition. Performance variables may be further categorized by their approach to time: instant of time, interval of time, and time-averaged interval of time. As the names suggest, an instant of time performance variable calculates the reward accumulated at a particular discrete point in time; interval of time reward variables calculate the reward accumulated during a particular interval of time; and time-averaged interval of time reward variables calculate the reward accumulated during a particular interval of time divided by the length of the interval of time. Performance variables are very flexible, and allow a great deal of freedom and creativity in the definition of cybersecurity metrics.

5.2.3 Examples of Metrics

We describe several example metrics to give a more concrete understanding of the kinds of metrics that may be calculated using quantitative models. The metrics may be divided into two broad classes. The first class consists of metrics that may be used to analyze the behavior of any type of agent in general, while the second class consists of metrics apply

more specifically to one agent type: adversary, defender, or user, for example.

First, an analyst may calculate the expected costs, expected payoffs, and expected net profit accumulated by the agent in a set time range. For example, an analyst may estimate how much more an attack would cost an adversary if the defender implemented a new security feature, or what sort of return a defender may expect on an investment in security infrastructure or employee training. Another metric that may be calculated is the number of times a particular action is attempted, and the number of times a particular sequence or chain of actions is attempted. By examining these metrics, an analyst may be able to tell the most likely attack path an adversary would take against the system, find inefficiencies in defender response, or forecast how changing the configuration of the system may influence users' actions. The analyst may also calculate the probability of a state variable having a particular value during a time interval. That value may represent the possession or control of a particular part of the cyber system. For example, an analyst may use this kind of metric to determine whether an adversary has root access on a particular machine by examining value of the state variable that represents access on that machine. By constructing several such metrics, an analyst may analyze network penetration by the adversary. Similarly, an analyst could examine whether a user's service was interrupted by examining the value of the state variable that represents access to the service provided by the defender's system. If the value of the state variable does not change for the duration of the model execution, the analyst may surmise that the user's service was not interrupted by the actions and counteractions of the adversaries and defenders.

Many metrics are generic and may be used to help the modeler understand the behavior of each kind of agent, but there are also metrics that apply more specifically to particular kinds of agents, whether adversary, defender, or user. For example, a metric may be defined that will allow the analyst to estimate the probability an adversary will remain undetected by the defender for the duration of an attack. This would allow the analyst to examine the effectiveness of different intrusion detection systems. An example of a defender-focused metric is the amount of damage the adversary does to the system being defended. An analyst could use this metric to provide aid in a number of design decisions, for example, to examine whether changing the policy used by a defender reduces or increases the expected

damage to the system. A more user-specific metric is the overhead and loss of performance a user may incur if the system is reconfigured to be more secure. For example, a bank may be interested in improving the security of online account access by implementing two-factor authentication, but may also be concerned that the change will overly frustrate their legitimate customers. Those are just a few examples of metrics that apply to one agent type that could be calculated using the GAMES formalism (or similar state-based agent modeling formalisms).

5.3 MODELING ISSUES REQUIRING SPECIAL ATTENTION

We have presented our modeling framework and a discussion of metrics in the cybersecurity modeling context, and we shall now proceed to give advice on a number of modeling issues requiring special attention, including collaboration, choosing the right level of abstraction, and proper management of modeling inputs and outputs.

5.3.1 Collaboration

Constructing and maintaining large networked cyber systems composed of heterogeneous components requires the collaboration of multiple subject matter experts. Each expert may have a deep understanding of some aspect of the system, but not fully grasp how the system as a whole will function. Quantitative models can serve as a common language used by different subject matter experts to collaborate on a shared design. A formally documented model much more easily enables collaboration compared to a mental model held in the mind of the manager who is responsible for coordinating the efforts of the subject matter experts. Multiple people can contribute their individual subject matter expertise to different aspects of the model to increase its richness and accuracy, and leverage a form of the "wisdom of the crowds" phenomenon [87] to reach a shared vision that may be more accurate than what could otherwise be achieved. Similarly, a formally documented model can also be easily examined by third-party auditors. In this way the model's assumptions can be checked and the the design can be analyzed to determine whether a system built to its specifications will be in compliance with established policies. A well-constructed, well-documented model will

also allow those charged with maintaining the constructed system in the future to check whether new technological developments or newly discovered threats affect the assumptions and results of the model (and consequently the real-world system). Quantitative models can also help newly hired security personnel to quickly understand the underlying assumptions, design choices, and policies of a system's security architecture. For these reasons, it is our recommendation that security models be created by a collaborative team of subject matter experts, and the security model should be well documented so others in the future can understand and update the model.

5.3.2 Choosing a Level of Abstraction

Choosing the right level of abstraction for the model is one of the most important decisions a modeler must make. For example, consider the construction of a security model for an enterprise network. Should the model include every packet that flows through the network, every process on every host in the system, and every read and write to every database in the system, every type of adversary that could attack the system, every individual user of the system, and the diurnal usage patterns in the network, and a hundred other details? Or should the model be highly simplified, perhaps limited to a rational adversary and a defender (each having a well-defined and highly restricted subset of actions), and a contested "resource"?[1] Or should the model's level of abstraction fall somewhere on the spectrum between these two extremes?

Fundamentally, we believe that a modeler should aim to construct the simplest model that can still produce high-quality actionable metrics. The addition of unnecessary details consumes the modeler's time and resources, makes it more difficult for others to understand, audit, and collaborate on the model, and may lead to a false sense of confidence, since the unnecessary details may obscure poor assumptions built into the model.[2] Needless complexity also makes it difficult to maintain the model over time, which limits its effectiveness as

[1] This approach is taken by a number of game-theory inspired research papers, for a prominent example, see *FLIPIT* [88].

[2] In some unfortunate circumstances, a modeler may find it expedient to include unnecessary details to the model when presenting it to others (such as superiors, funding agencies, etc.) who would distrust a model that did not include these details.

a tool that could be used in the future to help redesign the system to respond to changing circumstances.

Of course, the simplifying process can be taken too far: over simplification and lack of detail can produce a model that poorly reflects reality and may produce low quality metrics. Determining the correct level of abstraction is more of an art than a science. Modelers are encouraged to find the golden mean between too much and too little detail. Modelers should always consider how the inclusion or exclusion of detail in the model will impact the quality of the metrics of interest.

The modeling of human behavior is one aspect of quantitative cybersecurity modeling that is frequently oversimplified and unrealistic. For example, one of the oldest and most well known modeling formalisms, attack trees [40], along with the more recent ADVISE formalism [25], explicitly model only the adversary behavior, which makes it difficult to model the interaction between the attacker and the defender. Attack-defense trees [41] and attack-defense graphs [42] can be used to model the interaction between the two parties, but neglect the other human entities whose behavior may enhance or detract from the security of the system, such as trusted third party contractors, customers, employees, and law enforcement.[3] The limitations reduce the realism of models made with these formalisms and reduce their utility to modelers. Another common simplifying assumption is the rationality of the humans in the system (a small selection of papers that make this assumption include [46] [47] [48] [25]), despite the well-known fact that humans often do not behave rationally [90]. Optimizing defenses against the worst-case rational adversary may be suboptimal if the system is never actually attacked by such an unrealistic adversary.

We believe that one of the most fruitful potential research directions in the field of cybersecurity modeling is developing ways to realistically model multiple parties interacting in a cybersecurity context. Motivated by this belief, we created the GAMES formalism [3]. We believe that there exist great opportunities for researchers to make interesting and useful discoveries in the quest to accurately model human behavior in a cybersecurity context.

[3]The 2013 Target data breach, which resulted in the theft of 40 million credit card records, is a good example of a cyber security attack which cannot be described as a game between the adversary and defender. Rather, it involved multiple parties: the attackers are believed to have first compromised a trusted third-party vendor and leveraged that trust relationship to access Target's systems, and ultimately Target was first made aware of the breach not by their own security team, but by the U.S. Department of Justice [89].

5.3.3 Model Inputs

A model can be well designed but still produce inaccurate, useless results if low quality input values are used. The phrase "garbage in, garbage out" is a phrase colloquially used to describe this phenomenon. Some input values can be relatively easy to find. Unfortunately, in many cases it is not easy to obtain highly accurate input values for quantitative cyber security models, due to (1) the rapid evolution of hardware, software, and networking technologies, (2) the understandable reluctance institutions have with sharing data about cyber breaches, (3) the difficulty of performing realistic experiments, and (4) the challenge of predicting human behavior. In an ideal world, a modeler could do extensive research to determine the value of each input variable, including realistic experiments and surveys, to determine an accurate value. However, time and budget constraints often make this approach infeasible. We propose a sketch of a method the modeler may use to mitigate the concerns regarding uncertain input values.

First of all, models should be kept as simple as possible, so a modeler doesn't spend unnecessary time and resources trying to determine values for a large number of inputs that have little to no effect on the results of the model. When in doubt regarding the value of a particular input, a modeler should seek the opinion of multiple subject matter and members of the design team to provide estimates for the value [87]. If the estimates are identical or very close, the modeler may be justified in trusting the estimate as the value for the input. If, however, the estimates vary, the modeler can use the set of estimates as the basis of a sensitivity analysis. The sensitivity analysis may reveal that the metrics of interest are not sensitive to that range of input values. In that case, the disagreement among the estimates is unimportant and do not detract from the effectiveness of the model as a decision-making aid, so the modeler can use any of the estimates (or the average, or median, etc.) as the value for the input without concern. If, however, the metrics of interest in the model are sensitive to the value of the input in the range of the estimates, more effort should be focused on experiments and analyses to create as accurate an estimate as possible. If the research is inconclusive or infeasible, the whole range of model outputs given the possible set of model input values should considered in the decision making process that the model was created to support.

5.3.4 Model Outputs

The modeler should have increased confidence in the quality of the model outputs if he or she collaborates with other modelers and subject matter experts in the creation of the model, documents the model to help make the assumptions explicit, makes the model as simple as possible without ignoring important factors or making unreasonable assumptions, and carefully selects input values, as we previously suggested. We believe that if those responsible for making impactful cyber security decisions build a model following these guidelines and incorporate the model results into their decision making, it would be a huge improvement compared to the current state-of-practice of relying solely on the intuition, experience, and ad hoc mental models of individual cyber security experts.

One model alone, even a well-constructed model, may not give the desired level of confidence to a decision maker. One way to increase confidence is to use ensemble modeling techniques, which have been successfully employed in other fields which make predictions about large complicated non-deterministic systems that are difficult to fully understand, such as climate modeling [91]. In the case of cyber security, a decision maker may gain additional insight by instructing multiple independent teams to each produce a model, and then comparing the models and their results with one another. If the models are in substantial agreement, the decision maker will have increased confidence in their results and recommendations. If the results do not match, the models and assumptions can be examined to determine the cause, and the inferior model (and its results) may be discarded, or the best features of the models may be merged together to produce a superior model. Ensemble modeling can be achieved using only one team (instead of multiple teams) if that team builds multiple different models (e.g. by using different modeling formalisms or different sets of reasonable starting assumptions).

If the models are being used to help guide the system design process, it may be advisable to have a design cycle of multiple mutually enriching stages that progress between modeling, emulation, and implementation, similar to the model-test-model process[92]. Our proposal is illustrated in Figure 5.1. First, a large number of small, simple models, each representing a different design, can be created and analyzed. After analyzing the large set of simple models, a smaller subset of the most promising model designs can be further developed into larger,

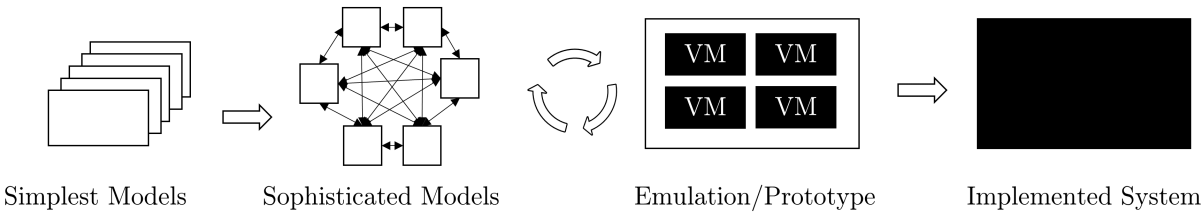


Figure 5.1: An illustration of a method to progressively validate and improve models.

more realistic models. Small-scale emulations of the best performing of the more realistic models can be made to support experiments that will confirm the model results, perhaps using virtual machines, testbeds or other similar technology that can cheaply emulate a real cyber system. Comparing the model and the emulation side by side may validate the model or reveal ways to improve the model. The models can be improved if necessary, and then new emulations and accompanying experiments can be run to further exercise and validate the improved model. The process can be applied iteratively until the system architect is satisfied and the first prototype of the system is made. The design cycle described above can be labor intensive and expensive, but may be useful in certain circumstances, particularly with the design of especially critical and expensive cyber infrastructure.

5.3.5 Summary of Advice

We have given many specific points of advice, but in general, one should pursue diversity and multiple perspectives whenever possible (whether it is different people, different models, or different methods to achieve a task) and compare them to see the underlying unity. One should avoid the false comfort that needless complexity brings: things should be as simple as possible, but no simpler. Above all, modelers should keep the goal of quantitative security modeling in mind: producing metrics that help decision-makers. With the advice given in this section we hope modelers in the future will be able to avoid or overcome common issues in cybersecurity modeling.

CHAPTER 6: CONCLUSIONS

Frequent successful attacks on important cyber infrastructure demonstrate the need for care in designing, maintaining, and insuring secure infrastructure. Quantitative cybersecurity modeling can help decision makers make better decisions so they can create and sustain useful cyber infrastructure that is performant and secure. Cybersecurity modeling has many benefits, but a number of practical challenges make it difficult in practice. Arguably, quantitative modeling is less practiced and developed in the cybersecurity field compared to other engineering disciplines such as civil and mechanical engineering.

In our dissertation, we aimed to address several of the most important challenges to quantitative cybersecurity modeling. We identified four important requirements for such modeling:

1. It should be usable without special modeling expertise.
2. It should model the correct things.
3. It should be able to validate the model results.
4. It should be guided by a reasonable framework.

Within each requirement, we identified a key challenge:

1. The first challenge is to construct cybersecurity models despite the fact that most cybersecurity domain experts lack modeling expertise, and can be overwhelmed with the effort and time needed to build a useful model.
2. The second challenge is to model the complex interplay between the cyber system and the humans that interact with it in a way that is intuitive, modular, and easy to use. Many current security modeling formalisms only consider the adversary's perspective, or only directly model the adversary and defender, excluding the other human agents that impact the security of the system (though other human parties often have a huge impact on the security of the system). These parties include customers, vendors, users, law enforcement, and the media. How should all of these disparate entities be modeled?

3. The third challenge is model validation given given uncertain input variables and long model execution times which make traditional sensitivity analysis and uncertainty quantification techniques unfeasible in many cases.

4. The fourth challenge is the identification of a framework that can guide the modeling process and selection of appropriate security metrics.

In this chapter, we shall review our contributions to addressing these key challenges, present directions for future work, and end with concluding thoughts.

6.1 REVIEW OF CONTRIBUTIONS

The first challenge to cybersecurity modeling we identified is the difficulty of creating models by hand. Our contribution to address the challenge is an ontology-assisted automatic model generation approach that can convert a user-created system diagram into a sophisticated executable cybersecurity model. We helped implement the approach in the Möbiusmodeling tool. We demonstrated its effectiveness with an AMI case study. We created a custom AMI-focused ontology that the generation algorithm could utilize to translate the user-created system diagram representing an AMI instance into a custom cybersecurity model. Using the ontology approach, a modeler could create a complex cybersecurity model in hours rather than days. The ontology shifts much of the needed modeling expertise from the average modeler to the creators and maintainers of the ontology. Since the ontology can be reused to study different AMI deployment scenarios across the industry, the modeling expertise that goes into the ontology can likewise scale and be reused in a way that was not previously possible. The ontology-assisted automatic model generation is a significant step towards addressing the first challenge and helps modelers make models faster and more easily.

The second challenge we identified is that none of the popular cybersecurity modeling formalisms allowed for the explicit modeling of all of the human players of the cybersecurity game: adversaries, defenders, users, customers, law enforcement, media, and others. While most security modeling formalisms allow either the adversary or the adversary/defender pair to be modeled explicitly, few allowed an arbitrary number of human entities to be modeled,

despite the impact other humans have on the security of the system. Our contribution is the creation of a new modeling formalism called GAMES that enables the creation of composable agent submodels to allow all human entities involved in cybersecurity scenarios to be modeled explicitly in a modular and intuitive way. We implemented a prototype tool to demonstrate the GAMES approach, and showed its utility with a case study involving an account security scenario that explicitly modeled an adversary, defender, customers of different types, and the media and their interaction with one another and the system. Our contribution of a novel modeling formalism that allows all of the humans in a cybersecurity scenario to be modeled is an important step towards addressing the second challenge.

The third challenge is that sensitivity analysis and uncertainty quantification are useful tools to understand and validate models, but traditional approaches to SA and UQ are often infeasible for cybersecurity models, due to the large number of uncertain input variables and long execution times. To address the challenge, our contribution is an exploration of indirect metamodel-based approaches to SA and UQ. In particular, we are the first to design and apply the stacking ensemble technique to metamodeling. We demonstrate with a botnet case study, an AMI IDS case study, and 6 PRISM models that stacking-based metamodeling produces metamodels that are significantly faster than the original model and can be accurate enough to be a useful standin for sensitivity analysis and uncertainty quantification purposes. Our contribution helps address the challenge by giving modelers a practical approach to help them explore and validate models that would otherwise run too slowly to be analyzed by traditional methods.

The fourth contribution address the challenge of a lack of (1) cohesive pragmatic and effective modeling frameworks, (2) guidance on the fundamentals of metrics, and (3) practical advice on overcoming common modeling issues. Our contribution includes a high-level framework to guide the modeling process, concrete suggestions on the creation of useful metrics, and practical advice on common modeling issues such as choosing the correct level of abstraction and detail, facilitating collaboration among domain experts, and validation of model results. Our contribution in this chapter will hopefully help modelers in the future to quickly build effective models that are useful aids to decision makers while avoiding common pitfalls.

6.2 FUTURE WORK

We recommend directions that can be taken in the future to further address the identified challenges.

The ontology-assisted automatic model generation approach can be extended in several ways. The ontology and generation algorithm can be enhanced in such a way as to enable the automatic generation of other kinds of models (not just security models), such as models of availability, reliability, and performance. Second, we only created an AMI-focused ontology, ontologies for other critical infrastructure and enterprise domains should be created. A common publicly-available repository or library of ontologies should be created and maintained to increase adoption. Different methods for testing, validating, and updating ontologies should explored, to ensure that the ontologies correctly codify expert knowledge, and to ensure that expert knowledge actually corresponds to reality.

Similarly, the GAMES formalism could be further developed to become an even more useful tool for modelers. The current prototype tool is limited, there are many ways it could be made more powerful and easier to use. Templates for common agent types could be developed (currently each agent must be made by hand). Different common agent decision algorithms could be implemented, inspired by artificial intelligence, game theory, agent based modeling, psychology, or other fields. We believe that policy-space response oracles [93] could be an especially fruitful research direction for developing appropriate agent-decision algorithms for the GAMES formalism. The addition of a graphical user interface (GUI) to the prototype tool would make it easier to create and study GAMES models. Finally, larger and more complex case studies should be developed using the GAMES formalism to exercise its capabilities and identify further areas for improvement.

The metamodeling approach provides rich opportunities for further research. Perhaps the most important issue to research is to determine how closely the sensitivity analysis and uncertainty quantification results obtained from the metamodel correspond with the results from the base model. We showed that the results do correspond with the AMI case study, but further research is needed to determine if the results generalize to other models. In addition, while we did explore a number of different committee and filter types, further research could

be done to explore in this direction to design the most effective stacked metamodel. Finally, we have not parallelized any of the code, though we believe that it would be relatively easy to parallelize and could significantly speed (1) training and testing data collection and (2) metamodel training.

Last, we recommend that more research be done in the area of metrics. We believe it is important for researchers to learn what metrics current practitioners in the field are using, and whether those metrics are serving decision makers (e.g. designers and maintainers of systems) well. It may be that currently popular metrics may not give particularly good insight into the system or support for decision making that will boost the utility of the system. If that is the case, a future effort could try to bridge the gap between theory and practice to improve metrics.

6.3 CONCLUDING REMARKS

In this dissertation, we addressed some of the most significant challenges facing quantitative cybersecurity modeling. We developed an AMI-focused ontology and automatic model generation approach so modelers can quickly and easily create sophisticated cybersecurity models from simple system diagrams without much prior modeling expertise. We created the GAMES modeling formalism to explicitly model all of the human entities involved in cybersecurity scenarios (not just the adversary or adversary and defender). We developed and tested a novel stacking-based metamodel approach to do much faster sensitivity analysis and uncertainty quantification than would be possible with traditional approaches. Finally, we provided a framework to guide the modeling process, presented a practical philosophy of metrics, and gave advice on common issues encountered while doing cybersecurity modeling. The research presented here can help modelers to develop useful quantitative models that support decision makers in their quest to develop and maintain useful and secure cyber systems.

REFERENCES

[1] M. Rausch, K. Keefe, B. Feddersen, and W. H. Sanders, "Automatically generating security models from system models to aid in the evaluation of AMI deployment options," in *International Conference on Critical Information Infrastructures Security*. Springer, 2017, pp. 156–167.

[2] K. Keefe, B. Feddersen, M. Rausch, R. Wright, and W. H. Sanders, "An ontology framework for generating discrete-event stochastic models," in *Computer Performance Engineering*, R. Bakhshi, P. Ballarini, B. Barbot, H. Castel-Taleb, and A. Remke, Eds. Springer International Publishing, 2018, pp. 173–189.

[3] M. Rausch, A. Fawaz, K. Keefe, and W. H. Sanders, "Modeling humans: A general agent model for the evaluation of security," in *Proc. International Conference on Quantitative Evaluation of Systems*. Springer, 2018, pp. 373–388.

[4] M. Rausch and W. H. Sanders, "Sensitivity analysis and uncertainty quantification of state-based discrete-event simulation models through a stacked ensemble metamodel," in *Proceedings of the International Conference on Quantitative Evaluation of Systems*, vol. 12289. Springer, 2020, pp. 276–293.

[5] M. Rausch and W. H. Sanders, "Stacked metamodels for sensitivity analysis and uncertainty quantification of ami models," in *Proceedings of the 2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–7.

[6] M. Rausch and W. H. Sanders, "Evaluating the effectiveness of metamodeling in emulating quantitative models," in *Proceedings of the International Conference on Quantitative Evaluation of Systems*, vol. 12846. Springer, 2021, pp. 127–145.

[7] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the cyber attack on the Ukrainian power grid," *SANS Industrial Control Systems*, 2016.

[8] M. Buratowski, "The DNC server breach: who did it and what does it mean?" *Network Security*, vol. 2016, no. 10, pp. 5–7, 2016.

[9] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[10] S. Cowley, "2.5 million more people potentially exposed in equifax breach," October 2017, accessed: 2019-01-31. [Online]. Available: https://www.nytimes.com/2017/10/02/business/equifax-breach.html

[11] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1093–1110.

[12] D. Temple-Raston, "A 'worst nightmare' cyberattack: The untold story of the solarwinds hack," April 2021. [Online]. Available: https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack

[13] T. C. of Economic Advisers, "The cost of malicious cyber activity to the u.s. economy," February 2018. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2018/02/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf

[14] C. Herley and P. C. van Oorschot, "Science of security: Combining theory and measurement to reflect the observable," *IEEE Security Privacy*, vol. 16, no. 1, pp. 12–22, January 2018.

[15] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, "Rre: A game-theoretic intrusion response and recovery engine," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 395–406, 2013.

[16] C. Biener, M. Eling, and J. H. Wirfs, "Insurability of cyber risk: An empirical analysis," *The Geneva Papers on Risk and Insurance-Issues and Practice*, vol. 40, no. 1, pp. 131–158, 2015.

[17] A. Marotta, F. Martinelli, S. Nanni, A. Orlando, and A. Yautsiukhin, "Cyber-insurance survey," *Computer Science Review*, vol. 24, pp. 35–61, 2017.

[18] R. Betterley, "The betterley report: Cyber/privacy insurance market survey - 2018," June 2018. [Online]. Available: https://www.irmi.com/docs/default-source/publication-tocs/betterley-report—cyber-risk-market-survey-june-2018-summary.pdf

[19] A. Setalvad, "Demand to fill cybersecurity jobs booming," March 2015. [Online]. Available: http://peninsulapress.com/2015/03/31/cybersecurity-jobs-growth/

[20] M. Ford, K. Keefe, E. Lemay, W. Sanders, and C. Muehrcke, "Implementing the advise security modeling formalism in Möbius," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, June 2013, pp. 1–8.

[21] Möbius Team, *Official Möbius Documentation*, University of Illinois at Urbana-Champaign, Urbana, IL, 2014. [Online]. Available: https://www.mobius.illinois.edu/wiki/

[22] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 273–284.

[23] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 336–345.

[24] M. G. Ivanova, C. W. Probst, R. R. Hansen, and F. Kammüller, *Transforming Graphical System Models to Graphical Attack Models*. Cham: Springer International Publishing, 2016, pp. 82–96. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-29968-6_6

[25] E. LeMay, "Adversary-driven state-based system security evaluation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2011. [Online]. Available: https://www.perform.illinois.edu/Papers/USAN_papers/11LEM02.pdf

[26] M. Rausch, "Determining cost-effective intrusion detection approaches for an advanced metering infrastructure deployment using advise," M.S. thesis, University of Illinois at Urbana-Champaign, 2016.

[27] C. Gellings, "Estimating the costs and benefits of the smart grid: A preliminary estimate of the investment requirements and the resultant benefits of a fully functioning smart grid," Electric Power Research Institute, Tech. Rep., March 2011.

[28] L. Alejandro, C. Blair, L. Bloodgood, M. Khan, M. Lawless, D. Meehan, P. Schneider, and K. Tsuji, "Global market for smart electricity meters: Government policies driving strong growth," 2014. [Online]. Available: https://www.usitc.gov/publications/332/id-037smart_meters_final.pdf

[29] D. Grochocki, J. Huh, R. Berthier, R. Bobba, W. Sanders, A. Cardenas, and J. Jetcheva, "Ami threats, intrusion detection requirements and deployment recommendations," in *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, Nov 2012, pp. 395–400.

[30] Federal Bureau of Investigation Cyber Intelligence Section, "Smart grid electric meters altered to steal electricity," *International Journal of Semantic Computing*, 2010. [Online]. Available: http://krebsonsecurity.com/wp-content/uploads/2012/04/FBI-SmartMeterHack-285x305.png

[31] V. Badrinath Krishna, K. Lee, G. A. Weaver, R. K. Iyer, and W. H. Sanders, "F-DETA: A framework for detecting electricity theft attacks in smart grids," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016, pp. 407–418.

[32] M. Ward, "Smart meters can be hacked to cut power bills," October 2014. [Online]. Available: http://www.bbc.com/news/technology-29643276

[33] M. Rausch, B. Feddersen, K. Keefe, and W. H. Sanders, *A Comparison of Different Intrusion Detection Approaches in an Advanced Metering Infrastructure Network Using ADVISE*. Cham: Springer International Publishing, 2016, pp. 279–294. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-43425-4_19

[34] J. Eloff, L. Labuschagne, and K. Badenhorst, "A comparative framework for risk analysis methods," *Computers & Security*, vol. 12, no. 6, pp. 597 – 603, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016740489390056B

[35] D. W. Straub and R. J. Welke, "Coping with systems risk: security planning models for management decision making," *MIS quarterly*, pp. 441–469, 1998.

[36] M. H. Manshaei, Q. Zhu, T. Alpcan, T. Bacşar, and J.-P. Hubaux, "Game theory meets network security and privacy," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 25:1–25:39, July 2013. [Online]. Available: http://doi.acm.org/10.1145/2480741.2480742

[37] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE, 2010, pp. 1–10.

[38] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *CoRR*, vol. abs/1303.7397, 2013. [Online]. Available: http://arxiv.org/abs/1303.7397

[39] V. Verendel, "Quantified security is a weak hypothesis: A critical survey of results and assumptions," in *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, ser. NSPW '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1719030.1719036 pp. 37–50.

[40] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner, "Toward a secure system engineering methodolgy," in *Proceedings of the 1998 Workshop on New Security Paradigms*, ser. NSPW '98. New York, NY, USA: ACM, 1998. [Online]. Available: http://doi.acm.org/10.1145/310889.310900 pp. 2–10.

[41] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Attack–defense trees," *Journal of Logic and Computation*, vol. 24, no. 1, pp. 55–87, 2012.

[42] J. Kramer, "Attack-defence graphs: On the formalisation of security-critical systems," M.S. thesis, Saarland University, 2015. [Online]. Available: https://www-old.cs.uni-paderborn.de/uploads/tx_sibibtex/main_01.pdf

[43] F. Baiardi, F. Corò, F. Tonelli, A. Bertolini, R. Bertolotti, and L. Guidi, *Security Stress: Evaluating ICT Robustness Through a Monte Carlo Method*. Springer International Publishing, 2016, pp. 222–227. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-31664-2_23

[44] V. Gorodetski, I. Kotenko, and O. Karsaev, "Multi-agent technologies for computer network security: Attack simulation, intrusion detection and intrusion detection learning," *International Journal of Computer Systems Science & Engineering*, vol. 18, no. 4, pp. 191–200, 2003.

[45] N. Wagner, R. Lippmann, M. Winterrose, J. Riordan, T. Yu, and W. W. Streilein, "Agent-based simulation for assessing network security risk due to unauthorized hardware," in *Proceedings of the Symposium on Agent-Directed Simulation*, ser. ADS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2872538.2872541 pp. 18–26.

[46] S. Izmalkov, S. Micali, and M. Lepinski, "Rational secure computation and ideal mechanism design," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, Oct 2005, pp. 585–594.

[47] X. Z. You and Z. Shiyong, "A kind of network security behavior model based on game theory," in *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, Aug 2003, pp. 950–954.

[48] J. Grossklags, N. Christin, and J. Chuang, "Secure or insure? a game-theoretic analysis of information security games," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367526 pp. 209–218.

[49] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[50] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *Proceedings of the International Conference on Timed Petri Nets*, Torino, Italy, July 1985, pp. 106–115.

[51] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications, special issue on Computer-Aided Modeling, Analysis, and Design of Communication Networks*, vol. 9, no. 1, pp. 25–36, Jan. 1991.

[52] M. Risdal, "Stacking made easy: An introduction to StackNet by competitions grandmaster Marios Michailidis (KazAnova)," http://blog.kaggle.com/2017/06/15/stacking-made-easy-an-introduction-to-stacknet-by-competitions-grandmaster-marios-michailidis-kazanova/, accessed: 2019-12-13.

[53] E. V. Ruitenbeek and W. H. Sanders, "Modeling peer-to-peer botnets," in *Proc. 2008 Fifth International Conference on Quantitative Evaluation of Systems*, Sep. 2008, pp. 307–316.

[54] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[55] R. L. Iman, J. C. Helton, and J. E. Campbell, "An approach to sensitivity analysis of computer models: Part i – introduction, input variable selection and preliminary variable assessment," *Journal of Quality Technology*, vol. 13, no. 3, pp. 174–183, 1981.

[56] I. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.

[57] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[58] A. Cunha, R. Nasser, R. Sampaio, H. Lopes, and K. Breitman, "Uncertainty quantification through the Monte Carlo method in a cloud computing setting," *Computer Physics Communications*, vol. 185, no. 5, pp. 1355–1363, 2014.

[59] I. Sobol, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, no. 1, pp. 271–280, 2001.

[60] M. D. Morris, "Factorial sampling plans for preliminary computational experiments," *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991.

[61] F. Campolongo, J. Cariboni, and A. Saltelli, "An effective screening design for sensitivity analysis of large models," *Environmental Modelling and Software*, vol. 22, no. 10, pp. 1509–1518, 2007.

[62] B. Iooss and P. Lemaître, "A review on global sensitivity analysis methods," in *Uncertainty Management in Simulation-Optimization of Complex Systems: Algorithms and Applications*, G. Dellino and C. Meloni, Eds. Boston, MA: Springer US, 2015, pp. 101–122.

[63] A. Razmjoo, P. Xanthopoulos, and Q. P. Zheng, "Online feature importance ranking based on sensitivity analysis," *Expert Systems with Applications*, vol. 85, pp. 397–406, 2017.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[65] J. Herman and W. Usher, "SALib: An open-source Python library for sensitivity analysis," *The Journal of Open Source Software*, vol. 2, no. 9, January 2017. [Online]. Available: https://doi.org/10.21105/joss.00097

[66] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 2002, pp. 200–204.

[67] "Prism tutorial: Circadian clock," accessed: 2021-10-28. [Online]. Available: https://www.prismmodelchecker.org/tutorial/circadian.php

[68] N. Barkai and S. Leibler, "Biological rhythms: Circadian clocks limited by noise," *Nature*, vol. 403, pp. 267–268, 2000.

[69] J. Vilar, H.-Y. Kueh, N. Barkai, and S. Leibler, "Mechanisms of noise-resistance in genetic oscillators," *Proc. National Academy of Sciences of the United States of America*, vol. 99, no. 9, pp. 5988–5992, 2002.

[70] "Workstation cluster," accessed: 2021-10-28. [Online]. Available: https://www.prismmodelchecker.org/casestudies/cluster.php

[71] B. Haverkort, H. Hermanns, and J.-P. Katoen, "On the use of model checking techniques for dependability evaluation," in *Proc. 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, Erlangen, Germany, October 2000, pp. 228–237.

[72] "Cell cycle control in eukaryotes," accessed: 2021-10-28. [Online]. Available: https://www.prismmodelchecker.org/casestudies/cyclin.php

[73] P. Lecca and C. Priami, "Cell cycle control in eukaryotes: A BioSpi model," in *Proc. Workshop on Concurrent Models in Molecular Biology (BioConcur'03)*, ser. Electronic Notes in Theoretical Computer Science, 2003.

[74] "Embedded control system," accessed: 2021-10-28. [Online]. Available: https://www.prismmodelchecker.org/casestudies/embedded.php

[75] J. Muppala, G. Ciardo, and K. Trivedi, "Stochastic reward nets for reliability prediction," *Communications in Reliability, Maintainability and Serviceability*, vol. 1, no. 2, pp. 9–20, July 1994.

[76] "Kanban manufacturing system," accessed: 2021-10-29. [Online]. Available: https://www.prismmodelchecker.org/casestudies/kanban.php

[77] G. Ciardo and M. Tilgner, "On the use of Kronecker operators for the solution of generalized stocastic Petri nets," Institute for Computer Applications in Science and Engineering, ICASE Report 96-35, 1996.

[78] "Simple molecular reactions," accessed: 2021-10-29. [Online]. Available: https://www.prismmodelchecker.org/casestudies/molecules.php

[79] O. S. Saydjari, *Engineering Trustworthy Systems*. McGraw Hill Education, 2018.

[80] D. Stamatis, *Failure Mode and Effect Analysis: FMEA From Theory to Execution*. ASQ Quality Press, 2003.

[81] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, "Security application of failure mode and effect analysis (fmea)," in *Computer Safety, Reliability, and Security*, A. Bondavalli and F. Di Giandomenico, Eds. Cham: Springer International Publishing, 2014, pp. 310–325.

[82] U. D. of Energy, "Cybersecurity capability maturity model (c2m2), version 1.1." February 2014, accessed: 2019-01-31. [Online]. Available: https://www.energy.gov/sites/prod/files/2014/03/f13/C2M2-v1-1_cor.pdf

[83] S. Kaplan, Y. Y. Haimes, and B. J. Garrick, "Fitting hierarchical holographic modeling into the theory of scenario structuring and a resulting refinement to the quantitative definition of risk," *Risk Analysis*, vol. 21, no. 5, pp. 807–807, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/0272-4332.215153

[84] G. Macher, H. Sporer, R. Berlach, E. Armengaud, and C. Kreiner, "Sahara: A security-aware hazard and risk analysis method," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2755753.2755894 pp. 621–624.

[85] C. Herley and P. C. v. Oorschot, "Sok: Science, security and the elusive goal of security as a scientific pursuit," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 99–120.

[86] W. H. Sanders and J. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," in *Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems*, A. Avizienis, H. Kopetz, and J. Laprie, Eds. Springer-Verlag, 1991, pp. 215–237.

[87] J. Surowiecki, *The wisdom of crowds*. Anchor, 2005.

[88] M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "Flipit: The game of stealthy takeover," *Journal of Cryptology*, vol. 26, no. 4, pp. 655–713, 2013.

[89] X. Shu, K. Tian, A. Ciambrone et al., "Breaking the target: An analysis of target data breach and lessons learned," *arXiv preprint arXiv:1701.04940*, 2017.

[90] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," in *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, 2013, pp. 99–127.

[91] W. S. Parker, "Ensemble modeling, uncertainty and robust predictions," *Wiley Interdisciplinary Reviews: Climate Change*, vol. 4, no. 3, pp. 213–223, 2013.

[92] B. Mansager, "Model test model." Naval Postgraduate School Monterey CA Dept. of Mathematics, Tech. Rep., 1994.

[93] M. Lanctot, V. F. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," *CoRR*, vol. abs/1711.00832, 2017. [Online]. Available: http://arxiv.org/abs/1711.00832