Mental Models for Cybersecurity: A Formal Methods Approach

by

Adam Michael Houser
May 25, 2018

A dissertation submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
In partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

Department of Industrial and Systems Engineering

Dedication

To the giants upon whose shoulders I now stand.

Acknowledgments

I would like to thank my advisor and committee chair, Dr. Matthew L. Bolton, for his guidance, advice, and support. I would not be the scientist I am today without him.

I would also like to thank my committee members, Dr. Ann M. Bisantz and Dr. Jun Zhuang. Their feedback and guidance throughout this process has been invaluable, and both are role models to whom I will look up for the rest of my life.

Others in the department also deserve gratitude, including all of the Human Factors faculty members and Steve Hart, Deanie Hedrick, Noelle Matthews, and Diane Porter.

I would also like to thank William C. Elm, Samantha Szymczak, Paul W. Durbin, and the Duckling Class of 2015 at Resilient Cognitive Solutions, with whom I learned to be an engineer.

Though I graduated in 2010, faculty from my undergraduate university are continually on my mind and deserve special thanks. These include my undergraduate advisor, Dr. Matthew Stolick, who taught me the value of critical thought and the necessity of full engagement; Drs. Nicole Diederich and Ronald J. Tulley, with whose guidance I became a writer; and Dr. DeBow Freed, whose dignity and leadership I try to emulate yet today.

More gratitude goes to my colleagues and friends for their time, support, and engaging conversation: Theresa Guarrera, Woodrow Gustafson, Sudeep Hegde, Nicolette McGeorge, Kylie Molinaro, David T. LaVergne, Meng Li, Judith Tiferes, and Joshua S. Ziegler, among many others. I would also like to thank Mahiyar Nasarwanji for his advice at an inflection point in my academic career.

I would also like to thank Click & Clack the Tappet Brothers, as well as ABGT, for their companionship on drives to campus.

Finally, and most importantly, I would like to thank Alyssa for her patience, love, and support. The stress, time, and occasional frustration of the dissertation process would have been unbearable without you.

Contents

List of Figures

List of Tables

List of Symbols

A	The "All" path quantifier.
E	The "Exists" path quantifier.
F	The "Future" temporal quantifier.
G	The "Global" temporal quantifier.
U	The "Until" temporal quantifier.
X	The "Next" temporal quantifier.
$\neg$	The "negation" logical connective, where $\neg\ X$ is read as "not X."
$\Rightarrow$	The "implication" logical connective, where $X \Rightarrow Y$ is read as "if X then Y."
$\wedge$	The logical conjunctive, where $X \wedge Y$ is read as "X and Y."
$\vee$	The logical disjunctive, $X \vee Y$ read as "X or Y."
ϕ	A temporal logic proposition about a formal model state or path.
ψ	A temporal logic proposition about a formal model state or path.

Abstract

Failures in complex systems often result from problematic interactions between different system components, including human users. Furthermore, the complexity of these systems means that system designers may fail to anticipate component interactions, particularly where rare and undesirable human-automation interaction can significantly impact the safe and effective operation of these systems. These problematic and often unanticipated interactions may be driven in part by the mental models of system users, where incongruousness between user mental models—which can drive the actions users take within a given system—and functional system characteristics can lead to inappropriate or unsafe operational states.

This is particularly true with respect to cybersecurity vulnerabilities. Employees of most organizations are recommended or required to use the complex digital resources that are now essential for modern business operations, including externally-facing email accounts, internet access, and filesharing capabilities. In response, threat actors have increasingly begun to eschew the highly technical exploits once used to "hack" these systems in favor of attacks that instead target users themselves, taking advantage of this confusion to trick them into voluntarily executing malicious code inside their network and bypassing sophisticated cybersecurity countermeasures. Because network defenders have traditionally relied on employee cybersecurity training and restricting network access privileges to counter this threat, these attacks and their consequences continue to grow.

One potential explanation for this growth is that the increasing complexity of our digital tools is making it difficult for users to understand how they work and the impacts of using computer programs in certain deleterious ways. These tools can be hard to understand and

interact with, particularly for those unfamiliar with the tool or for which controls presented by the system interface do not tightly correspond with the underlying system actions. If this is true, then attackers could certainly use this lack of understanding to their advantage, particularly if system defenders have also failed to anticipate how a user's lack of understanding can inadvertently make a system more vulnerable to attack. We therefore need strategies for investigating these misunderstandings and their impacts on cybersecurity.

Decades of work in human factors engineering have resulted in powerful methods for exploring and improving the performance of complex systems, but the use of these methods for discovering rare and unanticipated human-automation interactive effects is difficult, time-consuming, and may not be robust to the discovery of rare events. One potential solution to this problem is the use of formal methods, a well-defined collection of mathematical languages and techniques used to prove whether system models do or do not support desirable properties. Model checking is a formal methods technique that leverages exhaustive state-space search to mathematically prove the presence or absence of these properties, regardless of their rarity. This capability offers a powerful approach to the discovery of unanticipated human-automation interaction, and the identification of these interactions can offer specific targets for improving the safety and reliability of complex digital systems. Human-automation interaction researchers have successfully explored ways of using formal methods and mental model concepts in safety critical systems. In these analyses, proof-based analyses were used to determine if dangerous mismatches exist between human mental models and models of system automation. However, these techniques have never been applied to the cybersecurity domain.

This work shows that it is possible to extend formal mental modeling techniques with cybersecurity-specific concepts and computer security folk models to explore and discover unanticipated human-automation interactions resultant from mismatches between mental models and system characteristics that can lead to computer security failures. These findings could be used to improve system security by suggesting improvements to the behavior of the system or components of the user interface. Two cases are used in this investigation. The first explores configuration errors committed by users of a popular cloud data storage service, using a generic formal modeling framework to explore the actions that users take and discuss the vul-

nerabilities introduced into the system by users with certain mental models. The second case extends the initial generic framework by integrating folk modeling concepts in cybersecurity applications. We investigate a phishing email attack example, then use our findings to describe how users with different folk models could commit potentially dangerous actions in response to threats posed by malicious emails.

Results indicate that users can commit actions leaving them vulnerable to cybersecurity risks, such as unauthorized access to data stored in the cloud or a computer compromised by an opened phishing email. While this may not be surprising to computer security professionals, our results offer a novel perspective on these problems by describing how unexpectedly dangerous consequences can occur through specific actions that may appear innocuous to the user. Furthermore, we find that effects may seem disconnected from causes, potentially making it more difficult for users to associate one with the other and understand the origin of these dangers. By rigorously characterizing these risks and their provenance, results suggest that our method sheds significant light on the unintended cybersecurity consequences of real-world user actions, offering a way to anticipate and potentially remediate hazardous system conditions before they can be exploited by threat actors.

1 | Introduction

> Are people the solution or are people the problem to be solved?
>
> *David D. Woods & Erik Hollnagel*
> *Joint Cognitive Systems:*
> *Patterns in Cognitive Systems Engineering*

1.1 Motivation

The conveniences and opportunities afforded to those living in this 21st century are made possible in part through the continued operation of sprawling, interconnected complex systems. Energy production at nuclear power facilities, wind and solar farms, hydroelectric dams, and oil and gas fields that can be monitored and controlled from hundreds of miles away (Finnan & Melrose, 2017). Digital communication has revolutionized global commerce, allowing for unprecedented wealth creation through the trade of goods and services, as well as the exchange of value through international stock markets (Hendershott, 2004). Intercontinental air travel provides a means of extremely safe transportation coordinated with hundreds of air traffic control towers, thousands of airplanes, and millions of passengers daily within United States airspace alone (Federal Avaiation Administration, 2017). The rise of these and other systems have improved the lives of most modern humans, and they depend on a tight coupling between system elements to function effectively.

Though the efforts of engineers and system designers to improve system safety have been

successful in some domains, such as aviation, humans still contribute to the failure of complex systems in part because the ways in which human-automation interaction leads to system failure can be hard to anticipate (Taleb, 2007). This is true when successful task execution relies on the user's mental model of system operational characteristics and actions, particularly where users have had little opportunity (through practice, training, or reading, among other strategies) to develop an appropriate mental model. The potential for problematic human-automation interaction may therefore rise as a consequence of miscalibrations between operator mental models and system characteristics.[1]

Given both the difficulty and importance of anticipating how these miscalibrations may contribute to problematic human-automation interactions and the failure of complex systems, the need for techniques that facilitate the discovery and investigation of these interactions becomes clear. One potential path forward is the use of formal methods, and specifically model checking, to discover the possibility of these rare interactions before they occur (Baier & Katoen, 2008). This is made possible through model checking's capability for exhaustive statespace search, whereby the combinatoric interaction between all system elements—including those characterized by the mental models of users—are explored for undesirable properties (Oliveira, Palanque, Weyers, Bowen, & Dix, 2017). This deliberate, systematic exploration of the statespace means that even rare and unanticipated interactive events can be discovered. These findings can also inform strategies to improve system safety through, for example, targeted system improvements or revisions to training protocols. Model checking is therefore particularly well-suited for these complex and difficult analytic efforts.

One area of imminent, significant concern is cybersecurity. As the interconnected web of digital systems managing our health, wealth, and national security continues to diversify and increase in complexity, so too does the requirement for their security and integrity. Failures of these complex systems can be devastating, for example by halting healthcare services to thousands of patients (UK National Audit Office, 2017) or costing companies hundreds of millions of dollars in lost profit (A.P. Møller-Mærsk A/S, 2017). As organizations roll out improved

[1] Note that this does not necessarily imply that useful or effective mental models must exactly match system characteristics (see Chapter 6).

threat detection and intrusion prevention systems to bolster cybersecurity, hackers have begun targeting humans rather than hardware or software. By using unsuspecting employees to open malware-laden emails or execute other hidden attacks that compromise systems from the inside, hackers can bypass sophisticated perimeter network defenses and more easily accomplish the same objectives (IBM, 2016; Symantec, 2016; Verizon Enterprise, 2016).

If attackers are exploring the human component in the execution of cyber attacks, then we must also explore this human component in cyber defense. One potential way forward is an examination of user mental models, focusing specifically on the methods by which attackers can exploit miscalibrations between user mental models of cybersecurity threats and the complex digital systems on which they operate. While some of these efforts have produced rather loose qualitative results (see, for example, Asgharpour, Liu, & Camp, 2007; Camp, 2009; Kang, Dabbish, Fruchter, & Kiesler, 2015), others have used empirical techniques to gather data and comment on user mental representations of computer risks (Wash, 2010). Still others have gone a step further by introducing formal representations of these findings and using them for more rigorous analysis (Blythe & Camp, 2012). This work aims to go one step further by extending compatible techniques from the formal methods community, bringing to bear methods from formal mental modeling and human-automation interaction analysis to discover vulnerabilities introduced through unanticipated user actions and suggest remediations before attackers can exploit them. This has never been done before and offers a promising new area of exploration that could significantly benefit modern approaches in cybersecurity, human factors engineering, and formal methods.

1.2 Research Objectives

This work investigates the feasibility of discovering unanticipated cybersecurity risks by identifying, formalizing, and exploring the interactions between user mental models and various computer programs using formal methods techniques. By discovering and rigorously characterizing these interactions within the context of real-world digital systems, we can suggest improvements to specific elements of the human-computer interaction space and the system

architecture to address these risks and improve the cybersecurity of target systems.

To that end, our research objectives are of both practical importance to the cybersecurity community and of theoretical significance to work in the human factors and formal methods communities. There are in total four broad objectives:

1. **Establish** a rigorous framework for exploring user mental models of computer security vulnerabilities with formal methods techniques;
2. **Demonstrate** that these vulnerabilities correspond to vulnerabilities that attackers could realistically exploit in a modern cloud data storage system;
3. **Extend** this framework to integrate folk modeling techniques and investigate user responses to phishing email attacks; and
4. **Analyze** data collected from multiple sources to validate our results and demonstrate that situations revealed by the model correspond to real-world conditions.

Meeting these objectives will help researchers understand the utility of formal methods techniques in exploring user mental models and refine the approach that practitioners take towards securing humans in modern digital systems.

1.3 Methodological Approach

Meeting our research objectives will require an approach broadly divided into two distinct efforts: our preliminary effort establishing a framework for formally instantiating runnable mental models within a cybersecurity context, and a secondary effort extending our method to integrate folk modeling concepts. Real-world data will be used to validate the findings resultant from both components.

1.3.1 Method Formulation and Cloud Data Storage Use Case

We will formulate our method generically, establishing a framework that combines the *error* and *blocking* state architecture of Degani and Heymann (2002) with mental modeling concepts from formal methods and human factors engineering. After developing patterns of specifications that demonstrate the utility of the method, we will investigate a popular cloud data storage system for potentially dangerous cybersecurity conditions arising unforeseen from user

actions. We will validate our analysis results by reproducing these dangerous situations within the target system, then discuss the implications of our findings.

1.3.2 Extension to Folk Modeling Techniques and Email Phishing Use Case

We will then extend our generic formulation, adapting work on folk models of computer security threats initially conducted by Wash (2010) and integrating this work into the generic method. Work will also focus on refining the appropriate specifications from Part I that capture system behaviors and characteristics desirable from a cybersecurity perspective. A proof-of-concept formal analysis on our framework will then be conducted, exploring an email phishing attack scenario and comparing the actions of users with different folk models to different phishing attack strategies. Comparisons will also be made between the results of our analysis and phishing threat actor tradecraft that has been observed in the wild. A discussion of our results will follow.

1.4 Organization of this Dissertation

Chapter 1 has provided an introduction to this problem and motivated the need for its exploration within the context of a digital and highly-connected 21st century. It has also outlined the dissertation's research objectives and discussed in broad strokes how those objectives will be met.

Chapter 2 provides a thorough review of the literature that underpins our work. This chapter surveys perspectives on mental models, folk models, and their uses from the human factors community, emphasizing the differences between them and the utility of both paradigms for exploring human-automation interaction. It also discusses work from the formal methods community on contemporary issues in modeling human-automation interaction, highlighting strategies for investigating mental models and their findings. The third section of this chapter reviews literature from the cybersecurity community on the modern threat landscape, discussing the role that user-focused attacks play in cybercrime and the importance of securing

humans in digital systems, rather than focusing only on the systems themselves. This chapter will conclude with a discussion of issues identified in the literature review.

Chapter 3 provides an overview of this work's research questions and the research objectives that will serve as milestones for measuring progress towards answering the research questions, describing how they relate to conclusions drawn from the literature review.

Chapter 4 describes our formulation of a method integrating Degani's *error* and *blocking* state architecture with formal methods techniques and cybersecurity concepts. We formulate generic formal specification patterns, then translate these into assertions that can be model checked to reveal rare, hard to anticipate system configurations that could result in unintentional system vulnerabilities. We use our method to explore potential security implications of configuration decisions made within a popular cloud data storage environment, validating our findings by reproducing them within the target system. We conclude with a discussion of our results.

Chapter 5 extends the generic method presented in Chapter 4 to integrate folk modeling concepts, explaining the method undertaken for translating data published regarding folk models of computer security vulnerabilities and threats into a runnable formal representation. We discuss the architecture of the model, its components, and the specifications developed to identify potentially undesirable behaviors as users with different computer security folk models respond to phishing email attacks. We discuss the findings of these investigations and how they align with real-world attacker strategies, then validate our findings using a large corpus of real-world data on phishing attack strategies and methods of compromise. We conclude with a discussion of our results.

Chapter 6 will discuss the implications of our results, the contributions of this work to the human factors, formal methods, and cybersecurity communities, and suggest directions of future work.

Chapter 7 will conclude the work.

The Appendices offer supplementary information. Appendix A provides the full model code used in Chapter 4, and Appendix B provides the full model code used in Chapter 5.

The Bibliography provides an index of all literature cited throughout the work.

2 | Surveying the Relevant Literature

> Any journey into new territory must build upon the past explorations of others, and draw upon the knowledge and guidance of current adventuresome experts.
>
> *Gerd Gigerenzer & Peter Todd*
> *Simple Heuristics that Make Us Smart*

2.1 Overview

This chapter provides a review of the literature underpinning our research effort and includes work from a diversity of domains, including human factors engineering, formal methods, and cybersecurity. After a discussion of work on mental models conducted within the human factors, applied cognitive psychology, and other communities, the review will then identify a selection of the most dangerous contemporaneous issues in cybersecurity that face modern computer users. We will move to explore the synergies between research conducted in these communities, laying the foundation for our initial developments of a cybersecurity modeling framework. This review will then address techniques from the formal methods community that offer promising strategies to explore issues in user-facing cybersecurity threats, with particular focus on past successes in similar domains and the suitability of formal methods techniques for

our work. The chapter will conclude with a discussion of several problems identified through the literature review.

2.2 Exploring Diverse Perspectives on Mental Models

2.2.1 Historical Perspectives on the Development of Thought in Mental Models

Humans have been thinking about thinking since at least the Greeks, when Socrates discussed (and Plato eventually recorded) the nature of knowledge and representation in the *Theaetetus* sometime around 370 B.C.E. Here, a young mathematician named Theaetetus is led by Socrates to see that "true" knowledge of the World can only be had if one possesses a "justified true belief with an account" of some thing or event (Plato, 2007). For example, one can truly understand the function of a simple machine only once she has come to verify its method of function through deduction (which provides the *justification* for the reasoner to regard her internalization of its function as *true*, as a deductive conclusion must be true since the reasoner has verified that all premises are true) and can then completely articulate how the machine works to someone else. She therefore has an account of the nature of the machine. In modern readings of the classic, some argue that this articulation of an account relies on one possessing some sort of mental model of the machine; however, the compatibility between these views and those articulated in the human factors engineering literature is potentially dubious (Reynolds, Sinatra, & Jetton, 1996).

Perspectives on mental models in human factors engineering hew closer to a more structured interpretation presented in modern works. One could consider the work of Craik in the 1940s to be a reasonable point of entry into the discussion. In "The Nature of Explanation," he writes:

> If the organism carries a 'small-scale model' of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and the future, and in every way

> to react in a much fuller, safer, and more competent manner to the emergencies which face it.[1] (Craik, 1943, p. 61)

Here, Craik draws the reader to a discussion of representations of processes or events (for example, tidal calculators that encoded gravitational fluctuations into an externalized tool, or flak cannon gun sights that encoded complex Newtonian transformations into an easy-to-use aiming reticle), leveraging these notes into an argument for the same process happening *mentally* (p. 57). These mental models serve as internal representations of external worlds.[2] The method by which these representations were formed, and what structure they took, has since been the subject of intense work across a variety of fields. It also helped seed more modern discussions of cognitive artifacts, interface design, and decision support in complex systems, to be addressed below.

After Craik, those involved in cognitive science, and later human factors engineering, began adopting and working with mental models in the 1970s. In his chapter "Mental Models and Thought," Philip Johnson-Laird provides a history of development in mental models organized by theoretical approach (Johnson-Laird, 2005). For example, some work concerns the use of predicate logic to crudely model and describe basic concepts of fluid flow (Hayes, 1989). Other work advocates for the construction of mental three-dimensional representations as models of selective problem spaces (Marr, 1982), using these arguments to study how people used such "mental depictions" to reason about simple machines (Schwartz & Black, 1996) and for use in fault detection activities (Rouse & Hunt, 1986), among other topics. Yet more work spends time with iconic strategies for model construction, leveraging work in token-type representation (Johnson-Laird, 1970; Johnson-Laird, Byrne, & Schaeken, 1992) and formal semantic constructs (Kamp, 2008) to formulate mental models about the state of the world. These developments are not field-specific, however, and in fact encompass a number of disciplines and

[1] Elsewhere in this chapter he writes of "three essential processes" that occur during the process of reasoning: "Translation of external processes into words, numbers, or symbols; Arrival at other symbols by a process of reasoning, deduction, inference, etc., and; Re-translation of these symbols into external processes (as in building a bridge to a design) or at least recognition of the correspondence between these symbols and external events" (Craik, 1943, p. 51). There seems an interesting symmetry here between Craik and Plato's writing in the *Theaetetus*.

[2] Craik also seems to foreshadow the notion of cognitive artifacts, writing about the importance of "instruments which [have] extended the scope of our sense–organs, our brains [and] our limbs" (p. 61).

Table 2.1: Domains Citing "Mental Models" in their Applications

Author	Domain
DeKeyser, 1986	Industrial process control operators
Dutton & Starbuck, 1971	Scheduling in discrete manufacturing
Gentner & Stevens, 1983	Simple physical causal systems
Hoc, 1995	Aviation and process control
Rasmussen, 1986	Industrial fault diagnosis
Sanderson, 1990	Electronic troubleshooting
Sheridan, 1976	Supervisory control
Woods & Roth, 1988	Cognitive systems engineering
Young, 1969	Complex perceptual-motor skills

Note. This table has been partially adapted from Table 8.1 in Moray, 1999, p. 224.

their forays into what each may locally call "mental models."

Within the human factors literature, there has been an effort to consolidate meaning and more rigorously define what is meant when discussing mental models. Several researchers have done a significant amount of work in this effort. For example, Moray argues that "the term 'mental model' is widely used, and its meanings vary just as widely," casting a broad net across the literature from other fields to highlight the variegation (much in the same way that Johnson-Laird did above) (Moray, 1999, p. 224). Table 2.1 is partially adapted from Moray and identifies a selection of different contexts in which "mental models" have been used to discuss subject behavior, thinking, and performance.

Though impressive in its diversity of application, one should not assume this diversity to signify that these or other researchers and practitioners mean the same thing without unpacking the term "mental model," as Moray cautions above.

Others have been more forceful in their approaches, trying to formulate theoretical constructs around which the field could gather or arguing that such agreement has not been shown. We will therefore move to discuss the variety of perspectives on mental models in the human factors literature, then conclude the section by discussing different perspectives on mental models, citing examples from literature to make the case for each.

2.2.2 Functional Approaches to Mental Models

Don Norman, in "Some Observations in Mental Models," unpacks the term "mental model" and provides a common *lingua franca* for those interested in human factors engineering perspectives on mental models (Norman, 1983). Norman writes that people invested in discussing mental models must actually consider four different aspects that are sometimes blurred by those less careful with their thinking:

- The target system: the system a person is learning or using;
- The conceptual model, later (Norman, 2013) called the "design model": a contrivance of the target system, "invented by teachers, designers, scientists, and engineers" to help provide users an accurate, consistent, and complete representation of the target system in the system image;
- The mental model: A user's internal representation of the function of the target system; and
- The scientist's conceptualization: the scientist's model of the user's mental model (Norman, 1983, p. 7).

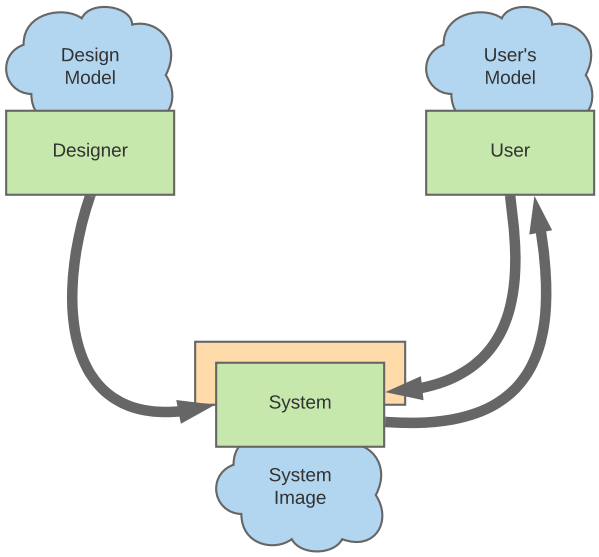

Figure 2.1: The three components of Norman's mental models. Note that the "design model" is equivalent to the "conceptual model" in Norman, 1983. Adapted from Norman, 2013, Figure 7.1.

Norman uses these distinctions to constrain his discussion of mental models to the third item. He argues that these models are often incomplete, unscientific, and can harbor "superstitious" beliefs about how the user thinks the target system works (p. 8). For example, users

operating basic four-function calculators often did not completely encapsulate all of its functions in their mental models of calculator operation, instead writing down intermediate totals on paper rather than using the calculator's built-in memory function, or perhaps pressing the "clear" button multiple times because it made them feel more comfortable about its operation. Norman also writes that mental models are unstable, with users forgetting seldom-used system details and dropping them from their mental models, as well as imprecisely defined, blending operational concepts together into a sort of gestalt about the target system's operation that may or may not be correct (p. 8).

Another key aspect for Norman is the ability for users to run their mental models, or for mental models to possess predictive power. He writes that, "The purpose of a mental model is to allow the person to understand and anticipate the behavior of the physical system [...] in other words, it should be possible for people to 'run' their models mentally" (p. 12). This capability is critical, because without it the user may not be able to respond to changing system conditions; for Norman this is perhaps a hallmark of the usefulness of mental models in the world (p. 13). He does note in his observations on mental models, however, that peoples' abilities to run their own models can sometimes be "severely limited" (p. 8). Discussion of these aspects will resurface later in this work due to their compatibility with and relevance to concepts in formal methods.

Finally, Norman argues that there should be "a direct and simple relationship" between the system's image (generally speaking, the interface presented to the user to enable interaction with a target system) and the user's mental model (pp. 12–13). This implies that there should also be a direct and simple relationship between the target system and the user's mental model, mediated by the system image (and therefore based upon a congruent conceptual model). If an accurate representation of the possible target system states is reflected in the system image, and the user is able to construct a mental model that accurately reflects the deeper structure and interaction among elements of the conceptual model, then system changes mentally run, anticipated, and enacted by the user will accurately reflect system changes within the target system (p. 13). Such agreement is critical for a variety of pursuits within human factors engineering, including effective decision support, teaming with automation, and joint cognitive

system design (Hollnagel & Woods, 2005; Woods & Hollnagel, 2006), as well as in ecological interface design (see Wickens, Hollands, Banbury, & Parasuraman, 2015, chap. 4). It may also be important for select issues in cybersecurity, which shall be discussed later in this work.

What results from Norman is the perspective of mental models as a mental structure that reflects the user's understanding—however incomplete, incorrect, or misaligned—of the physical system, "acquired either through observation, instruction, or inference" (p. 12). However, this perspective could be somewhat problematic if one attempts to describe mental models with more rigor. Are mental models visual representations in the mind's eye attempting to mirror an approximation of the physical system? Is it more abstract, similar to a lattice network or state transition diagram used to run scenarios and reach conclusions? Are these representational strategies even knowable, or do they make any difference in how people formulate and make use of mental models? We must turn to other work for perspectives on these questions.

2.2.3 Strategies for Representing Mental Models

Wilson and Rutherford (1989) offer insight here. They pose an unsparing Moray-like review of the then current state of work specific to mental models in human factors engineering, agreeing that "Norman's observations that mental models are incomplete, unstable, nonexclusive, and unscientific—among other attributes—might also characterize the whole enterprise of applying them, judging by the literature" (p. 617). In their survey of mental model literature from both human factors and cognitive psychology, a litany of views are presented.

For example, authors such as Rasmussen (1986), Rouse and Morris (1986), and Carroll, Anderson, Olson, Council, et al. (1987) present arguments that mental models can be seen as descriptive system abstractions, and importantly all three seem to agree that a multitude of mental models can be used and switched between according to the needs of the reasoner (Wilson & Rutherford, 1989, p. 620).[3] The authors also argue that mental models have no clearly defined structures, but rather (by virtue of use of the term *abstraction*) may be a network of functional concepts, or even further abstractions.

[3]This is also reflective of Moray (1999)'s general remarks about mental model use and description of the literature; see p. 252 for further explanation.

Others notions are explored, including the proposition that mental models can take the forms of "pictures in the mind" (Wilson & Rutherford, 1989, p. 625). Rouse and Hunt (1986) concede that it is "reasonable to suggest that mental models are frequently pictorial or image-like" (p. 359), and de Kleer and Brown (1981) argue for the same type of pictorial representation based upon their explorations of user mental models of an electromagnetic buzzer.

From Wilson and Rutherford we see that, at least until the late 1980s, there existed several competing uses and theoretical constructs across which the term "mental models" had been used. From these constructs, deeper undercurrents or research thrusts can be discovered. One major thrust includes a discussion of the structure of mental models, such that they can be created and deployed to aid with system operation and anticipatory action (Moray, 1999, p. 227).

For example, consider the view that mental models could be seen as pseudo-visual constructs or analogies; work by de Kleer and Brown (1981), Gentner and Gentner (1983), Clement (1983), and Williams, Hollan, and Stevens (1983), among others, is useful here. At play is the notion that physical interaction can drive mental model formulation, particularly where the notion of physical interaction is more approachable and relatable (see Sanderson, 1990 for a more detailed exploration of adjusting task difficulty by manipulating the physical layout of the task space).

This stands in contrast to mental model representation as a more abstract process, particularly if the introduction or use of the target system cannot be easily mapped to physical constructs. Consider, for example, work by Dutton and Starbuck (1971) on "Charlie's run-time estimator," where—even though Charlie's startlingly effective model of fabric ripper operation is based on a physical plant—it did not seem to closely mirror that of the plant itself, instead seeming to take a range of abstract inputs about the manufacturing process to develop solutions to complex fabric production and machine scheduling problems.[4] At issue here are also mental models of abstract concepts such as team performance and communication (Mathieu, Heffner, Goodwin, Salas, & Cannon-Bowers, 2000), robot capabilities and function (Kiesler &

[4]One could take issue here with Charlie's capabilities of verbal recall, as some research indicates that people are notoriously bad at correctly describing their own procedures or mental constructs.

Goetz, 2002), and even humble calculators (Norman, 1983).

Why the question of mental model representation may be important for this work, and why this represents an important deep-structure differentiator within the study of human factors, has two components. First, we maintain the variety of uses and meanings of the term "mental model" mentioned in Moray (1999) and Wilson and Rutherford (1989). It is easy to see the shallow differences between papers (with respect to experimental protocol and results, among other factors), but we feel there is an interesting perspective to be had if one steps back and examines the results or central themes of papers in context of target system, conceptual model, and mental model representation strategies. Namely, if these strategies are (even weakly) related to the abstract or concrete nature of interaction with the target system, this may impact the success of user reasoning, troubleshooting, and predictive capability with the mental model; mismatches could result in reduced effectiveness, leading to erroneous behavior or other deleterious effects. This issue is thoroughly discussed with respect to mental models in cybersecurity later in the dissertation.

Second, these differences can have significant influence on system design and user interaction with a given system. This is particularly relevant in light of Norman (1983)'s contention that mental models can be acquired "...either through observation, instruction, or inference" (p. 12) (see section 2.2.4, below, for more details). It is important to consider these factors when attempting to parse existing literature on mental models as they specifically relate to cybersecurity, as well as any potential solutions to these problems that could result directly from this work.

However, one should note that these differences in constructs should not be taken as competing for correctness. In other words, even though deeper differences in mental model representation strategies (resembling a physical construct as compared to a more ethereal or abstract representation) could result in very different constructs, one is not necessarily more appropriate than the other. Users should not strive to build a mental model that relies on physical analogies for every task, for example. Alternatively, it may be possible to express a mental model rooted in the physical experience through the use of state transition diagrams and other formalisms.

Furthermore, work suggests that users are likely able to toggle back and forth between mental models (and therefore disparate representation strategies) depending upon the demands of the task at hand (Rasmussen & Rouse, 2013). Expertise *vis-á-vis* the use of mental models could mean that expert users may possess mental models with several different levels of abstraction. One potential way expert behavior may manifest is the ability to rapidly switch between mental models to find the one most appropriate (Sanderson, 1990; Wickens et al., 2015; Williams et al., 1983). A great deal of work has been done on the differences between expert and novice users within the human factors literature, including on questions of problem solving and knowledge representation (Chi, Feltovich, & Glaser, 1981) and how these differences can manifest themselves in design requirements (Day & Boyce, 1993). These and other works therefore lend support to the notion of a continuum of abstraction among mental model representation techniques, and that expertise likely has a significant role in the mediation of this continuum.

Though work has been conducted on a variety of use cases and topics within human factors engineering, at their most fundamental level, "mental models research is [...] concerned with understanding human knowledge about the world" (Gentner & Stevens, 1983, p. 1). The intent of every mental model is to codify, in some type of representation, an abstraction about the way a given system works such that the modeler may understand the present state of the system and attempt to anticipate a given future state (Moray, 1999). From a less simplistic yet equally foundational perspective, mental models encapsulate a variety of approaches to knowledge representation, though they may take disparate forms depending on a range of characteristics about the use and purpose of mental models, as discussed above (Johnson-Laird et al., 1992; Moray, 1999).

2.2.4 The Utility of Mental Model Representational Strategies in Human-systems Interaction

Based on the analysis above, we would like to focus briefly on the mental representation of these models and their uses in a small selection of applied engineering problems. Ecological interface design offers an area of congruence, whereby information displays are integrated

and arranged such that they impact the user's mental model of the information space, among other things (Bennett, Posey, & Shattuck, 2008; Vicente & Rasmussen, 1992). In other words, improving agreement between the user's mental model of the information space and the information space itself will generally improve task performance when using an information space so designed (Hollnagel & Woods, 2005). Norman may characterize this agreement as one between the *system image* and the *user's mental model* (Norman, 1983, pp. 12–13).

The design of decision support systems (DSS) can benefit from well-grounded ecological interface design and, therefore, congruence between the conceptual model, system image, and mental models of a target system. For example, DSS are designed to support operator decision making under high-stress, high-risk, time-pressured circumstances (Militello, Klein, Lee, & Kirlik, 2013). These constraints on DSS mean that users should ideally spent as little time as possible fighting the system, instead devoting available cognitive capacity to fighting the engagement (in other words, completing the task at hand). Here, congruence between conceptual and mental models is beneficial, in that it can reduce the cognitive burden of translating between mental and system images (Hollnagel & Woods, 2005, chap. 3).

Consideration of user mental model representation is also a potentially useful paradigm to explore responses to cybersecurity threats. Our work explores situations where computer users may have difficulty anticipating cybersecurity threats, or perhaps may take incorrect actions and end up making themselves less secure. Attempting to learn more about their representational strategies may be beneficial. For example, the mental model of a particular computer user faced with a potential threat may be internally coherent but not correspondent with the target system (the actual effects of system settings on an email client, for example), perhaps resulting in settings mismatches that seem appropriate but are actually dangerous. Alternatively, the user may not be able to sufficiently "run" the mental model to begin with, resulting in a lack of anticipatory capability and remaining susceptible to security risks. We will probe these and other issues in this work.

2.3 Selected Current Problems in Cybersecurity

Cybersecurity is a global security risk and lucrative business opportunity in this 21st century, and market forecasts indicate that security firms and their services will continue to grow for the foreseeable future. Gartner, an internet and technology advisory group, put worldwide cybersecurity market spending at $75 billion in 2015 (Gartner, 2015), and that spending is projected to increase to at least $170 billion per year by 2020 (Morgan, 2016). A quarterly report by Cybersecurity Ventures, an organization releasing market sizing and projection information for Fortune-listed companies, projects total global expenditures over a five-year period between 2017 and 2021 will top $1 trillion (Cybersecurity Ventures, 2016). These and other reports also forecast an expected 12% to 15% annual compounded growth, with significant portions of this profit directed to organizations focusing on threat intelligence, mobile security, and cloud security (Morgan, 2016).

This spending is being driven by equally explosive growth in cybercrime. Analyses from Lloyd's of London, a British insurance provider specializing in high-risk and obscure coverages, estimate that in 2015 alone cybercrime cost businesses as much as $400 billion in direct damages and lost revenue (Gandel, 2015). This is projected to nearly quintuple, with estimates by Juniper Research analysts pegging total expenditures to surpass $2 trillion by 2019, a jump due to the anticipated surge in e-commerce, cloud data storage, and internet-connected business and enterprise communications posing lucrative targets for threat actors (Juniper Research, 2015). Others consider this projection too low: Satya Nadella, CEO of Microsoft, argued in 2015 that more complete evaluations put the actual cost at $3 trillion, meaning that the Juniper Research figure projected for 2019 may significantly understate the threat (Morgan, 2016).

While cybercrime has a diverse array of outcomes, such as data theft, fraudulent financial transactions, stolen intellectual property, or software or hardware destruction, hacking often plays a significant role in the execution of cybercrime (*FBI: Cyber Crime*, 2016).[5] In these cases, unauthorized entities gain access to computer systems through a variety of different

[5] It should be noted that hacking is not *always* required for cybercrime. Examples where hacking may play a less important role include cyberbullying, cyberstalking, and harassment, among others.

methods. Though hacking in the early days of computer technology usually entailed guessing passwords or finding clever security and system workarounds, more modern software- and hardware-based techniques can be incredibly complex. For example, approaches exist to exploit vulnerabilities in web applications (Adobe Flash, Java, XSS attacks); internet information transfer protocols (DDoS attacks, UDP amplification attacks); wired and wireless routers (ARP spoofing); networked printers (LDAP attacks); database management software (SQL injection attacks); credential management software (golden ticket attacks); boot-level hardware attacks (cold boot and Evil Maid attacks); machine-side credential scraping attacks against users themselves (keylogging malware, Mimikatz and other RAM scraping attacks); and "nuclear option" zero-day attacks (Shamoon, Shamoon 2, and Project Aurora, among others) (Kim, 2015). There are yet others not mentioned here.

2.3.1 Technological Responses to Cyberthreats

In the face of this incessant dance back and forth between vulnerability discovery and patching, a host of technologies and techniques have been developed to deter and recover from attacks that leverage these vulnerabilities. Consider, for example, the use of perimeter defense and program-level countermeasures in service of network defense (Rashid, 2015). These strategies can be highly automated, such as using firewalls, antivirus scanners, malware scanners, spam filters, and intrusion detection systems to monitor computer networks using a host of advanced technologies to thwart malware, unauthorized access attempts, and other dangerous events (Obregon, 2015). Program-level countermeasures address the problem in a slightly different way, implementing network-wide rules about the types of files that computers can execute, how to handle specific events and processes, and ensuring that certain actions can only be taken by uses with the appropriate amount of authority. Together, these and other techniques help to comprise a defense in depth strategy (Paloma, 2007). Though skill, time, and money are required to roll out effective automated network defense protocols, they can be powerful tools that enable quick response times to detected intrusions.

2.3.2 Leveraging Humans to Bypass Network Defenses

One of the consequences of improvements in automated network defense is the move by hackers to quieter, harder-to-detect attack vectors that probe humans rather than network protocols for points of failure. Both of the use cases discussed below are problematic because they can bypass network defense-in-depth, perimeter defenses, and automated intrusion detection utilities in place to defend against cyberattacks, providing attackers with direct access to the internal networks of a target corporation (Collett, 2014). Exploits delivered via email or other human-controlled vectors avoid the need to find security flaws in programs or network defense protocols, and compromised or insecure personal devices can create a more direct line for attackers into the heart of an organization. Once internal network access is granted, attackers can attempt to move laterally through the system in search of their target or important user credentials to escalate their own privileges. Furthermore, both of these situations take advantage of human targets that may be unsuspecting in their victimhood, providing the added benefit of surprise to their network exploits. A user may consider themselves unlikely to be phished at work or may not consider attacker use of personal devices to exploit workplace networks. However, industry research and cybersecurity threat modeling have indicated that attackers are beginning to adopt these strategies in force. It is dangerous for users to labor under these assumptions any more.

2.3.2.1 Email Phishing: The Target Becomes the Unwitting Accomplice

Information gathered in cybersecurity incident reports by Verizon Enterprise (2016), IBM (2016), Symantec (2016), and others indicate that exploit trends are in fact beginning to capitalize on user-enabled attacks. These reports indicate positive trends for phishing attacks, where human users are "tricked into opening an infected email attachment or browsing to a malicious website disguised as a trusted destination, where they provide information that can be used to access a system or account, or steal their identities" (IBM, 2016, p. 18). Attacks discussed in cybersecurity research under the umbrella of "phishing" include simple phishing,[6]

[6]Sending plausible but generic malicious emails to many people, with the hopes that a percentage of recipients click.

spearphishing,[7] and whaling. [8]

As previously described, phishing emails usually ask the user to download an attachment, provide a fake login screen into which the user can enter their credentials, or navigate to a hostile website and download something undesired. In all cases, the attacker's goal is to use these attacks to gain access to the target system using stolen credentials, malware implants, or other techniques (Hong, 2012). Furthermore, using phishing emails as a method of attack has become a rather effective tools for doing so. For example, phishing email attacks were estimated to cost the United States $2.4 billion in monetary thefts, costs for addressing breaches, and future business losses in 2014 alone (Microsoft, 2014). Furthermore, the rate and severity of phishing attacks continue to grow: the Anti-Phishing Working Group, for example, found a 65% increase in phishing attacks from 2015 to 2016 (APWG, 2017), and cybersecurity incident reports continue to identify phishing as one of the top threats facing corporate internet users (Cofense, 2016; Symantec, 2016; Verizon Enterprise, 2016).

Because of this growth, organizations have attempted to implement some remediations and training programs to inform users of phishing email risks and provide tips for how to spot them. However, empirical research suggests that the long-term effectiveness of training programs in reducing clickthrough, or "clicker recidivism," is mixed. For example, Kumaraguru et al. (2009) found that their subjects retained comparable victimization rates up to 28 days after training, while Sheng, Holbrook, Kumaraguru, Cranor, and Downs (2010) found differences in the effectiveness of different training styles for reducing clickthrough rates, but did not explore retention. Interestingly, Siadati, Palka, Siegel, and McCoy (2017) found that phishing training exercises only reduced clickthrough rates for specific types of phishing emails (those that included persuasive messages for the victim), whereas clickthrough rates for non-persuasive emails were generally unchanged. These and other findings suggest that phishing emails are a significant problem and offer a worthwhile target for deeper exploration.

[7] Sending customized malicious emails to specific targets.
[8] Targeting high-ranking leaders or executives with malicious emails.

2.3.2.2 Online Storage: Exploiting Dangerous User Configuration Decisions

Other methods of compromise encapsulated in this trend also include system misconfiguration and user device vulnerability. Here, the user of a particular program or service may have selected settings or configurations that enhance productivity or seem safe, but actually make the device or software less secure. For example, users of Microsoft Outlook may have their "read pane" enabled at all times, which enhances productivity by reducing the amount of time it takes to read through emails: it may seem much easier to read the emails that open in a dedicated portion of the computer screen by single–clicking on it, rather than double–clicking on the email, opening it in a pop–up window, reading it, and then closing the window before moving on to another email. However, users with the read pane enabled risk automatically executing malware hidden in embedded images[9] or code associated with the email, as these users are prone to skipping a step normally used to scrutinize the email details (header information, sender, language in the subject line, and other points of interest). What is initially configured as a productivity enhancement can actually reduce the user's system security, consequently making it easier for phishing compromise to potentially occur.

2.3.3 The Future of Cybersecurity will Depend on Human Users

Both of the aforementioned scenarios are problematic because they bypass network defense-in-depth, perimeter defenses, and automated intrusion detection utilities in place to defend against cyberattacks, providing attackers with direct access to the internal networks of a target corporation (Collett, 2014). Exploits delivered via email or other human-controlled vectors avoid the need to find security flaws in programs or network defense protocols, and compromised or insecure personal devices can create a direct line for attackers into the heart of an organization. Once internal network access is granted, attackers can attempt to move laterally through the system in search of their target or important user credentials to escalate their own privileges. Furthermore, both of these situations take advantage of human targets that may be unsuspecting in their victimhood, providing the added benefit of surprise to their network

[9] Attacks that utilize malicious code buried in images that are executed when the image is rendered by a browser or operating system are called steganographic attacks.

exploits. A user may consider themselves unlikely to be phished at work or may not consider attacker use of personal devices to exploit workplace networks. However, industry research and cybersecurity threat modeling have indicated that attackers are beginning to adopt these strategies in force.

If attackers are exploring the human component in the execution of cyberattacks, then we must also explore this human component in cyber defense. One potential way forward is an examination of user mental models, particularly if attackers can take advantage of the inadequacy, incompleteness, or inability of users to properly "run" their mental models of security threats to anticipate potential vulnerabilities (Norman, 1983). Since there do exist bodies of work in both areas that could be useful in exploring the human component of cybersecurity attacks, there may be potential ways to synergistically explore them and lend new, actionable insight on a class of problems that is only expected to grow.

2.4 Research at the Intersection of Mental Models and Cybersecurity

Exploring issues in cybersecurity from within the context of user mental models is a relatively new development popularized after the year 2000. This does not mean that concerns over cybersecurity are also relatively new; in fact, the progression of these concerns is rather disjoint. In Warner's "pre-history" of cybersecurity, he catalogs its major milestones and writes that "the 'cyber'[security] issue is not new at all, but rather has taken a half-century to develop" (2012, p. 782). Here, Warner (also p. 782) argues that the recognition of cybersecurity as a major challenge has evolved along the following timeline:

- Computers can spill sensitive data and must be guarded (1960s);
- Computers can be attacked and data stolen (1970s);
- We can build computer attacks into military arsenals (1980s and 1990s);
- Others might do that to us – and perhaps already are (1990s).

For Warner, the paradigm shift came in 1997 with the Department of Defense's ELIGIBLE

RECEIVER internal exercise, which tested the government's ability to coordinate an interdepartmental response to a coordinated cyberattack (p. 797). The results of this exercise laid bare the deficiencies of the federal response mechanism to these attacks, resulting in an immediate surge in interest and funding for solving these problems and better coordinating an effective government response to cybersecurity incidents (p. 799; Graham, 1998).

Below, we discuss a summary of literature regarding the direct ways in which mental models have been used to study cybersecurity and is relevant to our work, as well as some assumptions about mental models upon which our work rests.

2.4.1 Diverse Perspectives in Cybersecurity

There exists a moderate amount of literature regarding mental model use in categorizing and responding to security risks. L. Jean Camp conducted a series of literature reviews and compiled a list of five mental models that had been used by others to communicate security and privacy risks to users (Camp, 2004). Camp argued that these mental models characterize security failures, and users lean on them to think about security events and failures (p. 109). They include:

- Physical security models (bypassing locks, gates, fences, or law enforcement to attack);
- Medical or public health models (worms, viruses, infections, diseases that spread from host to host);
- Criminal models (attackers are out to commit digital theft and burglary);
- Warfare models (attackers must bypass firewalls, network perimeters, and fight digital system defenders); and
- Market models (attackers are out to steal data to sell, or revenue loss through web page takedowns and commerce disruption).

Later, Asgharpour et al. (2007) attempted to validate these findings through the use of card-sorting tasks, wherein subjects presented with 66 words were asked to sort them into one of six categories (five corresponding to the five Camp mental models above, and one "I don't know" category) with which they seemed to best relate. The authors asked several questions.

First, do mental models implicit in the computer security literature (which, ostensibly, characterize the five mental models described by Camp) actually correlate with the mental models of experts or non-experts in this domain? Second, do the mental models of cybersecurity experts designing risk communication opportunities correspond with the mental models of lay users who would receive these risk communications? And finally, how sensitive is the correspondence between expert and novice mental models to an operational definition of expertise, if it exists (Asgharpour et al., 2007, p. 2)? One of the implicit hypotheses is that the authors would find support for the subjects' self-reported characterizations as either experts or novices and shed light on these questions.

The results of the card-sorting tasks led the authors to several conclusions. First, both experts and novices expressed much weaker support for the public health model of computer security risks than the other models, leading the authors to conclude that the "medical mental model is not well matched to the mental models of [either experts or novices]" (p. 5). Second, by mapping word associations gleaned from the card-sorting tasks on multidimensional scaling plots, the authors claimed that the decrease in heteroschedasticity across plots—moving from novice to expert—suggests that the expert mental models were more tightly focused on specific metaphors characterizing security risks, as opposed to novice mental models which were more ill-defined (p. 6). The authors therefore concluded that risk communication strategies between experts and novices were dissimilar, and these strategies should embed the mental models of the novices who would be absorbing these communications, rather than leveraging the mental models of the experts designing the communications (p. 7). The authors suggested that the warfare model was most apt for expert users, while novice users were most likely to gravitate towards the criminal and physical security metaphors. These differences could potentially reduce the effectiveness of cybersecurity risk communications.

There do exist potential weaknesses within these works. First, Camp makes no mention of the methods used to compile the initial list of five mental models in her 2004 article, nor are they discussed in a follow-on reprint (Camp, 2009). While the author makes it clear that these were gathered from the literature, details are very light and include no discussion of her metastudy or metareview techniques. This lack of methodological information makes it

difficult to support the veracity of her claims, upon which her work and that of other authors rest. This is perhaps manifest in the findings that some models only weakly corresponds to the empirical evidence marshalled by Asgharpour et al. (2007).

Another potential weakness is that these mental models may be insufficiently robust to be considered distinct from one another. For example, the criminal and physical security models may be tightly coupled, a point that Camp concedes (2009, p. 43). This is not surprising, particularly *vis-á-vis* Norman (1983)'s contention that mental models can have indistinct boundaries. However, later treatment by other researchers seem to neglect this fuzzing. This could be problematic if these results are used in more rigorous applications such as exploration with formal methods.

2.4.2 Wash's Approach to Folk Models of Cybersecurity Risks

Rick Wash, a researcher who explores the folk models of home computer users as they relate to various digital security risks, has used a battery of carefully designed multi-round semi-structured interviews to collect data and elicit folk models of cybersecurity risks (Wash, 2010). Folk models are "mental models that are not necessarily accurate in the real world, thus leading to erroneous decision making, but are shared among similar members of a culture" (p. 1; see also d'Andrade, 1987 and Dekker & Hollnagel, 2004). Ultimately, Wash found his subjects to rely on eight different models, with four relating to computer viruses and malware:

- Viruses are generically "bad," and users could become passively infected if they visited a "bad" place on the internet or ran across popup ads;
- Viruses are actually poorly-written or "buggy" software, but with worse system impacts, and the user had to download or open something deliberately to unleash the virus upon the system;
- Viruses are designed to intentionally cause computer system mischief, and can also be "contracted" by visiting "bad" places on the internet; and
- Viruses are designed to collect information, steal information, and support criminal activities.

and four others relating to hackers and digital break-ins:

- Hackers are digital graffiti artists, usually college-age people out to show off for friends

by defacing websites or causing problems for victims;

- Hackers are burglars who break in for criminal purposes, stealing victim information for later monetary profit;
- Another mental model is a distinct subtype of the burglar model: Hackers are criminals that target "big fish," only vandalizing people in positions of power or who control a significant amount of (personal or corporate) wealth; and
- Hackers are subcontractors, of the same age group as the digital graffiti artists, but who target large organizations or individuals—a hybrid of the graffiti artist and big fish models.

Wash uses this taxonomy of folk models in computer security to examine how users respond to different pieces of computer security advice, arriving at several findings (see Section 5.1.1 and Table 5.1). For example, Wash found that users with the third and fourth "virus" models generally used anti-virus products, whereas users with the first and second models (which cited user-initiated actions as the vector for infection, rather than vectors stemming from beyond their locus of control) did not follow anti-virus recommendations (p. 9). Second, Wash found that all users regardless of virus model tended to avoid suspicious email attachments, believing that emails were significant vectors for infection (p. 9). Third, users with the "big fish" and "contractor" models of hacking "uniformly believed that they were not the targets of hackers and were therefore safe" (p. 10). This perceived immunity meant that these users followed nearly none of the proposed advice.

Wash's study was re-run in Germany by Kauer, Günther, Storck, and Volkamer (2013) following the original study's protocol as closely as possible. Results indicated support for all eight of Wash's folk models, also categorized broadly into folk models of virus or malware infection and computer hacking. Interestingly, Kauer, et al. attribute this categorization to the focus of the interview subject: if the core of the subject's folk model was concerned with the functionality of the software or method of attack, that subject's folk model fell into one of the virus descriptors. Alternatively, if the core of the subject's model was concerned with the perpetrator rather than the method, that model fell into the hacker descriptor (p. 103). This can be compared to Wash's results, which indicated that subjects were more ambiguous with such distinctions and about which subjects admitted to not thinking about prior to the interviews

(Wash, 2010, p. 4).

In addition, Kauer, et al. also added three folk models to the corpus of results, one to the "virus" and two to the "hacker" folk model categories. First was the model of viruses as government software, which are created by federal officials and implanted on home computers either by state-employed hackers or police officials; in fact, some subjects referred to these viruses as "Bundestrojaner" or "Staatstrojaner," which roughly translate to "State Trojan horses" (p. 105). The other two are hacker folk models of similar nature. The first was a model of hackers as employees of governments or secret services, which strongly correlated with the federal virus model and typified those who were more interested or familiar with computer security and politics. These hackers were seen as either cyber-defenders used to prevent attacks against German internet infrastructure, or were themselves attacking either others outside the country or spying on the citizenry (p. 107). The other model, that of hackers driven by opportunistic or idealistic purposes, characterized hacker collectives such as Anonymous, LulzSec, and the German Chaos Computer Club (p. 108). These were seen by subjects as hackers that wanted to expose corporate vulnerabilities or to point out "deplorable circumstances in politics." Kauer, et al. attributed the addition of these folk models to the previous years' popular news about computer security and hacking events, including Wikileaks, shifting Facebook privacy laws, and news about German federal spyware (p. 109). This work was conducted before the internationally-popular Edward Snowden leaks of 2013.

2.4.3 Additional Investigations

Others have used interviews and other knowledge elicitation techniques in an attempt to abstract user mental models surrounding various computer security practices. For example, Dourish, Grinter, De La Flor, and Joseph (2004) used interviews to study user security approaches to ubiquitous computing (more commonly referred to as IoT or "Internet of Things" devices), finding that people usually classified security threats as either marketers, stalkers, spammers, or hackers. Weirich and Sasse (2001) used interviews to study password hygiene in large organizations so as to hone risk communication strategies, finding that users classified hackers

(individuals that passwords were meant to keep out of systems) as troublemaking kids, online criminals, vandals, vengeful individuals with particular vendettas, or a loosely-defined collection of "others" that included terrorists and spies (p. 139). Kang et al. (2015) use interviews to investigate mental models of internet security and functionality, but interestingly found that general awareness of privacy threats was a better predictor of protective actions, rather than the sophistication of the subject's mental model of the physical internet (p. 47).

Some researchers have also focused on computer security mental models with respect to interfaces and specific digital applications. For example, Raja, Hawkey, Hsu, Wang, and Beznosov (2011) designed and tested a personal firewall interface that leveraged Camp and Asgharpour et al. (2007)'s suggestions for risk communications emphasizing physical security metaphors with novice users. Human-subjects experimentation and semi-structured interviews suggested that users preferred vocabulary and warning language associated with the physical security metaphor over language that came packaged with a commercial firewall product sold by Comodo (p. 4). Users were reported to find the physical metaphor warnings more intuitive, easier to understand, faster to understand, and that they were better able to direct the attention of subjects (p. 5).[10]

Heckle, Lutters, and Gurzick (2008) studied the implementation and use of Novell's SecureLogin, a single sign-on (SSO) web portal service in a hospital. Through a fifteen-month ethnographic field study, it was discovered that user mental models of how SSO worked did not correspond to how SSO actually worked. In reality, SSO works by using one "master password" to authenticate a user across any number of secondary logins or applications. Each secondary application still maintains its own password, which are typically distinct from one another and the SSO master password. When a user is asked to sign in to an application, the SSO program authenticates the user, retrieves her password for the selected secondary application, and automatically enters that password to log the user in. However, users of the SSO service thought that it homogenized their login credentials across every secondary application, which meant that every secondary application would have the same login credentials; this was of

[10] Another possible explanation for these results is that Comodo's warning language was written in an unusually poor manner.

course problematic as users tried to bypass SSO but yet use their SSO logins for the secondary applications (p. 4).

The mismatch between system and mental models resulted from several problems. First, the training and enrollment process for hospital employees repeatedly emphasized "**one** username and **one** password across all applications" (emphasis theirs). Network administrators also reinforced this (incorrect) notion during staff training meetings and explainer sessions prior to the rollout. Second, the enrollment process required that users, once they had logged into the Novell system, select the secondary applications that they used in the hospital and enroll them, one by one, into the Novell SSO service. Once they selected an application, users would then provide their secondary application usernames and passwords, which the Novell client would store so that it could properly interface with these secondary applications if the SSO system was used. However, during the enrollment process the Novell logo was maintained on the same screen that users were asked to enter their secondary application login credentials, inducing mode confusion: was the user in Novell mode or secondary application mode? As a result, many users entered their incorrect credentials and were not able to login because the Novell SSO system did not check to ensure that the secondary application credentials actually worked. This was further exacerbated by most users' assumptions that all of their login credentials had already been homogenized across all secondary applications, because that was what the Novell system "was supposed to do" (pp. 5-6). This resulted in several extra months of work and three different user rollout periods in a very busy public hospital.

2.4.4 Limitations of these Works

Results from a number of studies cited above assume perspectives from the human factors mental models literature. First, several papers assume that there are differences between novices and experts, with some results bearing this out. Consider the work by Asgharpour et al. (2007) suggesting that expert users were more likely to gravitate towards warfare folk models of cybersecurity risks, whereas novice users were more likely to move towards physical security folk models. Raja et al. (2011) found empirical evidence suggesting that Ashgharpour, et al.'s

approach may have been correct. Kauer et al. (2013) also found something similar, noting that users more likely to possess the state-sponsored hacker folk models were more familiar with computer security and politics; this increased familiarity could be read as a higher level of expertise among those users.

Others assume the perspective from Norman's mental models can be ill-defined and integrate pieces of one another. For example, Wash found that the subcontractor folk model may be a hybrid of the graffiti artist and big fish models. Without the perspective of Norman and others, one could argue that Wash's investigation simply did not have enough power to provide a definitive answer as to the separation between these folk models.

Some assumptions could be easily seen as weaknesses. One serious general concern is that researchers in this area have a fluid, poorly-defined grasp on the term "mental model." This can lead to significant slack in the interpretation of results, the rigor with which we can discuss them, and their implication for future work or usefulness for more systematic exploration (for example, with formal methods). Consider Camp's work in establishing her five mental models of computer security risks. Again, she establishes physical security, public health, criminal activity, warfare, and economic or market models as five mental models that describe user perspectives on computer security failures.

Though she uses the term "mental model" to describe these constructs, it is in no way clear from her work that these constructs are anything more than, for example, a type of heuristic or metaphor that subjects use to think about security risks and make decisions about their actions. Her qualitative descriptions provide no way to assess whether subjects demonstrate the capability of running these mental models, or even of making predictions about ramifications by strict inference (Norman, 1983, p. 13). These models also seem to be light on any description of exactly *what* they model. Camp identifies "computer security failures" as the target, but this is ambiguous. From the standpoint of human factors engineering, mental models should discuss the mental representation of how a subject interacts with something in the world. It is therefore extremely difficult to comment on these models' observability, or on the coupling between the user's mental model and the conceptual model, because it is not clear whether these have been described in Camp's work.

However, there are some examples within the literature that approach mental models from positions more closely aligned with human factors engineering. For instance, Wash's four folk models of hacking perform better than folk models offered with no way to explore or verify their real-world ramifications, because subjects are asked to apply their models to respond to security criticisms and identify security advice they would or would not take; by extension, Kauer et al. (2013) deserve recognition here as well. Heckle et al. (2008) also seem to approach the problem in a more rigorous fashion, using a detailed ethnographic study to locate and articulate mismatches between user mental models and the aspects of the Novell SecureLogin automation with which hospital staff interacted. These findings could then be used to improve user training and interface design to more closely align these two models, target improvements our work will also discuss in Chapter 6.

2.5 Exploring Human-Systems Interaction with Formal Methods

2.5.1 Formal Methods

Formal methods are a collection of well-defined mathematical languages for modeling, specifying, and verifying system models and their properties (Wing, 1990). These system models, often of computer hardware or software programs, are formal descriptions that leverage well-supported theoretical formalisms (including finite state automata, directed graphs, and μ-calculus, and other strategies) to represent a system's structural properties (Bolton, Bass, & Siminiceanu, 2013). Specifications rigorously describe desirable system properties, and the verification process explores the system model to mathematically prove whether the model satisfies a specification. While a variety of options are available for formally verifying large, complex systems, theorem provers and model checkers offer powerful tools to accomplish verification tasks (Bolton, 2010).

2.5.1.1 Theorem Proving

Theorem proving is a deductive process by which computers can reason about and prove potentially complex mathematical theorems using first-order logical operations (Bibel, 2007). This is a straightforward process whereby a set of first-order logical axioms and rules can be used to construct theorems regarding correctness claims about the system, and these theorems can be subsequently verified (Bolton, 2010). Interactive theorem provers have been built to facilitate this process, though concerns over complexity prohibit analysts from completely automating theorem proving, particularly where higher-order logics are concerned (Harrison, 2013). Though perhaps esoteric in origin, these techniques have yielded valuable insight into issues in human-systems interaction, for example discovering potential post-completion errors for users interacting with a chip-and-PIN machine (Curzon, Rukšėnas, & Blandford, 2007) and exploring how users of a text editor modify their mental models of both the device and how to accomplish an editing task after failing to execute that task (Moher & Dirda, 1995). A wider corpus of work, however, has been conducted with model checking.

2.5.1.2 Model Checking

Model checking is a highly-automated approach to formal verification (Basuki, Cerone, Griesmayer, & Schlatte, 2009). A system model is formalized as a set of variables and descriptions of transitions between states. A temporal logic, such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL), is typically used to specify desirable system behavior by using the formal model's variables to construct propositions in a mathematically precise, unambiguous manner (see Table 2.2 for commonly used modal operators) (Clarke, Grumberg, & Peled, 1999). Verification then exhaustively explores the entire statespace of the model to prove whether or not the specifications hold (Baier & Katoen, 2008, pp. 11–14). Violations produce a counterexample, an execution trace capturing the model state in which the violation occurred, as well as a list of incremental model states leading to the violation. From this trace, it is possible to look back through the steps that led to the violation, diagnose the problem, and gain insight about why the counterexample was produced and how to potentially solve the problem if desired.

Table 2.2: Frequently Used Temporal Logic Operators

Operator Type	Name	Usage	Interpretation
Path Quantifier	**A**ll	**A** ψ	Starting from the current state, all future paths satisfy ψ.
	Exists	**E** ψ	Starting from the current state, there is at least one path that satisfies ψ.
Temporal Operator	**F**uture	**F** ψ	ψ is eventually true in some future state of a given path.
	Global	**G** ψ	ψ will always be true in a given path.
	Until	ϕ **U** ψ	ϕ will be true until ψ is true for a given path.
	Ne**X**t	**X** ψ	ψ is true in the next state of a given path.

Note. Here, a *path* is a valid temporally-ordered sequence of states for a given model. ϕ and ψ are two propositions about either a state or path in the model that can evaluate to either `true` or `false`. Adapted from Bolton (2010), p. 6.

While model checking techniques have been applied in a wide variety of use cases, complexity and scalability concerns have generally restricted these use cases to simple systems and models (Kneuper, 1997, p. 11). However, technological improvements within symbolic model checking (Rushby, 2007), such as abstraction (Chan, Anderson, Beame, & Notkin, 1998) and partial-order reduction (Flanagan & Godefroid, 2005), have helped researchers apply the rigor of formal methods to larger, more complex systems.

2.5.2 Formal Methods and Mode Confusion

Work within the confluence of formal methods, human-automation interaction (or HAI), and mental models has focused substantially on *mode confusion*. Here, an action that may be appropriate in one context is performed in an inappropriate context because the user is unclear (knowingly or unknowingly) about the current state of the system (Wickens et al., 2015, p. 314). These mismatches between user mental model state and the state of the world provide a ripe area of exploration for formal methods and model checking (Bredereke & Lankenau, 2002). More specifically, these systems can be captured in formal, stateful representations, paving the way for the rigorous exploration of system interactions.

Here, the work of Asaf Degani and others on the use of these techniques to examine mode confusion in the use of an automatic flight control system, or auto-FCS, is illuminating (Degani & Heymann, 2002). This analysis used state transition diagram representations of both the

FCS (p. 37) and the user model (p. 39), constrained to fragments specifically relating to the transitions between vertical flight modes that are associated with altitude capture during takeoff (p. 36). Importantly, the authors note that certain mode changes are actually the secondary results of primary changes in flight characteristic parameters.

2.5.2.1 An Illustrative Example of Mode Confusion

To control aircraft, pilots may at any time manually specify numerical altitudes to which the autopilot will fly the aircraft, such as a specific cruising altitude. In order for the FCS to arrive at the correct cruising altitude (say, 30,000 feet: "Flight Level 30,000" or "FL300" in the appropriate vernacular), the FCS will begin capture maneuvers at a specific altitude and enter "altitude capture" mode at an altitude lower than the desired cruising altitude, such as FL270. This will reduce the rate of ascent when it hits that capture altitude so that the aircraft climbs comfortably into and then holds the cruising altitude without overshooting it. When cruising altitude is successfully attained, the aircraft FCS exits "altitude capture" mode and enters "altitude hold" mode.

Suppose that an aircraft begins capture mode at FL270 and is progressing through FL290 while on its way up to FL300. If a pilot (because these values can be overridden at any time) revises the altitude hold target down to FL280, the FCS will automatically adjust the flight dynamics of the plane, remain in the current "altitude capture"mode, and bear out the remaining system actions until it drops to the new target and enters "altitude hold" mode. However, if the pilot manually overrides the hold altitude and sets it to FL250 feet, which is lower than the originally-specified FL270 "begin capture" target, the FCS will refuse the order and instead automatically transition to a different mode called "vertical speed unconstrained" in which complete control over the airplane's altitude is returned to the pilot and is given no oversight by the automation. In this case, the FCS mode automatically changes in response to a parameter change, rather than an FCS interface change specified by the pilot herself (Degani & Heymann, 2002, pp. 36-39). If left uncorrected, a pilot could be at risk for accidentally overshooting its originally-set, required holding altitude at FL300.

Because these details about the FCS of a popular aircraft could be gleaned from pilot training manuals, Degani and Heymann leveraged work described in (Degani, Shafto, & Kirlik, 1999) to explore the interaction between model state transition diagrams of the FCS and the pilot's mental model of the FCS operation with respect to this rare altitude capture scenario. Results identified a mismatch between the system model and the user's mental model of the system, finding that the example scenario above could result in a pilot assuming that the FCS would yet remain in altitude capture mode, potentially missing the automatic transition to the unconstrained mode due to an unassuming parameter change (p. 40). The authors confirmed this result by discovering an identical incident report published in NASA's Aviation Safety Reporting System (ASRS).

2.5.2.2 Degani's *error* and *blocking* states

Degani and Heymann (2002) also describes a generic architecture for discovering *error* and *blocking* states through a comparative exploration of user mental model and system state transition diagrams.

Consider a car driver and her mental model of the car's transmission system (Figure 2.2). The user can shift the vehicle through these gears by pushing *up* or *down* on the gear shift, similar the semi-automatic or clutchless manual transmissions used in modern cars where gears can be shifted by pushing up or down on a shifter stick rather than navigating through the gear pattern.

Here, the user's mental model on the left is comprised of three states: Low, Medium, and High gears, with push actions used to transition between each state. The transmission's model is similar and will shift gears based on the inputs provided by the user to L1, M1, or H1, correspondent with the user's mental model of the Low, Medium, and High gears. However, contrary to the user's mental model, the first gear is not actually just a singular Low gear. There are actually two low gears, L1 and L2. L1 is consistent with normal driving conditions, but if the user applies sufficient gas without shifting up, the transmission will enter L2 to provide a power boost. From here, a downshift or reduction in applied gas input returns the transmission

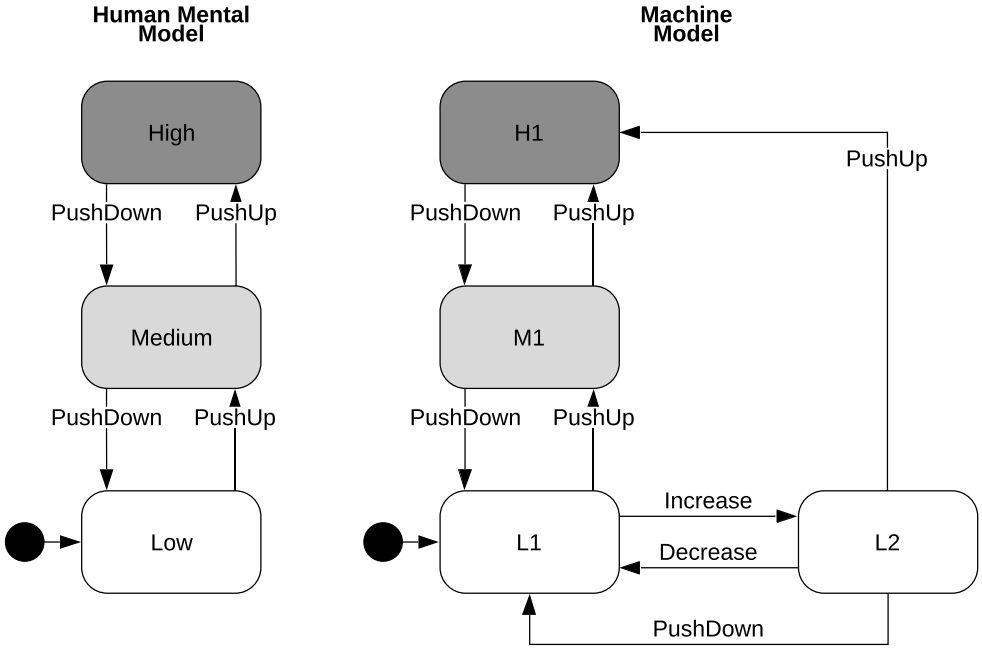

Figure 2.2: Example models of a vehicle transmission system and the user's mental model of the vehicle transmission system.

to L1, but an upshift input transitions the vehicle to H1 and skips M1.

Within Degani's architecture, an error state can result where there is a mismatch between the user's mental model of the system state and the system's actual state (p. 33). Figure 2.3 represents the composite of both state transition diagrams, representing the user's states with Low, Medium, and High, and the car's states with L1, L2, M1, and H1. State mismatches are highlighted in red.

Here, a user who thinks she is in Low gear and upshifts to enter the Medium gear will put the car into M1 so long as the car actually started in L1. If, however, the user was accelerating and was unaware the car was in L2, then upshifts, she will put the car into H1 rather than M1. This state mismatch is represented in the "H1, Medium" state. If the user downshifts in an attempt to put the car back to where she thinks it was (Low), she will actually put the car into M1; conversely, if the user upshifts in an attempt to shift into High gear, her action will

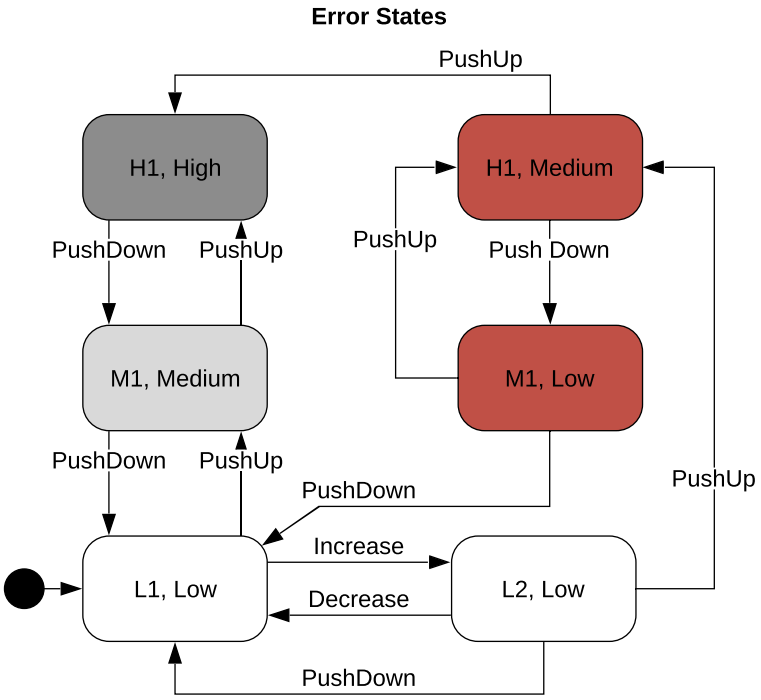

Figure 2.3: A composite of the user and machine models, where a mismatch between models generates an error state, in red.

not change the transmission's gear at all. Both of these conditions could be dangerous for the user (perhaps if trying to pass a car on a two-lane highway) or do potential damage to the car (perhaps by lugging the engine). Mismatches between user mental model and system states are called *error* states.

Alternatively, a *blocking* state occurs when a user mental model prevents her from interacting correctly with the system, blocking her from taking the appropriate action (p. 33). This can be seen in Figure 2.4.

Within the same composite model, the L2 gear and the transition action to get there (*Increase*, which signifies the application of more gas through the gas pedal) are both unknown to the user. Furthermore, the user may be unaware that she could either stop accelerating (*Decrease*) or downshift to return to the lowest gear. As a result the user is unaware of possible system transitions and, if the user encounters an unwanted state, could be prevented from successfully recovering or knowing how to recover back to a safe state. These conditions could

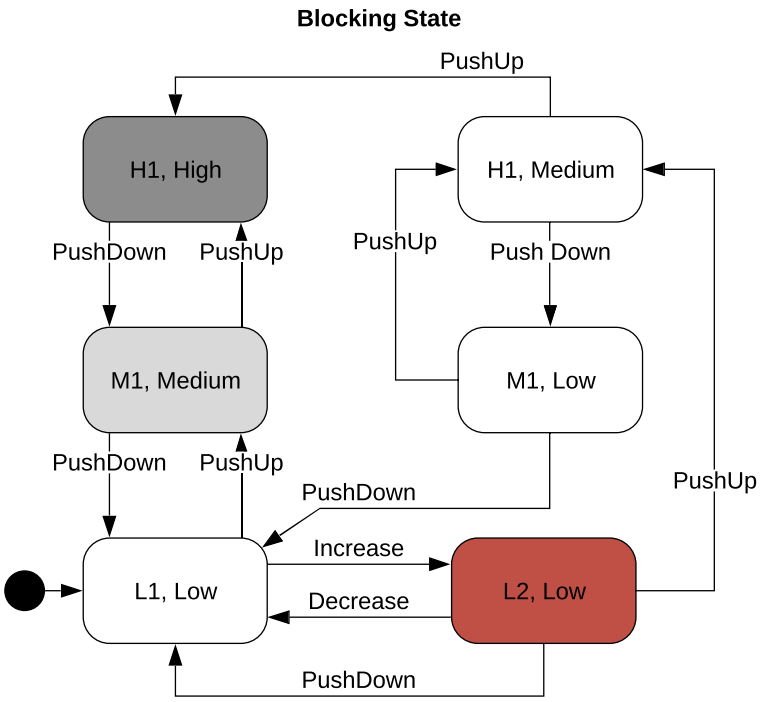

Figure 2.4: A composite of the user and machine models, where a mismatch between models generates an error state, in red.

also be dangerous, for example if she slows down but continues to downshift the transmission even though it is already in L1.

Error and blocking states are significant because they offer interesting and slightly more nuanced approaches to user and system model mismatches. Rather than simply noting whether or not mismatches exist, it is useful to determine with more precision where they exist and what the mismatch signifies, as their location within the user's task structure and the nature of the mismatch could suggest potential strategies for model remediation or further exploration. Detected blocking states may suggest targets for interface improvement, in that transitions unaccounted for in the user's mental model may be insufficiently observable to the user (Norman, 2013, pp. 197-198). These problems may also be alleviated through the improvement of information displays, for example by correcting erroneous information or displaying missing data. Alternatively, error states could suggest areas to which improvements in training regimens or user manuals are necessary, so that users can ensure agreement between their mental model of

system states and the operational model of the system itself (Degani & Heymann, 2002, p. 42).

Others have pursued similar research objectives. For example, John Rushby has explored autopilot altitude capture procedures in MD-88 aircraft systems (Rushby, 2002; Rushby, Crow, & Palmer, 1999) and used automated state exploration tools (a specific class of model checking tools discussed in Bredereke & Lankenau, 2002) to examine the results of Degani and Heymann (2002) and Rushby (2001a). Rushby's results are noteworthy because of his use of automated tools to aid in the model checking process (compare this with Degani and Heymann, above, which utilizes a manual construction and comparison of state transition diagrams). One of these tools is a model checker called Murϕ,[11] and Rushby routinely provides actual samples of code from his models, complete specifications, and counterexample output within his papers (Rushby, 2002).

Another problem that has been considered is the exploration of complex systems in ways that attempt to integrate, or at least account for, continuous system behavior. Known as hybrid systems techniques, work in this area has generally focused on aircraft dynamics and control systems, where discrete quantities may not fully capture the depth of the system in question. For example, Meeko Oishi and others have focused extensively on the interaction between mental models and models of hybrid systems. Oishi, Hwang, and Tomlin (2003) presents work on ways to rigorously describe, analyze, and make recommendations for interface design such that continuous quantity parameters or flight characteristics can be discreetly represented and effectively integrate with discrete models (like the user mental model). Oishi, Mitchell, Bayen, and Tomlin (2008) provides additional information on the process and design recommendations, applying these recommendations to case studies involving a car driving through an intersection with a yellow light and an airplane's autolanding function.

While the above results explore ways to look for positive evidence of mismatches between user and system models, others have considered using formal methods to find ways of *avoiding* mode confusion. One of these problem areas is the generation of interfaces that do not exhibit properties associated with mode confusion, an interesting problem that emphasizes exploration of the interface rather than on just the deleterious interactions between mental

[11] Pronounced MUR-fee, which is humorous.

models and system interfaces as they exist. For example, Combéfis worked to establish and check for what he and others term *full-control properties*, a type of conformance relation which attempt to capture whether or not "the operator's mental model carries enough information to enable full control of the system" (Combéfis, Giannakopoulou, & Pecheur, 2014, p. 16).

This means two things. First, the user must know the possible commands that can be executed, and these possible commands must match between the user and system models. Second, the set of possible observations of which the user is aware must match the observations possible according to affordances within the system model (p. 16). Full-control requires that these properties always hold across the entire interaction space between user and system models. In this particular paper, the authors were able to explore the interaction between pilot and system models and detect a potential mode confusion scenario within the automatic speed control features within a Boeing-777's FMS (p. 19).

This work also discusses *how* the user mental models are generated. User mental models can be abstracted from training manuals, subject interviews, categorization tasks, card-sorting tasks, cognitive walkthroughs, think-aloud protocols, or other knowledge elicitation exercises. They are then generally left as-is, perhaps with the expectation that the mental models deduced actually correspond to the mental models of subjects. However, Combéfis and colleagues have developed techniques to progressively abstract and refine these results down to what he and others term the *minimal safe mental model* (Combéfis et al., 2014; Combéfis, Giannakopoulou, & Pecheur, 2016; Pynadath & Marsella, 2007). These are the simplest mental model of a target system that can be constructed and yet still correctly explain or enable user interaction with the system, where the goal is to design succinct interfaces and shorter user interaction sequences that reduce the actual size of the interface, the number of indicators that must be designed and integrated, and the perceptual and cognitive burden on the user (Heymann & Degani, 2007, p. 312).

Combéfis uses two different algorithmic approaches: bisimulation and machine learning. Bisimulation is the exploration of two different transition systems, wherein the states of each are evaluated to detect potential congruence between them (Combéfis & Pecheur, 2009). These similarities can determine whether states relate closely enough such that one of the

systems can be simplified. This process is run iteratively until the simplest possible model is derived that still exhibits bisimilar equivalence; that is, the simplified transition system has no loss in fidelity or capability (Baier & Katoen, 2008, Section 7.1). The second method uses is called *L**, a type of machine learning that iteratively builds mental model candidates. These candidates are then explored until one is discovered that still maintains the full-control conformance relation (Combéfis, Giannakopoulou, Pecheur, & Feary, 2011). These methods are powerful, resulting in interesting perspectives on the *a posteriori* modification of mental models and potential reduction of model statespaces. These are important developments for the verification of human-automation interactive systems because they addresses some limitations of exploring mental models with these techniques.

2.6 Conclusions

This literature review has explored the intersection of mental models, formal methods, and cybersecurity. We began with an introduction to mental models and their evolution in the literature, surveying diverse perspectives on their development and implications of the term "mental model" when unpacked. We then moved to a discussion of selected current problems in cybersecurity, briefly addressing the seriousness of this issue for a hyper-connected 21st century and the role of mental models and folk models in modern cybersecurity challenges.

Our review then turned to theoretical perspectives from human factors literature seen in formal methods investigations. Here, we saw interplay between certain aspects of Norman (instability, incompleteness, parsimoniousness) and assumptions in general from the formal methods papers, noting that meaningful results from work in formal methods depends on mental models being stable, accurately-abstracted mental models that strongly characterize how users think about their interaction with the systems under scrutiny. The perspectives from human factors (that users may have several models or different abstractions that they employ depending on task; that they are fluid; that they leverage physical object metaphors, or functional abstractions, or something in between) may prevent work from being taken as *prima facie* valid without careful evaluation and thinking.

Based on our review, we argue that the development of robust formal methods techniques to explore human-automation interaction within a cybersecurity domain could offer powerful tools for discovering the hard-to-anticipate failure modes that threat actors leverage through social engineering and other attack strategies in which humans are the linchpin. This approach offers to bridge the work between human factors and formal methods practitioners, providing a unique perspective on cybersecurity issues that is well-supported by existing work in both fields. These novel contributions could also help stabilize work in folk modeling techniques, bringing runnability to folk models and aligning them more closely with the strengths of traditional mental models as defined in the human factors literature.

3 | Research Questions and Objectives

If I have seen further it is by standing on the shoulders of Giants.

Sir Isaac Newton
in a letter to Robert Hooke, 1675

3.1 Research Questions

We have identified two primary research questions (RQs) based on the gaps identified in the literature review from Chapter 2.

3.1.1 RQ1: Understanding the Impact of Mental Models on Cybersecurity

Modern cybersecurity threat actors are rapidly moving towards the use of humans as critical components in the progression of attacks. This poses a significant challenge for existing defensive techniques because the cybersecurity community currently lacks any well-developed capability to detect user-enabled exploits. Thus, being able to anticipate dangerous and potentially unforeseen user-enabled vulnerabilities before attackers can exploit them offers a promising strategy for improving network defenses. However, little work has been done to rigorously develop this type of cyberdefense tradecraft. Our work therefore intends to answer the following research question:

RQ1: How can an understanding of computer user mental models and their treatment of cyber threats help network defenders become more resilient to cyberattacks that exploit human users? Does this understanding shed light on the anticipation of dangerous and potentially unforeseen vulnerabilities? What more can be learned about the nature of these exploits and vulnerabilities by rigorously characterizing the influence that user mental models of security threats have on their actions within a computer system?

3.1.2 RQ2: Developing a Method to Rigorously Characterize and Explore User Vulnerabilities

While research from the formal methods community has addressed a variety of issues in human-automation interaction, comparatively little has investigated interactions between user mental models and computer security tools. Therefore, our work intends to answer the following research question:

RQ2: How can formal methods techniques be extended to give analysts the ability to discover and describe security vulnerabilities that arise from the interaction between user mental models of computer security tools and the tools themselves? Are these techniques extensible to a variety of scenarios? How can they help us discover real-world security vulnerabilities and demonstrably add value to modern digital defense strategies?

3.2 Research Objectives

To address these research questions, we are proposing four research objectives (ROs) that will serve as tangible milestones to direct our work. The RQ with which each RO corresponds is highlighted below.

3.2.1 RO1: Establish

> *RO1: Establish a rigorous framework for exploring user mental models of computer security vulnerabilities with formal methods techniques.*

This objective will leverage data-backed, validated, and published work in formal mental models to inform the construction of runnable formal models capable of exploring human-automation interaction in a cybersecurity context. In addition to the models, this framework will also incorporate a number of generic specification property types that, when formally checked, can characterize and detect potential security vulnerabilities and other undesirable conditions.

Correspondent Research Questions: RQ1 & RQ2

3.2.2 RO2: Demonstrate

> *RO2: Demonstrate that these vulnerabilities correspond to real-world conditions that attackers could potentially exploit in modern computer systems.*

This objective will exercise the framework by investigating user security configuration strategies within a popular cloud data storage solution and characterizing them through our perspective on Degani's *error* and *blocking* states. Leveraging the analytic constructs established in RO1, we will use model checking to discover how user actions taken in a variety of scenarios could lead to potential vulnerabilities and lead to data theft or loss. We will also investigate whether users could recover from potentially dangerous situations.

Correspondent Research Questions: RQ1 & RQ2

3.2.3 RO3: Extend

> *RO3: Extend our framework to incorporate folk models of cybersecurity risks.*

This objective will extend our framework to incorporate folk modeling techniques that explore user perspectives on cybersecurity risks. This objective will synergistically integrate work

from a variety of fields, leading to more rigorous characterizations of these vulnerabilities and introduce strategies to make folk models runnable. Our investigation will also result in the formulation of additional security specifications critical to discovering rare and potentially unforeseen vulnerabilities.

Correspondent Research Questions: RQ1 & RQ2

3.2.4 RO4: Validate

> *Analyze data collected from multiple sources to validate our modeling approach by discovering potential configuration errors anticipated with our method.*

This objective will offer empirical evidence to suggest that the results of our approach correspond to real-world vulnerabilities in modern digital systems. Two scenarios will be used. First, we will investigate the vulnerabilities discovered in RO2 and demonstrate that our findings are correspond with real-world vulnerabilities by reproducing these discovered scenarios within the target cloud data storage system. Second, we will use a large, published corpus of user responses to simulated and actual phishing emails to show that our method's anticipated user vulnerability profiles correspond with real-world phishing response data.

Correspondent Research Questions: RQ1 & RQ2

4 | Formulating a Method to Discover Unanticipated Cybersecurity Vulnerabilities

> So the rule of military operations is not to count on opponents not coming, but to rely on having ways of dealing with them;
>
> Not to count on opponents not attacking, but to rely on having what cannot be attacked.
>
> *Sun Tzu*
> *The Art of War*

The literature review in Chapter 2 discussed the concerns of researchers and practitioners over accelerating growth in cyberattacks that depend on actions taken by human users for successful attack execution (Collett, 2014). These actions are important because they can help attackers bypass the traditional network defense tools and countermeasures that otherwise make these attacks risky, difficult, and expensive to deliver and utilize (Verizon Enterprise, 2016). Furthermore, these actions need not be deliberate to result in the same vulnerabilities that attackers can exploit. One example is the configuration of software security settings, policies, or protocols, where misconfigurations that may be hard to detect or anticipate could result in weaknesses that open the door to further malicious activities. These erroneous configurations are also generally "legal," meaning that these settings are allowable and that integrity of the

software or its code was not directly compromised by viruses, malware, or targeted technical attacks. Because it may not be immediately clear for users or decision makers how to distinguish benign from potentially dangerous software misconfiguration errors, attacks leveraging technically-permissible security policy misconfigurations can therefore be difficult to anticipate or detect (Department of Homeland Security, 2009).

One way forward may be to formally explore the mental models of users making software configuration decisions. As discussed in Chapter 2, even crude or generic mental models allow users to run systems forward or backward in their minds, predicting future behavior given current actions or explaining current conditions given previous actions or system states (Norman, 1983). While mental models are likely essential to successful decision-making within a cybersecurity context, problems can arise where mismatches occur between the user's mental model of a system and the system itself (Degani & Heymann, 2002; Rushby, 2001a). These mismatches can also be very difficult to anticipate or detect by the users of the system *and* the system designers (Bolton et al., 2013). Here, the exhaustive statespace search and robust state representational capabilities of formal methods and model checking can shed light on these rare, unanticipated mismatches between user mental models and system states. Formal methods could therefore offer a powerful strategy for detecting these mismatches and the dangerous cybersecurity vulnerabilities that result.

However, there are several outstanding problems regarding the use of formal mental models to address cybersecurity issues. First, little work has been done in the human factors community to investigate mental models within a cybersecurity domain. While there are examples from the literature on this topic, a significant amount is from researchers in computer science, electrical engineering, information technology, and related domains (Furman, Theofanos, Choong, & Stanton, 2012; Moher & Dirda, 1995; Prettyman, Furman, Theofanos, & Stanton, 2015). This does not negate the value of their contributions. However, the literature reviewed in Chapter 2 indicates that terms and operational definitions can be quickly muddied through their use by practitioners in other domains with less rigorous definitions.

Second, it is difficult to construct external representations of mental models because they need be neither correspondent to the world nor internally consistent (Dutton & Starbuck,

1971; Norman, 2013). From the literature review we see that protocols exist for eliciting user mental models, but again little rigorous work has been done on mental model elicitation with respect to modern cybersecurity threats. This poses a challenge for using mental models constructed *de novo* to improve userspace cybersecurity because computer security threat mitigations—which require precise implementation specifications—may find little agreement with these neither correspondent nor consistent findings.

We propose a framework to address these problems and develop new strategies for discovering cybersecurity vulnerabilities introduced through unanticipated or hard-to-detect software configuration errors. This work adapts the formal mental model analysis approach of Degani, et al. for use in a cybersecurity domain, allowing security practitioners to explore vulnerabilities from the perspective of users at risk of committing these errors (see Section 2.5.2). Our method extends Degani's architecture by translating the notions of *error* and *blocking* states into concepts grounded within this domain. This offers to significantly expand the theoretical body of work to which Degani's architecture is relevant and offers a foundation for other researchers exploring these and related concepts.

We have also extended existing work to formulate a generic mental model of the target system's behavior using training materials and documentation (Bredereke, 2003; Bredereke & Lankenau, 2005; Degani & Heymann, 2002; Degani et al., 1999; Rushby, 2001b). This also provides a rigorous exploratory framework for researchers interested in using formal methods concepts to investigate current issues in computer security (Voas & Schaffer, 2016). To this end, we have created several patterns of specifications that specifically relate to cybersecurity concerns. Because they are inspired by Degani's *error* and *blocking* states, they form a strong and direct link between the theoretical underpinnings of our method and the real-world computer security threats that this work ultimately aims to address.

After generically formulating our method, we will demonstrate its capabilities by modeling and performing a use-case analysis of Amazon Web Services' (AWS) Simple Storage Service (S3), a popular cloud data storage solution owned by Amazon.com. Growth in AWS' usage and revenue rates have outpaced competitors for several years, positioning it as the leading provider of personal- and enterprise-level cloud computing resources (Asay, 2018; Novet,

2018). However, the configuration of certain resources used by customers of this service can be difficult to correctly configure, in part because several options allow for partially-overlapping configuration states and the future security implications of these states can be hard to determine. Errors in the configuration of these resources can lead to serious data loss, including leaks of sensitive personal information, corporate data theft, and financial loss. Moreover, because these services are marketed to users who may have relatively little experience or understanding about how to properly configure them, this has become a major area of interest that attackers have begun to explore in earnest. We argue that our method is well-suited for extension to this domain area, and our results yield immediate, widespread benefits for both the research communities studying these problems and real-world cybersecurity efforts.

We will conclude the chapter by exploring our formal analysis results and their significance for system security and integrity, then demonstrate that these failures are reproducible within AWS S3. A discussion of our results and their implications will follow.

4.1 Overview of the Problem Space

Degani's error and blocking states signify potential ways in which unanticipated human-automation interaction can result in undesirable behavior or system properties, and this is due to a mismatch between the human's mental model of a system's state and the actual state of the system (see Degani and Heymann (2002) and Section 2.5.2). These mismatches can lead to users unwittingly driving a system into an unacceptable or illegal mode of operation, as in the case of an error state, or block the user from taking her desired course of action altogether.

Within a cybersecurity context, error and blocking states could result in unsafe computer system behavior or prevent the user from accomplishing certain tasks. For example, a user that encounters an error state may change system settings, open seemingly innocuous emails, or otherwise take certain actions that she thinks are safe but are actually quite dangerous. In each case, the user was under the genuine impression that their actions were acceptable, or "legal," because her mental model of the system's security functionality gave no indication that

such an action might drive the system into an illegal or unacceptable state. These scenarios can result in credential theft; digital networks compromised by malware, remote access tools, or ransomware; data theft; and more.

Blocking states, where this mismatch prevents a user from taking an appropriate action, are also potentially dangerous for computer security. This could manifest in attempts to recover a system from an undesirable state, where the user thinks an action to bring the system back to a safe state is not allowed by the system—even though it is. Here, if a user changes a security setting or clicks on a suspicious file, she thinks she is unable to recover from her action. This risks causing additional damage to the system, its network, or the data it contains.

By creating a formal representation of mental modeling concepts and Degani's architecture, our method offers a way to rigorously describe and detect these unanticipated error and blocking states within a cybersecurity context. Furthermore, by utilizing counterexample traces that are automatically generated during model checking, analysts can trace the origin and evolution of these error states within a given scenario. We describe our method below and discuss its use in characterizing potential cybersecurity vulnerabilities.

4.2 The Method: A Generic Formulation

In order to integrate the work of Degani and techniques in formal mental modeling from the formal methods community, established work in the modeling of computer security issues, and work on mental models from the human factors community, we have created a general approach (see Figure 4.1).

In this method, a human analyst is able to examine generic information about cybersecurity within various external resources to construct a formal system model that captures a target cybersecurity application around a specific architecture. The analyst also uses the same information with generic specification properties to assert checkable properties about formally modeled concepts. By model checking these properties against the formal model, this process allows analysts to identify if, and if so how, discrepancies between machine and user mental models can cause cybersecurity problems. Below we describe how method inputs are formu-

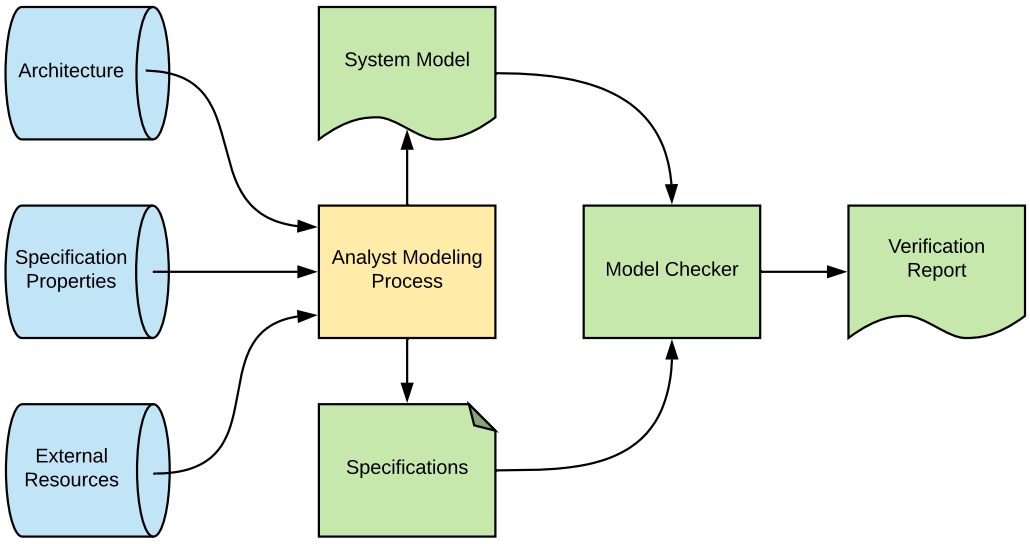

Figure 4.1: The generic formulation of our modeling framework.

lated into the system model and specification properties respectively, as well as how the model checking process is performed.

4.2.1 Architecture and System Model

The novel formal modeling architecture we developed, which extends the basic formal mental modeling concepts introduced by Degani and Heymann (2002), is shown in Figure 4.2.

This architecture represents a system as four synchronously composed modules: `user`, `system`, `action generator`, and `comparator`. The `user` module represents the user's mental model about how the system works. The `system` module captures the actual behavior of the system. Both are nondeterministic state transition systems and can transition between states based on input events generated by the `action generator` module, such as a user creating, modifying, or deleting system resources. In Degani's mental modeling work, mental models and system (machine) model states were each mapped to abstract concepts (such a legal or illegal states; or "high", "medium", or "low" states) that were important to the application being analyzed. Because we are explicitly concerned with the cybersecurity domain, we map user and system model states onto concepts important to cybersecurity. Specifically, the `user` model's states map onto whether the user thinks the system is *safe* or *vulnerable*. The

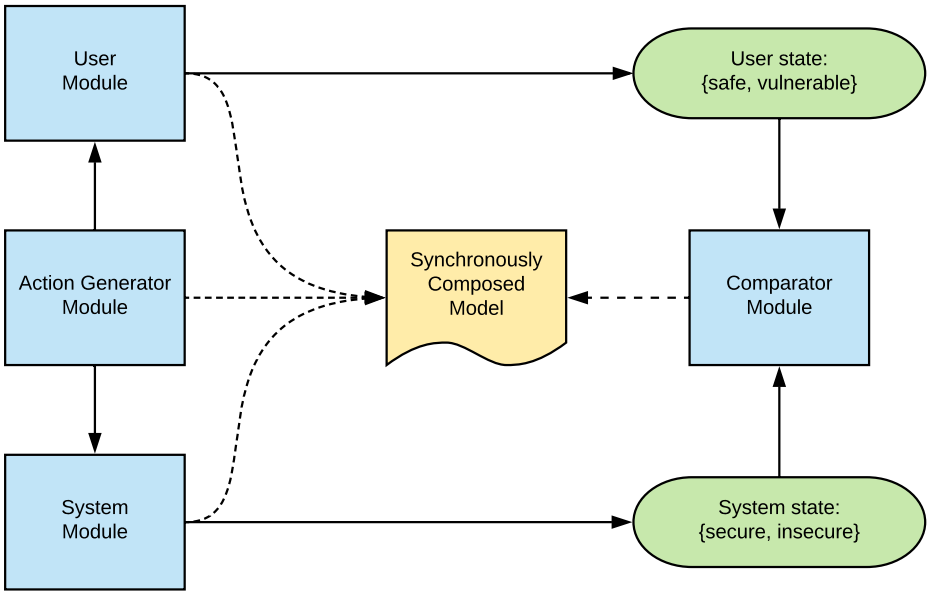

Figure 4.2: The architecture of the formal system model.

`system` model's state mappings indicate if the actual system is *secure* or *insecure*. The results of these mappings are taken as inputs to the `comparator` module, which computes whether mismatches exist between the `user` and `system` modules in the current state: if the user thinks the system is safe when it is insecure or vulnerable when it is secure.

The contents of the `user` and `system` modules must be derived from domain-specific external resources (*viz.* Amazon Web Services, 2018g). Here, our architecture allows a generic user mental model instantiation to function as an open parameter in the `user` module, where user actions and system-level parameter values can result in mental model state changes and how that state will map to our notions of "safe" or "vulnerable." Therefore, when constructing the `user` module, the analyst will use the information contained in domain-specific external resources to determine how these transitions and mappings should occur.

4.2.2 Specification Property Patterns

Our method identifies several novel specification property patterns that can be used with model checking to discover mismatches between the `user` and `system` modules. All of these extend the error state and blocking state concepts introduced by Degani and Heymann (2002).

Because we map `user` and `system` module states onto cybersecurity-relevant concepts (safe, vulnerable, secure, and insecure), we have two distinct error states that each mean something different for cybersecurity.

The first property asserts that we never want an error state where the user thinks the system is vulnerable when it is secure. We call this condition a "false vulnerability" mismatch, because the user has incorrectly assessed that a vulnerability is present. This can be asserted as follows:

$$\text{NoFalseVulnerabilityMismatch} \models \mathbf{G}\ \neg\ (user = vulnerable \wedge software = secure). \tag{4.1}$$

This asserts that globally (**G**) it should never ($\neg$) be true that, in the same state, the user thinks the system is vulnerable and ($\wedge$) the software is secure. A false security mismatch is problematic for two reasons. First, it can indicate a situation where a user may avoid otherwise secure actions because she thinks that those actions will make the system vulnerable. Second, if the user thinks the system is vulnerable when it is not, he or she may avoid sensitive behaviors when they are actually safe.

The other error state that can occur happens when the user thinks the system is secure when it is vulnerable. We call this condition a "false security" mismatch, because the user is operating under a false sense of security—a vulnerability exists where the user thinks there is none.

$$\text{NoFalseSecurityMismatch} \models \mathbf{G}\ \neg\ (user = safe \wedge software = insecure). \tag{4.2}$$

This generically asserts that globally it should never be true that, in the same state, the user is safe and the software is insecure. A false security mismatch is potentially dangerous for two reasons. First, the ability to reach such an error state can imply that the user can do something that compromises computer system security without her knowing it. Second, it indicates a situation where the user does not know that the system is insecure and thus she may not properly protect sensitive resources.

Degani and Heymann (2002) also identified blocking states as unwanted discrepancies

between user models and system models. These are conditions where users do not know that an action or event will change a system's state. Within a cybersecurity context, a blocking state is only important if an action or event would allow an insecure system to become secure without the user's awareness of the system's change or the possibility of such a change. We can formulate this as:

$$\text{NoRecoveryBlocking} \models \mathbf{G}\, \neg \left(\begin{array}{c} (user = vulnerable \wedge software = insecure) \\ \text{AND } \mathbf{X}\ (user = vulnerable \wedge software = secure) \end{array} \right). \tag{4.3}$$

This asserts that globally it should never be true that the current state of the user is vulnerable and the software is insecure and, in the next state (**X**), the user will still be vulnerable but the software will be secure.

4.2.3 Model Checking

Once the analyst's modeling and specification process is complete, the model checking environment uses the formal system model to generate a stateful logical representation of the model in computer memory. It then uses the specifications written by the analyst to explore the model for the properties they describe. If the specification holds, then those properties are not present in the model and a proof is returned. If those properties are discovered, a counterexample is returned detailing the nature of the property violation and an execution trace showing the steps that led to the violation's discovery. More details about this process have been outlined in Section 2.5.1.

4.3 Case Study: Amazon Web Services and Security Configuration Errors

4.3.1 Motivation for Exploring AWS S3 Configuration Errors With the Method

Amazon Web Services, or AWS, is a large collection of cloud computing resources offered by Amazon.com. Accounts can be created for free by anyone and provide access to a suite of online tools provided by Amazon. Some products are tailored for small individual users, while others are geared towards extremely large organizations: major AWS customers include Netflix, Hulu, GE Oil and Gas, Kellogg's, Airbnb, NASA, Siemens, a portion of the NASDAQ, and the credit scoring organization FICO, among many others (Amazon Web Services, 2018b). These services charge only for the computing or storage used and can be deployed either in conjunction with one another or as needed in an *à la carte* fashion. Similar to other cloud computing and storage providers, such as Microsoft Azure or Google Cloud Platform, AWS offers clients an efficient platform to handle complex computational tasks without paying for system idle time, infrastructure development, maintenance, or other costs associated with managing these services.

Using our method to investigate these types of inadvertent misconfigurations could yield a number of significant findings. First, because Amazon's services are widely used by both small and very large organizations, the breadth of their reach has created a large attack surface for hackers seeking opportunities to steal data and make money. This can put the sensitive information of consumers, businesses, and governments in the hands of threat actors willing to sell this information to the highest bidder.

Recent news stories and technical breach analyses, sometimes called "post-mortems," have laid bare the seriousness of recent AWS S3 security misconfigurations. Newman (2017) describes how 198 million US citizen voting records, possibly dating back ten years, were left publicly exposed by an analytics company. Cameron (2017) investigates how Booz Allen Hamilton, one of the United States' largest defense contractors, left 60,000 sensitive Department of Defense files and government passwords to sensitive systems in unprotected S3 resources. In

O'Sullivan (2017), the Pentagon left 100 gigabytes of data on an unsecured S3 server in 2013. Leyden (2017) describes how thousands of job applications classified at the secret level were left in public S3 resources. Muncaster (2017) discusses a small Verizon database with internal company information that was discovered as publicly accessible, and Whittaker (2017) writes of a much larger Verizon database with 14 million customer records that was left unprotected on a misconfigured S3 server. In Ashok (2017), the National Credit Federation left 100 gigabytes of customer social security numbers, back account numbers, and credit reports exposed in a misconfigured S3 bucket.

While these are but a sampling of the most recent news stories on the matter, in each case AWS S3 resources were simply misconfigured and left publicly discoverable by white-hat cybersecurity researchers who could guess where and how to look. No "hacking" was needed to violate the integrity of code running on Amazon's servers. Furthermore, while some stories describe misconfigurations committed by smaller organizations, others were erroneously configured by employees at large defense contractors and the United States government: organizations that likely employ in-house security teams and professionals to write these configuration protocols. This demonstrates that S3 misconfiguration problems are widespread, span an alarming array of organizations, and can leak potentially serious information. The reach of AWS and the opportunity to discover unanticipated cybersecurity vulnerabilities due to these configuration errors could therefore have an enormous, immediate impact on a global technical services provider and its clients.

4.3.2 Overview of Amazon Web Services

AWS offers 131 distinct tools and services that are organized into 19 different categories (Amazon Web Services, 2018a).[1] Among others, these service categories include:

- **Cloud computing:** Users can use virtual computers or servers that are managed and secured within Amazon's infrastructure;
- **Database services:** Users can create and manipulate databases of information that are written in different management systems and are capable of handling petabytes of data;

[1] These figures were current as of April 2018.

- **Networking:** Users can manage global web content delivery and routing; and
- **Cloud storage:** Users can store large amounts of data (on the order of exabytes) in various cloud storage repositories.

Amazon has been able to retain a significant portion of the cloud computing market share because it offers services that are suitable for a variety of use cases. For example, home users can back up important computer files to AWS Glacier, a cloud storage service meant for months- or years-long data storage requirements. University labs without access to computing sites can rent time on highly efficient computing platforms using Amazon Elastic Cloud Compute (AWS EC2). Large organizations can operate complete business productivity suites in AWS infrastructure using AWS WorkMail, WorkDocs, and Chime.

Amazon services can also be linked together to accomplish more complex tasks or handle extreme edge case computing requirements. For example, small businesses wishing to build their own website can do so cheaply by hosting static HTML pages on AWS S3, routing their website to Amazon's DNS provider through AWS Route53, and delivering its content globally with AWS CloudFront. Alternatively, large businesses attempting to modernize and migrate extensive amounts of data to the cloud can use AWS Snowball and Snowmobile to do so. AWS Snowballs are large self-contained data storage devices (about the size of a full-size desktop tower) that can be shipped to a customer. There, users can download up to 80 terabytes of data per snowball, then ship it back to Amazon where it can be uploaded to the customer's S3, Glacier, or other AWS data storage solution. Snowmobiles accomplish the same task, but can store up to 100 petabytes of data and instead come housed in a ruggedized 45'-long shipping container towed to the customer site via tractor trailer.[2]

The breadth and capability of Amazon's cloud computing and data storage services, coupled with their market share in this expanding industry, mean that any potential security vulnerabilities could have significant and far-reaching ramifications. The complexity inherent in some of these services, as well as their ease of access to potentially untrained users, also means that problems may be difficult to predict or detect before they manifest within these systems. We

[2]A first-hand account of an AWS Snowmobile use case was written by DigitalGlobe, a satellite imagery company. It can be found here: http://blog.digitalglobe.com/industry/digitalglobe-moves-to-the-cloud-with-aws-snowmobile/.

therefore feel that AWS services are an important and worthwhile subject to explore with our method.

4.3.3 Storing Data in the Cloud with AWS Simple Storage Service

We will use our method to explore one of Amazon's cloud storage services for potential cybersecurity vulnerabilities that could arise out of user configuration errors. AWS Simple Storage Service, or S3, is a cloud data storage solution offered by Amazon. Users with an AWS account can log in and create one or more data storage containers called "buckets." Within each bucket the user can store an unlimited amount of information of any file type, including photos, videos, text documents, system files, or any other items (called "objects") so desired. Once created or uploaded, buckets and objects can be downloaded, changed, or deleted at will from either a web browser or an authorized computer's command line interface.

For example, suppose that Alice wants to back up some of her files to the cloud. She creates an account with Amazon AWS and decides to store these files within S3. Alice has two types of files she wants to back up: her travel photos from the Grand Canyon and Italy, and her tax returns. She therefore makes two buckets, one for photos and another for tax returns, and from her computer she uploads her pictures to the photo bucket. They are now safely stored in the cloud.

Alice next moves to back up her tax returns, but realizes that these are of course much more sensitive and must be kept maximally secure. AWS S3 provides Alice with different ways to manage the permissions on her objects and buckets, and these permissions allow her to control who is and is not allowed to access them with granularity. By confirming that the permissions on the tax returns are configured to be extremely restrictive, Alice can be sure that only she is allowed access to those documents.

4.3.3.1 Methods for Control: Policies and Access Control Lists

Alice has different methods at her disposal for controlling access privileges to her tax returns, and each offers a slightly different way for doing so. The relationship between the system

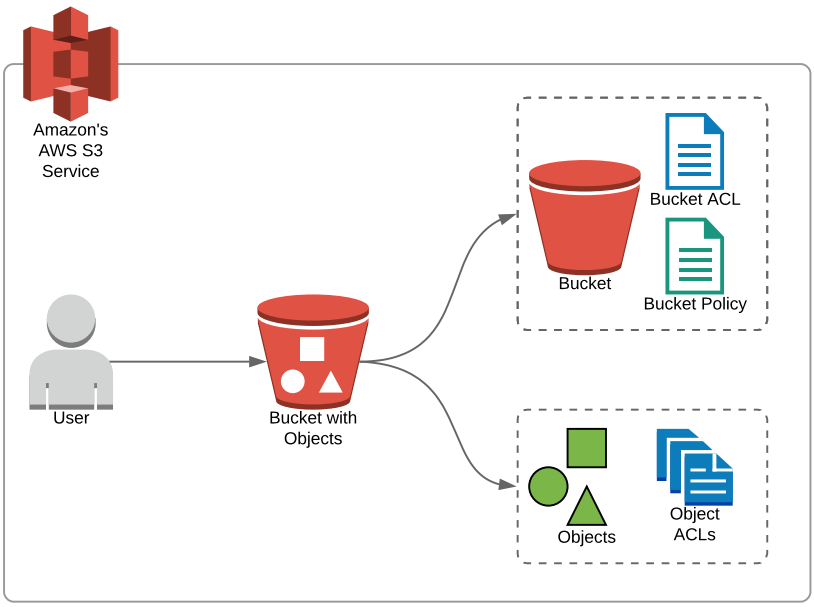

Figure 4.3: A graphical description of the relationship between Amazon AWS S3 buckets, objects, and their respective control documents.

components managing these privileges are shown in Figure 4.3.

Upon creation by a user, every bucket can be given a bucket policy, a bucket access control list (ACL), or both. These "control documents" are small snippets of code that dictate who can read, write, or even see that the bucket exists. By default, a newly-created bucket is assigned only an ACL; no policy document is created. A bucket policy is a document that expansively defines the types of actions that authorized users are allowed to perform with the bucket and all of the objects contained therein. The users authorized in this manner can vary, from just Alice herself, to other users Alice might know, to any public user on the internet who knows where to look.

Through bucket policies, authorized users are allowed to read or write objects to the bucket, change bucket or object permissions, and change how access attempts are logged, among other actions. These can also be made as simple or complex as desired, and by changing the bucket policy Alice can change which users are allowed to perform different actions. For example, a bucket policy can allow access to only certain files within a bucket, restricting access to all other files, or allow only users from certain IP addresses to access documents or make other

changes. In total, there are 68 distinct policy statements that users could add to a bucket policy, and each of them can be configured in several different ways (Amazon Web Services, 2018c). An example generated bucket policy is shown in Figure 4.4.

```
{
  "Id": "Policy1523476503635",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1523476501084",
      "Action": [
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:PutBucketPolicy",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::alice-taxreturns/*",
      "Principal": "*"
    }
  ]
}
```

Figure 4.4: An example bucket policy allowing public read and write for the bucket's policy and its objects.

This power is offered in part because bucket policies are written in JSON and are highly extensible, meaning that users can add as many policy statements she desires to finely control the access privileges of many users. However, AWS offers no graphical user interface for writing these policies within the bucket configuration page. They can either be typed by hand or generated using Amazon's external policy generation wizard, but this can be difficult to do and even the wizard requires some format-specific typing. Furthermore, while Amazon's bucket policy configuration page will provide an alert indicating if the current policy would make the bucket or its contents public, some of the 68 policy statements follow hard-to-distinguish naming conventions (for example, `ListBucketMultipartUploads` and `ListMultipartUploadParts`) and there are no checks for other problematic access configuration errors. Because these can be just as problematic as public access but much more difficult to spot, particularly for novices inexperienced with JSON, misconfigured bucket policies can be dangerous.

An ACL is the second way to control the permissions of individual users over the bucket and all of its contents, but they offer a much less granular approach to managing access. ACLs allow authorized users to read objects from or write objects to the bucket, but they also allow users to read the permissions ("Who can read, write, or change the bucket's structure?") or

modify the list of authorized users ("Let me add someone to the authorized user list."). These authorized users can be either those known by the user (requiring an AWS user ID or email address) or the general public. No other privileges are offered through ACLs, and these can be configured through the use of a simple graphical user interface shown in Figure 4.5.

The latter two ACL options allowing public users to write to the object or write to the object's ACL can be dangerous, offering attackers a way to write themselves into the access policies, modify files at will, or even stage a hostile takeover by writing the bucket owner out of the ACL and barring them access to their own resources. Because the ACL interface allows people to easily specify these permissions for the public, this means that anyone with the full bucket name could take advantage of its misconfiguration and write to those parameters. Practically speaking, if Alice configures her bucket ACL in a certain way, anyone who knows where to look could potentially access all tax return objects inside her tax return bucket with impunity.

The third method for controlling user access to bucket contents is to set ACLs for specific objects within a bucket, restricting the control of other users on a per-object basis. Here, it's possible to select the users who can read the object, as well as view and edit the permissions that users have over the objects. This would allow Alice to put both her pictures and her tax documents in the same bucket, but select different permissions for each document so that pictures could be publicly viewable while the tax returns could remain private. While this is also troubling, for example if Alice accidentally sets one of the tax documents to be publicly viewable, these permissions do not allow object-level write in the same way bucket ACLs allow bucket-level write. The configuration interface for object ACLs is highly similar to the bucket ACL configuration interface shown in Figure 4.5. Additional discussion regarding the ways that these permissions structures interact to determine who can read and modify bucket contents is provided in Section 4.3.3.2.

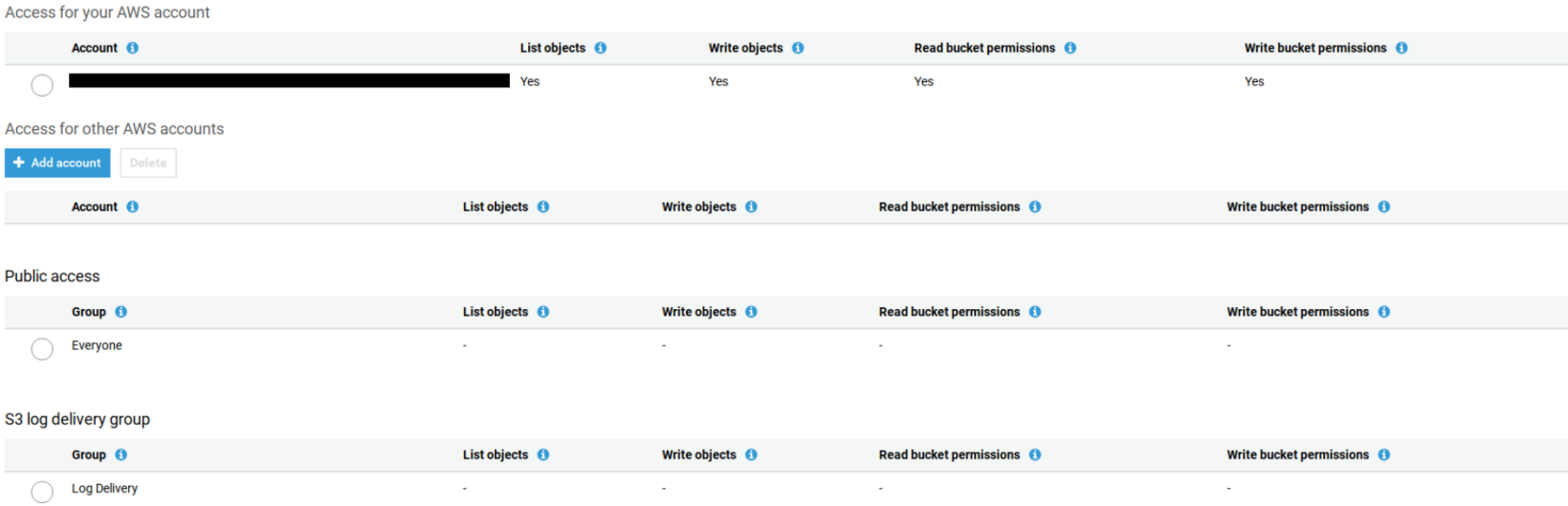

Figure 4.5: The bucket ACL configuration page. This example ACL allows only the owner read and write permissions over the bucket's contents.

4.3.3.2 Integrating Policies and ACLs to Determine Access Privileges

The control documents used to determine access privileges to buckets and objects offer complementary but somewhat overlapping access rights to users who would like to read or write to data stored in S3. It is therefore possible to write contradictory control documents, wherein one could write both a bucket policy allowing public users to list bucket contents and a bucket ACL disallowing all public access. To rectify these conditions, AWS automatically converts all of the relevant permissions into a set of policies, then evaluates that policy set in a series of steps that are outlined in Figure 4.6. S3 automatically executes this process whenever a user makes request to read, write, or perform any other operation on bucket or object (Amazon Web Services, 2018d, 2018e).

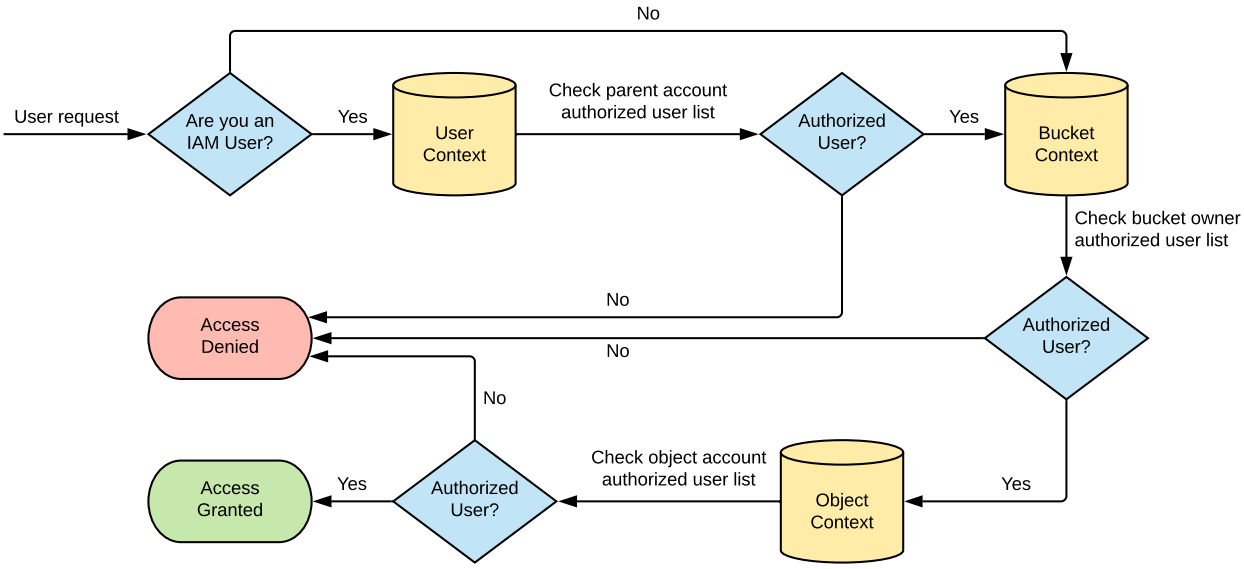

Figure 4.6: The policy evaluation method by which AWS S3 handles object access requests. Adapted from Amazon Web Services, 2018e.

Within the evaluation protocol, S3 checks the user request against the various permissions within three different contexts. First, if the user performing the operation is recognized by the Identity and Access Management (IAM) system, then S3 will check to be sure that the parent AWS account has authorized that IAM user's access to the parent account's resource. If not authorized, S3 will then evaluate the user request within the context of the bucket permissions

set by the bucket's owner. If not authorized, S3 will then evaluate the user request within the context of the object permissions set by the object's owner. If not authorized, the user is then authorized to perform her requested action on the object. Furthermore, note that if the user is only requesting access to a bucket, this process will grant access immediately after the bucket context check is passed.

This strategy allows AWS to build a whitelist of the users who have permission to access or modify S3 resources based upon the control documents written by resource owners. Whitelisting, whereby all users are denied access except for those specifically authorized, is considered a more robust cybersecurity practice than blacklisting, whereby all users are allowed access except for those specifically prohibited (Schneier, 2011). However, users that create a sufficiently expansive whitelist are in danger of inadvertently authorizing a malicious actor. While the strategy with which S3 collapses and explores the resource permissions structure is effective, users inadvertently casting an overly expansive whitelist through incorrect or erroneous policy and ACL configurations can compromise its integrity.

4.3.3.3 An Example of Responsible Access Privilege Deployment

Though these privileges are very dangerous if used improperly, when assigned with care they offer powerful methods for restricting access but maintaining a convenient file structure.[3] Suppose that Alice and Bob are co-workers at the GEICO Buffalo branch, and GEICO uses S3 buckets to store worker data. Alice is a member of the accounting department, while Bob is on the human resources (HR) team. GEICO has created a separate bucket for the each branch, but does not want all employees within the entire company to have access to all documents used by everyone. The corporate IT department could set each bucket's policy to allow access only from specific branch employees. This would allow Buffalo employees to read the documents stored inside the Buffalo bucket, but would prevent Chuck at the Nashua branch (or any others not authorized to access the Buffalo branch's resources) from seeing those documents.

[3]This is why Amazon offers such an expansive array of options: they want to provide as many tools and the most flexibility possible for power users. This helps users address the corner cases and varied deployments that gives AWS its competitive edge. However, this is a double-edged sword and can lead to the catastrophic erroneous configuration states mentioned in this chapter.

In a more advanced use case, suppose that the corporate IT team allotted each branch its own bucket, but wanted all documents to be stored within that bucket (rather than within any further subfolders labeled by department) for easier auditing. Because this makes the file storage hierarchy flat, the Buffalo branch IT team could, in the interest of security, more finely restrict the documents that specific employees can see. For example, Alice in accounting should perhaps not have access to Bob's HR personnel files, and both should be restricted from seeing the IT team's stored username and password files for all of the Buffalo branch employees; however, all of these documents will be stored together within the single bucket owned by the Buffalo branch. By using only bucket-level policies and ACLs to control access to files, that means that all employees with access to the Buffalo bucket could see all documents stored therein. Object-level ACLs, on the other hand, could be used to restrict user document access to their own departments and restrict access to department-specific files, but still allow the entire branch's documents to exist inside one bucket. This compartmentalizes employee access while also making it harder for hackers to move effortlessly through the GEICO documents looking for sensitive files. When used responsibly, the proper S3 control documents can therefore offer a secure computing environment and help users or administrators assert greater control over their data.

4.3.3.4 Implications of Access Configuration Policies for the AWS Security Landscape

While large organizations may have well-funded security teams hunting for security misconfigurations (though it seems to make little difference), the popularity and affordability of AWS products means that smaller companies and individual users make up a significant portion of the AWS userbase. These smaller users are perhaps less likely to possess the resources or knowledge to adequately protect their AWS resources from inadvertent access configuration errors. This represents an enormous gap between need and capability, and our method's findings could have an outsized benefit for these smaller AWS customers.

These problems likely arise because AWS S3 configuration can be difficult and confusing. Bucket policies and ACLs offer similar functionalities that can interact and behave in unintu-

itive ways, and with the addition of object-level ACLs it can become frustrating for users to pick out the details of these configuration strategies that offer appropriate security but do not prevent resources from behaving as expected. Misunderstanding how these settings can impact bucket and object security and how attackers might take advantage of misconfigured S3 buckets can lead to data theft, data loss, and other grave consequences.

Our method is therefore highly suitable for exploring AWS S3 for configuration errors. The opacity of these settings and their effects offers a target-rich environment in which Degani's *error* and *blocking* states could likely play a role *vis-à-vis* the interactions between policies and ACLs. Furthermore, AWS provides a variety of ways to configure these settings, with some offering point-and-click GUI interfaces and others requiring users to modify the JSON policy code manually. We anticipate that the methods by which these settings are configured and how they interact potentially make it difficult for users to determine the effects of configuration decisions and anticipate their future consequences on cyberattack resilience. If something undesirable does occur, we also anticipate that these problems may make it hard for users to determine how their S3 file storage system could be brought back to a safe state. Because our method has the capability to address these concerns, our strategy for investigation is uniquely situated to comment on problems in this domain.

4.3.4 Formal Modeling

Using the architecture outlined in Figure 4.2, we implemented a formal system model describing the domain around our mental modeling concepts. This was done using the input language of the Symbolic Analysis Laboratory (SAL) (de Moura, Owre, & Shankar, 2003). SAL was appropriate for our work because it is an extremely expressive formal language, is supported by a suite of different model checking tools, and has been used in a number of formal human factors analyses (Bolton et al., 2013; Oliveira et al., 2017).

In constructing the formal model, we considered Degani and Heymann's *error* and *blocking* states, as well as external resources that describe AWS S3 resource configuration protocols. Below we describe how each of these were integrated into our model by having the

`user` and `system` modules respond to actions dispatched through the `action generator` module. In this particular application, our analysis of system capabilities and potential file storage use cases resulted in the inclusion of several user actions that could result in changes in state. All actions represented specific operations on S3 resources described in AWS documentation: bucket-specific actions include `create_Bucket`, `change_BucketPermissions`, `change_BucketEncryption`, and `delete_Bucket`, and object-specific actions include `create_Object`, `change_ObjectPermissions`, `change_ObjectEncryption`, and `delete_Object`.

4.3.4.1 The Action Generator Module

The actions generated by the `action generator` module drive events across the system model. These events modify specific objects or buckets ("resources") and correspond with the eight bucket- or object-specific actions identified in Section 4.3.4. In addition to creating or deleting resources, read or write privileges can be specified for each resource through the assignment of one or more permissions: `read`, where the contents of the resource can be viewed; `readPermissions`, where the access control policy for the resource can be viewed; `write`, where the contents of the resource can be modified; and `writePermissions`, where the access control policy for the resource can be modified. Furthermore, permissions can be assigned to one of three distinct user groups: the `user`, or resource owner; an `authenticated` user, or one to which owner has deliberately granted access to a resource; or `public`, or any user on the internet. Finally, encryption can be applied to or removed from both bucket and object resources.

Each action is performed on system resources by executing logical statements contained within the module. All possible user actions occur concurrently with a set of action permissions (`actionPerm`) that reflect the permissions that could be assigned by a given action. Permissions respect the target system's default procedures and only correspond with allowable, real-world AWS S3 settings. These defaults are encoded in the `action generator` module, a portion of which is captured in Figure 4.7.

```
actionPerm IN IF action = change_ObjectPermissions OR action =
    ↪ create_Bucket OR action = change_BucketPermissions
     THEN {x : permissionSetting | NOT (x.public AND isPrivate?(x))}
     ELSE {x : permissionSetting | (NOT x.public) AND isPrivate?(x)}
    ↪ ENDIF;
```

Figure 4.7: An example of AWS S3 default settings captured within the Action Generator module.

In this `actionPerm` definition, a conditional (`IF .. THEN .. ELSE .. ENDIF;`) statement captures the logic describing how permissions settings should be assigned. The `IN` operator means that the `actionPerm` variable is assigned a value that satisfies the logic expressed in the conditional statement. Set notation is used to describe how permissions are assigned based on the action being considered. This takes the form `Identifier : Type | Expression`, where `Identifier` is a notional identifying variable, ":" can be read as "of," `Type` corresponds to a type described within the model, "|" can be read as "where," and `Expression` describes how the type is to be assigned. In the method, we have established the `permissionSetting` type as a record type, where each permissions value (`public, read, write, readPermissions, writePermissions`) is captured as a `BOOLEAN` basic type.

For example, buckets created within Amazon's S3 system can be assigned either public or private permissions. The `actionPerm` definition assigns these conditions and prevents contradictory conditions from occurring by requiring that, if the user creates a new bucket (`IF ... action = create_Bucket`), the new bucket is assigned a permissions setting that is not of both the public and all private types (`THEN ... NOT (x.public AND isPrivate?(x))`). Objects, however, are private by default at the time of creation. This is captured by the method on the `ELSE` condition (`ELSE ... (NOT x.public) AND isPrivate?(x)`).

The configuration and ramifications of these "public" and "private" permissions is handled elsewhere in the model through a series of functions:

```
isPublic?(perm : permissionSetting) : BOOLEAN = perm.public AND (perm.
    ↪ read OR perm.write OR perm.readPermissions OR perm.
    ↪ writePermissions);
```

Figure 4.8: A function identifying the "public" permissions set.

In Figure 4.8, a resource is "public" if the `perm.public BOOLEAN` flag and one or more flags

for `perm.read`, `perm.write`, `perm.readPermissions`, or `perm.writePermissions` are set to `TRUE`. This prohibits conditions where other resources, whose behavior is not specifically captured in the logic of the `action generator` module, could be set as "public" but have no other public read or write privileges. Private permissions settings are handled in a similar way by the `isPrivate?` function:

```
isPrivate?(perm : permissionSetting) : BOOLEAN = NOT perm.read AND NOT
    ↪ perm.write AND NOT perm.readPermissions AND NOT perm.
    ↪ writePermissions;
```

Figure 4.9: A function identifying the "private" permissions set.

In Figure 4.9, a resource is "private" if none of the public read or write permissions are set to `TRUE`. Additional functions provide similar configurations for other permissions types. See Appendix A for further details.

After computing all potential changes to system resources, the `action generator` module pushes each action and, if appropriate, its associated permission and encryption settings to both the `system` and `user` modules.

4.3.4.2 The System Module

The `system` module takes an action as input and computes all updates to existing system resources, if they exist, and initializes new resources if those created by the incoming action do not exist. These computations are handled by a series of guarded transition statements that ensure accurate system behavior is executed, with the guards prohibiting the execution of a system action unless all preconditions have been met. Within the `system` module, guards are generally of the form $(action = userAction \wedge condition[1 \,..\, n])$, where a specific action and one or more conditions must be met to permit execution of the transition.

For example, S3 does not allow an object to exist without a bucket to contain it, so preconditions on the `create_Object` transition statement shown in Figure 4.10 require that a bucket exist and that it currently contains no object.

This describes a situation where, if the `create_Object` action is called, a bucket exists, and

```
[] action = create_Object AND bucketExists AND NOT objectExists -->
     objectExists'       = TRUE;
     objectEncryption'   = IF actionEnc = none_Option THEN
   ↪ bucketEncryption ELSE actionEnc ENDIF;
     objectACL'          = actionPerm;
```

Figure 4.10: An example guarded transition statement.

it contains no object, then the guarded transition is executed. The system model updates its internal state to reflect that an object exists (`objectExists' = TRUE;`) and encryption and permissions values that have been assigned. In the former, if an object is created but no encryption protocol was manually assigned (`actionEnc = none_Option`), then the object will inherit a matching encryption protocol from its bucket (`THEN bucketEncryption`), else it will default to the protocol assigned by the `action generator` module at object creation (`ELSE actionEnc ENDIF;`). In the latter, the object's permissions will take the value assigned by the `action generator` module at object creation (`objectACL' = actionPerm;`).

After computing all potential changes to system resources, the `system` module will update both of its system security variables— `softwareRead` and `softwareWrite` —to either *secure* or *insecure*. These are important cybersecurity characteristics within this application because they signify whether the AWS S3 system is safe from or vulnerable to potential public data-read or data-write conditions resultant from the system actions taken, respectively. We compute system resource read safety thus:

```
softwareRead = IF objectExists AND objectACL.public AND (objectACL.read
   ↪  OR objectACL.readPermissions OR objectACL.writePermissions) THEN
   ↪  insecure ELSE secure ENDIF;
```

Figure 4.11: A definition to compute whether the system is safe from dangerous public read conditions.

Here, the system is vulnerable to having its data publicly read if an object is public and it has permissions allowing public users to read the object's data, read its permissions, or write its permissions; if these conditions are not met, it is secure from being publicly read. Public write permissions, while they disallow explicit public read permissions, can allow a user to rewrite the object's control document so that other public users can read the object. We therefore consider unsecured "public write permissions" settings to still pose a public read danger.

We similarly compute system resource write safety thus:

```
softwareWrite = IF objectExists AND objectACL.public AND (objectACL.
    ↪ write OR objectACL.writePermissions) THEN insecure ELSE secure
    ↪ ENDIF;
```

Figure 4.12: A definition to compute whether the system is safe from dangerous public read conditions.

Here, the system is vulnerable to having its data publicly writeable if an object is public and allows public users to write to the object or write to its permissions control document; if these conditions are not met, the object is secure from being publicly written.

Once these states are computed, their outputs are pushed to the `comparator` module.

4.3.4.3 The User Module

The `user` module functions similarly to the `system` module, taking actions as inputs and computing updates to all new or existing system resources. However, while the `user` module uses the same state variables as the `system` module, each are prepended with `think-`. For example, where the `system` module stores `BOOLEAN` values for the `bucketExists` and `objectExists` variables, the `user` module instead identifies these as `thinkBucketExists` and `thinkObjectExists`. This strategy offers an easy method for direct comparisons between the state of the system and what the user *thinks* is the state of the system. These comparisons provide handholds for exploring Degani's *error* and *blocking* states, discussed in Section 4.4.

For example, consider the `user` module's `create_Object` transition statement example in Figure 4.13. While similar to the `system` module's `create_Object` transition statement in Figure 4.10, slight differences between them reflect the user's perspective on this action:

```
action = create_Object AND thinkBucketExists AND NOT thinkObjectExists
    ↪ -->
        thinkObjectExists'      = TRUE;
        thinkObjectEncryption'  = actionEnc;
        thinkObjectACL'         = actionPerm;
```

Figure 4.13: A second example guarded transition statement.

Here, the user thinks that the encryption setting corresponds with the setting associated with

the object's initial creation (`thinkObjectEncryption' = actionEnc;`). This reflects a subtle idiosyncrasy within Amazon's S3 interface where, if the user tells the object to be created with no encryption, the object will silently adopt the bucket's encryption setting without alerting the user before doing so. The user could therefore think they have assigned no encryption to the object, but in actuality the object has been stored with encryption. Further examples of these differences can be found in Appendix A.

After computing all potential changes to system resources, the `user` module will update both of its system security variables— `userRead` and `userWrite` —to either *safe* or *vulnerable*. In contrast to the system security variables discussed above, these signify whether the user thinks the AWS S3 system is safe from or vulnerable to potential public data-read or data-write conditions resultant from the system actions taken, respectively. We therefore perform their computation slightly differently. User object read vulnerability is therefore expressed as:

```
userRead = IF thinkObjectExists AND thinkObjectACL.public AND (
    ↪ thinkObjectACL.read OR thinkObjectACL.readPermissions) AND
    ↪ thinkObjectEncryption = none_Option THEN vulnerable ELSE safe
    ↪ ENDIF;
```

Figure 4.14: A definition to compute whether the user thinks she is safe from dangerous public read conditions.

Here, if the user thinks the object is public and that the object or its permissions are publicly readable, then the user thinks the object is vulnerable to being read by a public user; if this is not the case, then the user thinks the object is safe. There are also two additional conditions specified here that do not appear in the system's read security definition. First, the user thinks the object is vulnerable if the object is not encrypted. We feel that this is a reasonable user assumption because—since the Egyptians, and perhaps before—the purpose of encryption has been to prevent anyone but she who knows the decryption key from reading a document (Singh, 2000). A user storing their object in AWS and is presented with an option to encrypt their object is therefore likely to feel as if, because she has uploaded the object to her personal account, only her personal account is able to see the object decrypted. The user may suppose that, even if the object is listed publicly, since only her account possesses the decryption key, the public could not see the contents of the object.

However, this is not the case. AWS only encrypts the object while it is at rest within its S3 server, and AWS automatically decrypts the object for any authorized user attempting to read or write the object (Amazon Web Services, 2018g). Thus, if an object is made public, then AWS regards every public user as an authorized user with privileges over that object, and it will decrypt the object for any public user attempting to read the object. While this policy was instituted by Amazon for system architectural reasons, it does not function in a way that matches user experiences when applying data encryption. We have captured this assumption in Figure 4.14.

Second, we have omitted the `objectACL.writePermissions` condition present in Figure 4.11 from Figure 4.14 because, during object configuration, a user may not take the time (nor should they be expected) to "wargame" through conditions where an attacker's capability to write object permissions could pose a threat to the public readability of that object. Furthermore, AWS offers no alerting within the interface regarding these dangers during configuration tasks. Issuing any public permissions to an object results in the same generic notification about the danger of any public permissions application. We therefore feel it reasonable that a user experiencing alert fatigue may skim the generic warning, unaware of the secondary effect potentially resultant from their primary action, and inadvertently open their object to public read permissions.

Our method also computes the user object write vulnerability thus:

```
userWrite = IF thinkObjectExists AND thinkObjectACL.public AND (
    ↪ thinkObjectACL.write OR thinkObjectACL.writePermissions) AND
    ↪ thinkObjectEncryption = none_Option THEN vulnerable ELSE safe
    ↪ ENDIF;
```

Figure 4.15: A definition to compute whether the user thinks she is safe from dangerous public write conditions.

Here, the user thinks the object is vulnerable to public object write conditions if the object is public and is either publicly writeable or has publicly writeable permissions, and is not encrypted; if this is not the case, then the object is safe from public write conditions. Similarly to above, the user's assumptions about the nature of encryption do not reflect the method in which AWS has implemented encryption. Because the system considers a public user as an

authorized party, it grants access to the object—despite it being encrypted within its containing S3 server—and allows the public user to write to the object. The user's mental model of encryption's role within the context of this system therefore offers no protection against a public user attempting to leverage object write permissions.

Once these states are computed, their outputs are pushed to the `comparator` module.

4.3.4.4 The Comparator Module

The comparator module was implemented as described generically above. Its calculations regarding whether mismatches exist between the security variables described in `system` and `user` modules were used to assert specification properties.

4.3.5 Specification Properties

Using the conditional specification property patterns from (5.3)–(5.7), we asserted specification properties against our formal model. Because a user should be concerned with protecting system resources from both read and write access, these specifications can be expanded to check for both conditions. When applied in turn, each of the original three specifications can therefore be reformulated as:

$$\text{NoFalseVulnerabilityReadMismatch} \models \mathbf{G}\ \neg\ (userRead = vulnerable \wedge softwareRead = secure). \tag{4.4}$$

$$\text{NoFalseVulnerabilityWriteMismatch} \models \mathbf{G}\ \neg\ (userWrite = vulnerable \wedge softwareWrite = secure). \tag{4.5}$$

$$\text{NoFalseSecurityReadMismatch} \models \mathbf{G}\ \neg\ (userRead = safe \wedge softwareRead = insecure). \tag{4.6}$$

$$\text{NoFalseSecurityWriteMismatch} \models \mathbf{G}\ \neg\ (userWrite = safe \wedge softwareWrite = insecure). \quad (4.7)$$

$$\text{NoRecoveryReadBlocking} \models \mathbf{G}\ \neg \left(\begin{array}{l} (userRead = vulnerable \wedge softwareRead = insecure) \\ \text{AND } \mathbf{X}\ (userRead = vulnerable \wedge softwareRead = secure) \end{array} \right). \quad (4.8)$$

$$\text{NoRecoveryWriteBlocking} \models \mathbf{G}\ \neg \left(\begin{array}{l} (userWrite = vulnerable \wedge softwareWrite = insecure) \\ \text{AND } \mathbf{X}\ (userWrite = vulnerable \wedge softwareWrite = secure) \end{array} \right). \quad (4.9)$$

In these analyses we wanted to prove that mismatches between user and system read and write conditions, as well as recovery blocking conditions, could never occur during typical user tasks within the AWS S3 system. For the former, this results in a *condition* of the form $\mathbf{G}\ (\neg\ (mismatch))$, where *mismatch* is the type of read or write mismatch condition. For the latter, the method uses a *condition* of the form $\mathbf{G}\ (\neg\ ((mismatch) \wedge \mathbf{X}\ (mismatch))$. Here, it should never be the case that, if a mismatch is observed, the mismatch must also be observed in the next (**X**) state. Our three mismatch types across both user and system therefore yielded these six specification properties.

4.3.6 Apparatus

Each of our specification properties were verified against our formal system model using version 3.3 of SAL's symbolic model checker (SAL-SMC).[4] Analyses were conducted on a desktop computer with a 3.7 GHz quad-core Intel Xeon processor and 128 gigabytes of RAM running the Debian 9 Linux distribution.

[4] http://sal.csl.sri.com/index.shtml

4.4 Verification Results

Results from our preliminary investigations of the AWS S3 use case are summarized in Table 4.1 and discussed in more detail below. In each case, our results have been extracted from counterexample execution traces.

We have separated our mismatch results into three classifications: "false vulnerability," "false security," and "blocking." False vulnerability mismatches result when the user thinks that the system is vulnerable to potentially dangerous read or write conditions, but the system is protected from them; here, the user erroneously believes the system is vulnerable. Converse situations where the user thinks the system is safe from dangerous read or write conditions, but the system is in fact vulnerable to them, are designated "false security" mismatches because of the user is operating under a "false sense of security." Both of these correspond to Degani's notion of *error* states, whereby a mismatch between the user's mental model of system state and the actual system state can result in the user driving the system into an illegal state of some form (Degani & Heymann, 2002). "Blocking" mismatches occur when the user is unable to determine how to recover the AWS S3 system from a dangerous or undesirable system state. This corresponds with Degani's notion of *blocking* states, where the user is unaware that certain actions or events are possible within the system (Degani & Heymann, 2002).

Table 4.1: Amazon AWS S3 Use Case Verification Results

	False Vulnerability			False Safety			Blocking		
	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result
Read	932,580	0.12	✘	932,580	0.12	✘	3,229,908	0.61	✔
Write	932,580	0.12	✘	932,580	0.11	✘	3,188,724	0.36	✔

Note. A ✔ indicates a proof was returned, while a ✘ indicates a counterexample was returned.

Our results indicate that counterexamples were discovered for false vulnerability and false safety specifications with respect to both read and write conditions, suggesting that users can configure S3 resources in such a way as to result in potentially dangerous cybersecurity vul-

nerabilities. However, results also indicate that users can potentially recover from these conditions. This suggests that users may be able to mitigate these vulnerabilities if they are detected. A more detailed analysis of these conditions is provided below.

4.4.1 False Vulnerability Mismatch Results and Interpretation

4.4.1.1 Read Mismatch

Our method discovered a situation where the user thought the system was vulnerable to read conditions when it was actually secure. At step 0 the user created an encrypted bucket with public write permissions enabled, then at step 1 created a private object with no encryption. Finally, at step 2 the user then changed the bucket permissions, making the object publicly readable but revoking the bucket's public write permissions. At this point a mismatch was discovered between what the user thought the object's encryption level was and what it actually was, generating the observed counterexample.

This mismatch occurred because, if a user creates an encrypted bucket and then creates a non-encrypted object, the object silently inherits the bucket's encryption level without alerting the user. This happened in step 1. In step 2, the user makes the bucket and all of its contents publicly readable, expecting the object to be publicly readable because the user created it to be non-encrypted—a dangerous state. However, because the user was unaware that the object silently inherited the bucket's encryption level, the object could be protected from public read, resulting in the read permissions false vulnerability mismatch.

4.4.1.2 Write Mismatch

Our method discovered situation where the user thought the system was vulnerable to write conditions when it was actually secure. At step 0 the user created an encrypted bucket that only allowed authenticated users to write objects. At step 1, the user created a non-encrypted private object. At step 2, the user made the bucket publicly readable and its permissions publicly writeable. This is where the mismatch occurred.

When the user changes the bucket's permissions structure to provide public users with permissions write privileges, the user assumes that the object inherits the same type of permissions for its own policy. This would be potentially catastrophic, because an attacker could write themselves into the permissions structure and grant them unfettered access to the object. One possibility is that the attacker could grant themselves, or the public, write privileges over the object. However, this is not the case, and the object remains protected against having its permissions or contents publicly written. This results in the observed mismatch.

4.4.2 False Security Mismatch Results and Interpretation

4.4.2.1 Read Mismatch

Our method discovered a situation where the user thought the system was secure from read conditions when it was actually vulnerable to them. At step 0 the user created an encrypted public bucket with all four permissions (read, write, read permissions, and write permissions) allowable. At step 1, the user created an encrypted private object. At step 2, the user changed all four object permissions to be publicly allowable. This is where the mismatch occurred.

When the user creates the object and encrypts it, the expectation is that an encrypted object must first be decrypted to read it. This is accurate. However, AWS S3 *automatically* decrypts an object whenever an authorized user attempts to read it. This happens without any input from the user, which could defy the user's understanding of the encryption and decryption process because it deviates from conventional processes (wherein the only way a file is decrypted is by providing a private key or password, which is typically possessed only by the encryptor). By "encryption," the AWS S3 system means "server-side encryption," or encryption at rest. Users that choose to encrypt buckets or objects therefore protect the file while it sits at rest within Amazon servers, not when it is accessed by an authorized party, potentially resulting in the false security read permissions mismatch.

4.4.2.2 Write Mismatch

Our method discovered a situation where the user thought the system was secure from write conditions when it was actually vulnerable to them. At step 0 the user created an encrypted bucket with all four permissions publicly allowable. At step 1, the user created a private encrypted object. At step 2, the user made the object completely public, and at step 3 the user changed the object permissions again to allow authenticated users write permissions. This is where the mismatch occurred.

Though all public access to the object was denied, AWS S3's "authenticated users" capability is still dangerous, in that users can add a user to the "authenticated" list with nothing more than an email address. Because these are subject to typos, particularly where email addresses end in numerical characters, it is perhaps rather easy to inadvertently authenticate the wrong recipient and fail to notice this threat until it is too late. Granting an authenticated user write permissions over an object may seem safe when it really is not, resulting in the false security write permissions mismatch.

4.4.3 Blocking State Results and Interpretation

Our results indicate that no examples of blocking states were discovered. This indicates that users characterized by our method may be able to recover from the dangerous read or write conditions explored during our analysis.

4.5 Validating the Results

One way to validate our findings discovered through model checking is to determine whether conditions captured by counterexample execution traces are possible within the target system. To check, we opened a free Amazon AWS account and recreated all four mismatch conditions described in Sections 4.4.1.1–4.4.2.2. Our validation effort confirmed the existence of all states described in those mismatch conditions, as well as the progression of steps needed to realize those conditions. Our method's findings therefore correspond with potentially dangerous

cybersecurity vulnerabilities that real-world users could encounter while using AWS S3.

4.6 Discussion

We have provided an overview of the problem space, situating modern cybersecurity concerns within the context of configuration errors and the role that user mental models play with respect to these errors. We have generically formulated a method capable of exploring user mental models for potential misconfiguration states, leveraging Degani's *error* and *blocking* states to comment on their provenance and cybersecurity implications. We have demonstrated that our method can be used to discover these errors in real systems, investigating a cloud data storage solution for potentially dangerous user configuration errors and the threats they pose for modern computer system integrity. We then validated our findings by reproducing each of the model's identified conditions within the target system, confirming that the progression of steps by which each condition occur corresponds to a realistic path through the system.

Significant results and their implications are discussed below.

4.6.1 Extensibility of the Method to Other Systems

Our results demonstrate that Degani's *error* and *blocking* state architecture can be translated to comment on issues within a cybersecurity domain. We show that these results can identify the types of errors that users can commit when attempting to use or configure a digital system, as well as highlight the sequence of steps taken and how these can result in unforeseen vulnerabilities. While Degani's architecture was not initially developed to comment on cybersecurity vulnerabilities, validation efforts show that our method successfully integrates the architecture and that our *error* and *blocking* state findings correspond to real vulnerabilities within a target system. The method therefore represents a novel and successful extension of the architecture to a new domain. Our formulation's implementation of the target system also offers a significant opportunity for future development, in that analysts wishing to explore other file storage systems (such as Box, Dropbox, Google Drive, Mega, or NextCloud, among others) could eas-

ily instantiate their behavior and functionality within the method's framework. Analysts could then explore these systems for erroneous states as well.

4.6.2 Potential Dangerous User Misconfigurations

Results summarized in Section 4.4 demonstrate that our method is capable of finding mismatches between the system state and the user's mental model of the system state. Our results indicate that users can configure the target system in such a way that they believe their system is vulnerable to public users reading or writing the data stored with the target system, when it is really protected by the system. Conversely, users could configure it in such a way that they believe the system is safe from these conditions, when in reality their data is not protected. In all cases, the lack of correspondence between the user's mental model of the system and the actual system model could result in dangerous configuration errors.

Furthermore, the chain of user actions needed to realize these vulnerabilities are reasonable but involve multiple steps. All of the discovered erroneous configuration conditions could be executed with four or fewer steps, representing realistic combinations of actions for users attempting to secure documents within a cloud storage solution. However, this separation between user actions and cybersecurity effect—and indeed, the lack of direct causal links in some scenarios due to the unanticipated interactions between settings and results—can make it exceptionally hard for users to determine the origin of the error. This can be exacerbated by the temporal separation (potentially days, weeks, or even months) between the initial configuration and the discovery of its dangerous results. A lack of immediate feedback can mean that these misconfigurations can continue to do damage long after initialization and without the user's knowledge.

Finally, the consequences of these errors can range from inconvenient to catastrophic depending on the types of information stored within S3. Users storing backups of pictures already uploaded to image sharing or social media sites may not experience significant harm if left publicly readable, but users storing publicly readable tax returns or sensitive consumer information could have their identities stolen and experience financial loss (Whittaker, 2017).

Those who store sensitive government documents could face penalties for their carelessness (Leyden, 2017).

4.6.3 Mismatches Can Result from Non-normative Aspects of System Behavior

Our results also demonstrate that configuration errors occur because user understanding of the target system may fail to correspond with the system's functionality. Importantly, these mismatches can be resultant from the way design and behavior choices made by system architects are represented to users. For example, several mismatches discussed in Section 4.4 result from differences in the way users may generally understand encryption to work and the way in which AWS architecturally employs encryption (see Section 4.3.4.3 for details). Furthermore, while AWS does explain the principles by which their implementation of server-side encryption works, these explanations are buried in Amazon's extensive corpus of documentation (Amazon Web Services, 2018f). The ease with which users can get started building S3 buckets and uploading objects may also mean that users can do so without reading the technical guidance required to understand some of these less obvious system design choices.

Another mismatch discussed in Section 4.4, the false vulnerability write mismatch, refers to the system of public and private permissions used by AWS. Here, a public object remains public when placed inside a private bucket. The continuing public visibility of this object could seem counterintuitive to a user because the object has essentially pierced the veil of privacy offered by the bucket. This functions as a type of anti-pattern, acting counterintuitively to other filesharing programs and procedures where users can make folders and documents either wholly public or wholly private (Koenig, 1995). Furthermore, the bucket-level summary view provided within the S3 interface offers no dynamic indication whether or which public buckets contain public objects; rather, a generic alert (offered by an asterisk and page footnote) is displayed informing the user that objects "may" still be visible due to their control policy documents.[5] This effectively forces users to drill down into each bucket and *individually* examine each object's properties to determine whether or not they possess any public

[5]This functionality is current as of April 2018.

permissions. For situations where many buckets can contain hundreds or thousands of objects, which is entirely plausible in an enterprise setting, this task can be extremely tedious if not impossible.

Taken together, these factors suggest that our results are reasonable for the target system and that our method is capable of finding realistic scenarios where users could unwittingly fall into dangerous configuration states. Potential system improvements informed by these results are discussed in Chapter 6.

4.6.4 Potential Limitations of the Approach

4.6.4.1 Identifying Multiple Paths to Failure

Another way to validate our findings is to approach the mismatch conditions from a different direction. Our results described in Section 4.4 were discovered with specifications that were deliberately formulated in a generic fashion. This was done for two reasons. First, these properties follow the specification patterns identified in Section 4.2.2, and our goal was to demonstrate that Degani's *error* and *blocking* state constructs underpin our method and can lead to the discovery of real-world vulnerability conditions. Second, we wanted to do so without writing specifications purposefully designed to find these conditions. We argue that our ability to find vulnerabilities that fall out of our generic specifications speaks to the capabilities of our method and its strong theoretical basis within Degani's constructs.

Using more precisely-formulated specifications could yield additional insights regarding these conditions, including additional ways in which they can manifest. Based on details seen within the initial counterexample execution traces, we hypothesized that users could encounter a situation where they thought an object was private when it was actually public. We also wanted to further explore the observed encryption mismatches, hypothesizing that similar conditions could manifest in ways other than those discovered in the execution traces.

To that end, three specifications were used. The first addressed permissions mismatch conditions by searching for situations where the the user thinks the bucket and object are both private, yet the object remains publicly readable or writeable. The second addressed

encryption setting mismatches by searching for situations where the object is encrypted yet remains publicly readable. The third also addressed encryption setting mismatches, instead searching for situations where an object does not inherit the bucket's encryption setting in an expected way. Specification formulations can be found in Appendix A. Table 4.2 summarizes our verification analysis findings.

Table 4.2: Amazon AWS S3 Use Case Findings Validation Results

Permissions Mismatch			Encrypted but Readable			Surprising Inheritance		
States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result
3,777,972	1.19	✘	3,833,412	0.14	✘	2,200,836	0.38	✘

Note. A ✔ indicates a proof was returned, while a ✘ indicates a counterexample was returned.

Each specification produced a counterexample. In the first scenario, the user creates a public bucket in step 0, then a private object in step 1. In step 2, the user changes the object permissions to be public. In step 3, the user changes the bucket to be completely private, at which point the user expects that by making the bucket private, those permissions will be inherited by the object. This is incorrect, and objects with a public permissions structure will remain so even if the bucket is later reconfigured to be private. A permissions mismatch results.

In the second scenario, the user creates a public encrypted bucket in step 0, and in step 1 the user creates a private encrypted object. In step 2 the user makes the object readable by authenticated users. This results in a dangerous state where, while the user thinks the object is read-safe because it is encrypted, an (intended or mistakenly) authenticated user could read the document because it is decrypted by S3 before every authorized read or write request. An encryption setting mismatch results.

In the third example, the user creates an unencrypted public bucket in step 0, then a private unencrypted object in step 1. The user changes the bucket's encryption setting in step 2. The mismatch occurs here, because the user expects that the object will inherit the bucket's encryption setting. It will not. This is contrary to the user's expectation, because objects *do*

inherit bucket encryption settings when those objects are created, but this inheritance does *not* occur if bucket encryption settings are changed after the object already exists. This results in a mismatch between the object's encryption setting and the user's expectation of that setting.

While these conditions are similar in the general type of mismatch encountered, the series of actions taken by a user in each of these conditions are different from those discovered in Section 4.4. Furthermore, each of these findings were also manually reproduced within the AWS S3 system. This demonstrates that our method is sufficiently powerful to find a variety of unanticipated cybersecurity vulnerabilities within a real-world system, and that the method is robust to different exploratory techniques and specification formulation strategies.

4.6.4.2 Assessing the Implications of Diverse Mental Models

Our results offer compelling evidence that the method can identify real-world system misconfigurations and their cybersecurity implications. However, this is only true using the generic mental model instantiated within the method. It lacks the capability to determine how different types of user mental models could impact system security. This issue is addressed in Chapter 5.

5 | Extending the Method to Investigate Folk Models of Cybersecurity Vulnerabilities

> It bears emphasizing: our traditional ways of thinking have ignored—and virtually made invisible—the relationship between people and technology.
>
> *Kim Vicente*
> *The Human Factor*

The literature review in Chapter 2 discussed the concerns of researchers and practitioners over accelerating growth in cyberattacks that depend on actions taken by human users for successful attack execution (Collett, 2014). These actions are important because they can help attackers bypass the traditional network defense tools and countermeasures that otherwise make these attacks risky, difficult, and expensive to deliver and utilize (Verizon Enterprise, 2016).

In response, systems administrators and security departments have widely adopted cybersecurity awareness education and training programs for employees, but empirical research suggests that the improvements rendered by these training programs are mixed, particularly with respect to long-term success (Kumaraguru et al., 2009; Sheng et al., 2010; Siadati et al., 2017). The work in Chapter 4 offered one potential solution to this problem through the generic extension of Degani's architecture to the discovery of cybersecurity vulnerabilities.

Moving from training-based, reactionary vulnerability discovery procedures towards anticipatory cybersecurity tradecraft could help system security professionals remove vulnerabilities from play before they can be leveraged by threat actors. However, this approach may not scale to capture the variety of ways in which users can think about these vulnerabilities.

There may be additional strategies for improving computer security by more fully exploring the variety of user cyber threat models. One potential way forward identified in the literature review considers the efforts of some practitioners to explore folk model concepts within the domains of cybersecurity and human-automation interaction. These approaches could offer significant potential for improving the cybersecurity hygiene and practice of computer users because they offer rigorous strategies for exploring the actions that a plurality of users take when interacting with real-world systems. By synergistically integrating folk modeling concepts with the formal mental modeling architecture established in Chapter 4, it may be possible to suggest novel methods for defenders to bolster cybersecurity that extend well beyond traditional training approaches or insights gleaned from generic exploratory methods.

However, there do exist some potential difficulties in extending folk models into a formal framework. First, folk models can trade heavily in metaphor and are not runnable *qua* human factors mental models (Norman, 1983). The use of metaphor can result in a descriptive, rather than diagnostic, account of events or conditions with a system, potentially leading to a lack of detail useful for explaining system events and failures. Runnability enables the user to run her model forward to predict future states or consequences of actions based on the information of the present state, as well as run her model backward to explain the provenance of observed behavior (Norman, 1983). Runnability also eases the transition to a formal mental model representation that affords exploration using formal methods techniques (Degani & Heymann, 2002; Rushby, 2001b, 2002). These capabilities are critical for anticipating the cybersecurity consequences of user actions, but their reliance on metaphor and lack or runnability mean that this is very difficult to do with folk models alone.

Second, incorporating folk modeling concepts into the framework may help to realistically determine how specific users will run their mental models in domain-specific circumstances. Importantly, this makes folk models runnable, allowing analysts to extract insights from folk

models using powerful model checking and formal methods techniques. Our work also provides a theoretically-grounded method for constructing and exploring imperfect mental models, a capability that dovetails with observations from the human factors community about the nature and structure of mental models (Norman, 1983). These methods are application-agnostic, affording researchers a framework to potentially explore folk models in a number of cybersecurity areas.

In this chapter we will expand the generic architecture described in Chapter 4 to integrate formal folk modeling concepts. We will then describe our strategy for evaluating its ability to detect realistic examples of Degani's *error* and *blocking* states and potential cybersecurity vulnerabilities that result from our folk modeling analysis. We will then use a phishing email attack use case to investigate potential folk model-informed user responses to these threats, exploring the real-world implications of our findings and validating these results with a large open-source data corpus of phishing email responses. We will conclude with a general discussion of our findings and targets for future work in Chapter 6.

5.1 Integrating Folk Models into the Generic Formulation

We have integrated folk modeling concepts into our previous work regarding Degani's architecture and techniques in formal mental modeling by extending our generic approach presented in Chapter 4. This is described in Figure 5.1. Note the inclusion of folk modeling resources as an input to the analyst modeling process and is discussed further in Section 5.1.1 below.

Similarly to our generic approach, our expanded method allows a human analyst to examine information about cybersecurity within various external resources to construct a formal system model that captures a target cybersecurity application around a specific architecture. The analyst can also use this information with both generic and targeted specification properties to assert checkable properties about formally modeled concepts. By model checking these properties against the formal model, the analyst can identify if, and if so how, discrepancies between machine and user mental models can cause cybersecurity problems. Below we describe how the expanded method's inputs are formulated into the system model and specification

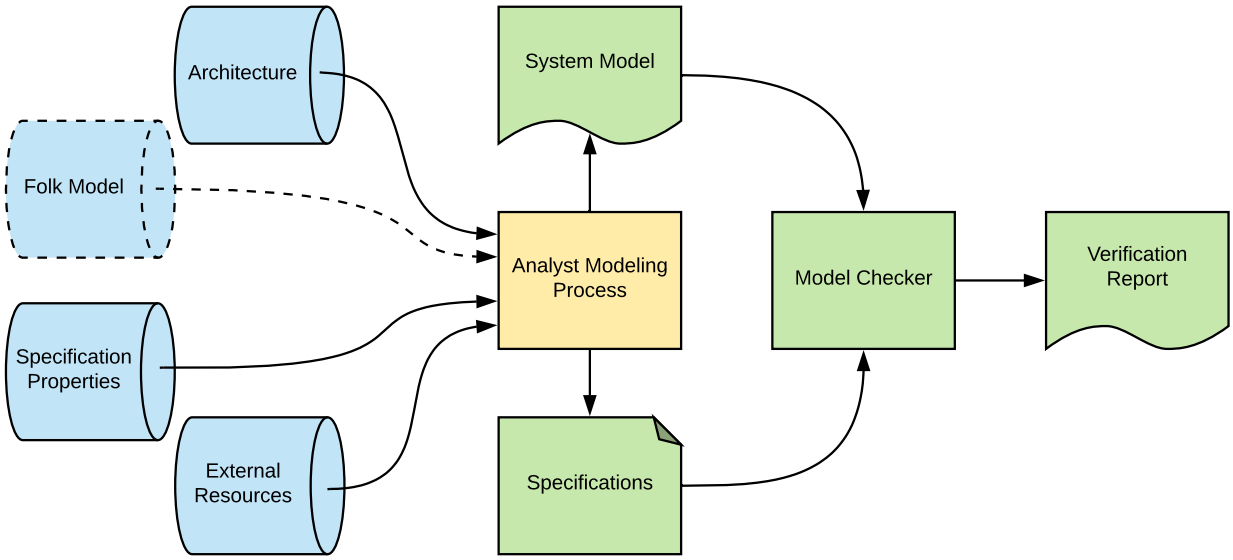

Figure 5.1: The generic formulation of our modeling framework.

properties, respectively, as well as how the model checking process is performed.

5.1.1 Architecture and System Model

Our novel formal modeling architecture, which extends the basic formal mental modeling concepts introduced by Degani and Heymann (2002), is shown in Figure 5.2.

This architecture represents a system as three synchronously composed modules: the `user`, `system`, and `comparator`. The `user` module represents the user's mental model, now augmented with folk modeling concepts, about how the system works. The `system` module captures the actual behavior of the system. Both are nondeterministic state transition systems and can transition between states based on input events: `environmental events` or `human actions`. These input events are directly coordinated between both the `user` and `system` module, eliminating the need for the `action generator` module used to coordinate actions in Chapter 4. In our method's generic formulation, mental models and system (machine) model states were each mapped to abstract concepts (such a legal or illegal states; or "high", "medium", or "low" states) that were important to the application being analyzed. This was extended from Degani's original work. Because we are explicitly concerned with the cybersecurity domain, we map user and system model states onto concepts important to cy-

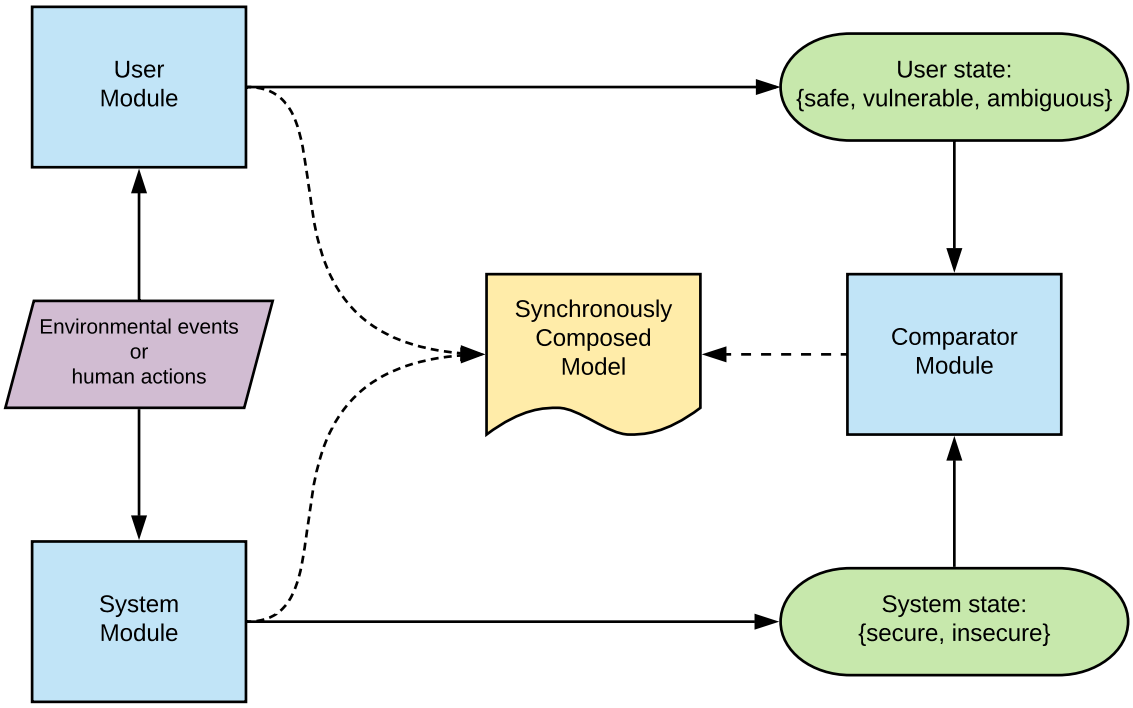

Figure 5.2: The architecture of the formal system model.

bersecurity. Specifically, the `user` model's states map onto whether the user thinks the system is safe, vulnerable, or ambiguous. This "ambiguity" abstraction was added because folk model concepts do not always clearly map to binary *safe* or *vulnerable* states, particularly if the user is potentially uncertain about system operational modes or the cybersecurity ramifications of their actions. The `system` model's state mappings indicate if the actual system is secure or insecure. The results of these mappings are taken as inputs to the `comparator` module, which computes whether mismatches exist between the `user` and `system` modules in the current state: if the user thinks the system is safe when it is insecure, vulnerable when it is secure, or if it is unclear (ambiguous).

The contents of the `user` and `system` modules must be derived from domain-specific external resources. However, the `user` module can also be heavily influenced by folk models. In particular, our architecture allows a folk model to be treated as an open parameter in the `user` module, where the value of the folk model can impact how events will change mental model state and how that state will map to safe or vulnerable. Therefore, when constructing

Table 5.1: Folk Models and Adherence to Security Advice

Computer security advice		Folk models			
		Graffiti	Burglar	Big Fish	Contractor
1.	Use anti-virus software	∅	3	1	1
2.	Keep anti-virus software updated	∅	∅	∅	1
3.	Regularly scan computer with anti-virus	∅	∅	∅	1
4.	Use security software (firewall, etc.)	2	2	2	1
5.	Don't click on attachments	3	3	∅	∅
6.	Be careful downloading from websites	2	2	1	1
7.	Be careful which websites you visit	3	3	2	3
8.	Disable scripting in web and mail	∅	∅	∅	1
9.	Use good passwords	2	∅	2	1
10.	Make regular backups	3	1	1	1
11.	Keep patches up-to-date	3	3	1	1
12.	Turn off computer when not in use	2	3	1	1

3: "It is very important to follow this advice."
2: "Following this advice might help, but it isn't all that important to do."
1: "It is not necessary to follow this advice."
∅: "This model has nothing to say about this advice, or there is insufficient data from interviews to determine an opinion."

Note. A summary of survey user data regarding the importance of following security advice with respect to their folk models of hackers. Descriptions of the importance levels appearing in quotations has been taken directly from Wash, 2010, p. 10, and the remainder of the table has been adapted.

the `user` module, the analyst will use the information contained in Table 5.1 to determine how these transitions and mappings should occur. These folk models are derived from Wash (2010) and are discussed in Section 2.4.2 above. This table contains three types of information: the computer security recommendations (or "advice") presented during the interview and data collection task; the four folk models of hackers identified through data analysis; and the folk model-cross-advice assessments given by interviewed participants who identified with those specific folk models. Here, Wash provides user responses to the importance (or lack thereof) of following each security recommendation is given. They have been translated here into numerical values between 3 and ∅. 3 signifies where users felt it very important to follow this advice; 2 signifies where users felt that following this advice may be important to computer

security, but is not terribly important to do; 1 signifies where users felt a particular piece of advice was not important to follow; and ∅ signifies where either users had no comment on the piece of advice or Wash felt the data insufficiently robust to offer commentary (Wash, 2010, p. 10).

Note that the twelve security recommendations on the left were collected by Wash from reputable computer security websites, including resources provided by Microsoft[1] and the United States Computer Emergency Readiness Team, or US-CERT.[2] These were intended to capture a significant number of best practices that home users could deploy for effective defense. The four folk models of hackers on the right of Table 5.1 were extracted through Wash's data analysis, and represent different attitudes towards threat actors that people using a computer may encounter. The numerical importance ratings given for each assessment were adapted from the coding provided in the original table.

5.1.2 Specification Property Patterns

Our extended method identifies several novel specification property patterns that can be used with model checking to discover mismatches between the `user` and `system` modules. All of these extend the error state and blocking state concepts introduced by Degani and Heymann (Degani & Heymann, 2002). Because we map `user` and `system` module states onto cybersecurity-relevant concepts (safe, vulnerable, secure, insecure, and ambiguous), we have three distinct error states that each mean something different for cybersecurity.

Introduced in Chapter 4, we continue to use generic specification property patterns to discover mismatches between the `user` and `system` modules. For example, the expanded method searches for the false vulnerability and false security mismatches generically captured in Specifications 4.1 and 4.2, and these specifications search for conditions where the user thinks the system is vulnerable when it is secure, or where the user thinks the system is secure when it is vulnerable, respectively. We also search for Degani's *blocking* conditions, where the

[1]The URL provided in Wash's work now redirects to an enterprise-level cybersecurity resources page: https://www.microsoft.com/en-US/security/default.aspx

[2]This URL from Wash's work is now broken; see: https://www.us-cert.gov/home-and-business

user is potentially unaware that a system action is possible and indicates a situation where the user is unable to bring the system out of a dangerous state. This was captured in Specification 4.3.

Our extension of the generic model to include folk modeling concepts has led to the introduction of additional specifications. As discussed in Section 5.1, two folk model-cross-attack evaluations for Item 5 have been have given ∅ assessments. Our model accounts for this lack of clarity through an intermediate condition called `ambiguity`, signifying that the user could be either safe or vulnerable; it does not apply to the system's security state, as for our purposes a computer is either secure or insecure. Practically speaking, an "ambiguity" mismatch identifies situations where, if parameters of the current model were to be changed to other values possible for those parameters, a counterexample could result. This principle can be applied to (4.1) and (4.2), resulting in:

$$\text{NoAmbiguousFalseVulnerabilityMismatch} \models \mathbf{G} \neg (user = ambiguous \wedge software = secure), \text{ and} \tag{5.1}$$

$$\text{NoAmbiguousFalseSafetyMismatch} \models \mathbf{G} \neg (user = ambiguous \wedge software = insecure). \tag{5.2}$$

While similar ambiguity properties could be formulated for blocking state conditions, we found little analytic utility in doing so.

Finally, when using model checking to prove whether specifications are true, a failure will only ever show one failure sequence in a counterexample. There could yet be others. For this reason, an analyst may want to add additional specificity to a property to allow the model checker to verify if a property holds assuming certain other conditions. For example, an analyst may want to check that a user with a particular mental model will never have recovery blocking.

To accomplish this, we can create conditional versions of all three of our general specification properties. Thus, for a specific *condition* that the analyst wants to assume, we can

reformulate all of the specifications as follows:

$$\text{ConditionalNoFalseVulnerabilityMismatch} \models \mathbf{G}\ (condition \Rightarrow \neg\ (user = vulnerable \wedge software = secure))\ , \tag{5.3}$$

$$\text{ConditionalNoAmbiguousFalseVulnerabilityMismatch} \models \mathbf{G}\ (condition \Rightarrow \neg\ (user = ambiguous \wedge software = secure))\ , \tag{5.4}$$

$$\text{ConditionalNoFalseSecurityMismatch} \models \mathbf{G}\ (condition \Rightarrow \neg\ (user = safe \wedge software = insecure))\ , \tag{5.5}$$

$$\text{ConditionalNoAmbiguousFalseSafetyMismatch} \models \mathbf{G}\ (condition \Rightarrow \neg\ (user = ambiguous \wedge software = secure))\ , \text{ and} \tag{5.6}$$

$$\text{ConditionalNoRecoveryBlocking} \models \mathbf{G}\left(\begin{array}{l} condition \Rightarrow \\ \neg\left(\begin{array}{r} (user = vulnerable \wedge software = insecure) \\ \text{AND } \mathbf{X}\ (user = vulnerable \wedge software = secure) \end{array}\right) \end{array}\right). \tag{5.7}$$

It should be noted that $a \Rightarrow b$ describes a logical implication relationship that is true if a implies b. Furthermore, because the `comparator` module from our architecture (see Figure 5.2) does the comparisons between `user` and `system` states, this makes asserting all of the desired properties for checking rather straightforward. When combined with the three generic formulations from Chapter 4 identified above, we now have ten checkable specifications available for use in this expanded method.

5.2 Case Study: A Specific Application of the Method

5.2.1 Overview of the Phishing Email Use Case

We investigated cybsersecurity vulnerabilities introduced into a computer system through the actions that users take in response to receiving a malicious email. As discussed in Chapter 2,

phishing is an activity whereby the recipient of a malicious email inadvertently shares sensitive information or installs malware on their computers (IBM, 2016). Phishing emails can use different tactics to execute an attack, for example by asking victims to click a link and log into the spoofed portal of a familiar web service, or by asking the victim to download and read an attachment secretly laden with a malware payload. While these methods may be different, all phishing attacks are made in an attempt to gain access to information that can be used for nefarious purposes. This information can be used to steal identities, execute financial fraud, steal corporate or national security information, and gain access to otherwise protected systems or networks. These attacks are rather dangerous because, while they may seem like mundane emails to users, they can surreptitiously circumvent sophisticated network defenses by tricking users into voluntarily executing malware or giving up their credentials (Hong, 2012).

Below we describe how we applied our method to this application area.

5.2.2 Formal Modeling

Using the architecture outlined in Figure 5.2, we implemented a formal system model describing the domain around our mental modeling concepts. This was done using the input language of the Symbolic Analysis Laboratory (SAL) (de Moura et al., 2003). SAL was appropriate for our work because it is an extremely expressive formal language, is supported by a suite of different model checking tools, and has been used in a number of formal human factors analyses (Bolton et al., 2013; Oliveira et al., 2017).

In constructing the formal model, we considered Wash's folk models from Table 5.1 as well as external resources that describe cyberattacker tactics. Below we describe how each of these were formulated into our model by having the `user` and `system` modules respond to events and actions. In this particular application, our analysis of cyberattacker strategies ultimately resulted in the inclusion of four human actions that could trigger changes in state. Three of these represented specific attack events that could be initiated by human actions: `Link`, for clicking on a malicious link in an email; `Download`, for downloading a malicious file from a website; and `Attachment`, for opening a malicious attachment in an email. The

last, `No Action`, represented a situation where the human did not do anything that would affect the security state of the system. In all cases below, a full listing of the model code is available in Appendix B.

5.2.2.1 The User Module

The `user` module addresses the capabilities of a human agent interacting with digital information. For example, the agent can examine the content of an email or a website and make a determination about whether any information requested for entry is *sensitive* (personally identifiable information, like Social Security or credit card numbers) or *basic* (seemingly simple, non-personal web browsing) in nature. The `user` module also handles aspects of human agent folk models and the method by which these models inform the user's response to system events or actions. This is accomplished through a series of guarded transition statements that prohibit the execution of a response until all preconditions are met. Each guard requires two pieces of precondition information: a folk model type and a phishing email attack type. Once the precondition is met and the guarded transition is triggered, the model is capable of inferring the sequence of states resultant from each interaction for each specific combination of environmental events and human actions. This allows our model to explore specific interactions between mental models and potential cybersecurity threats, investigating in detail their ramifications for system security, as well as the resilience of users with specific folk models to different phishing attack techniques.

After computing the potential changes to resources, the `user` module will update its *user state* variable to either *safe*, *vulnerable*, or *ambiguous*. This signifies the user's own sense of whether the computer system is safe from or vulnerable to cybersecurity vulnerabilities based on the actions that have been taken, or whether this could change depending on whether a particular analysis could unfold differently if one or more model parameters were changed. The `user` module then pushes the output of this state change to the `comparator` module (see Figure 5.2).

In our formal model, the specific behavior in the user model was derived from items 5, 6,

7, and 11 of the four folk model concepts captured in Table 5.1. Items 5, 6, and 7 relate to email use and web browsing, and 11 is related to patching and software integrity of a computer system and the software it operates. For example, consider the `graffiti` folk model's approach to advice item 5 ("Don't click on attachments"). Those with this folk model consider this advice very important to follow, and the implication is that clicking on attachments will put the user in danger of compromise. This was used to inform construction of the guarded transition statement shown in Figure 5.3 in our formal model. This describes a situation where the user with a `graffiti` folk model receives an attachment that could be potentially hostile and opens it, then in the next state the user will be vulnerable (`user' = vulnerable`).

```
[] model = graffiti AND actionType = hostileAttachment -->
   user' = vulnerable;
```

Figure 5.3: An example guarded transition statement.

Note that, where appropriate, we have taken a conservative but realistic interpretation of the security advice important levels. If in the above statement the user with a `graffiti` folk model rated the attachment advice importance as 2 rather than 3, the transition statement may be formulated slightly differently to reflect these concerns with respect to computer security vulnerabilities. This helps ensure that our model logic maintains correspondence with the real-world cybersecurity threats that users face from malicious emails and does not stray too far from Wash's original findings.

A more complex example of our process for incorporating folk model concepts into our formal model is associated with the `burglar` folk model. Here (see Table 5.1), those with the `burglar` model rate the importance of the advice "Be careful downloading from websites" as 2—in the words of Wash, "It might help, but it isn't all that important to do." This rating, when synergistically integrated with information from external resources, informs the guarded transition statement in Figure 5.4.

This describes a situation where a user with a `burglar` folk model receives an email requesting the user to download a document from a third-party website, and the user does so. In this situation, the user's security posture will not change in the next state if no personally-

```
[] model = burglar AND actionType = websiteDownload -->
  user' = IF siteContent = basic AND system = secure
  THEN user
  ELSE vulnerable
  ENDIF;
```

Figure 5.4: A second example guarded transition statement.

identifying information is requested by the download and the user's software is fully patched and up-to-date (`user' = IF siteContent = basic AND software = secure THEN user ... ENDIF;`). If one of those conditions does not hold, then in the next state the user will be vulnerable (`user' = ... ELSE vulnerable ENDIF;`).

Here, we have included a conditional statement that changes the user's sense of security based on the context of the email (the `siteContent`). It also contains logic to check the current status of software security tools in place. While the latter is a simple status question that has clear security ramifications, the former is a determination of the user herself. Therefore, if the user erroneously determines that the site is asking for basic content, when in reality the user is somehow providing sensitive information, then this user could mistakenly consider herself potentially safe. The implications of this design choice, as well as the possibility of extending this model to include more nuanced user approaches to these security concerns, is explored in Section 5.5.

5.2.2.2 The System Module

The `system` module addresses the features of the computer system being used in this application. This includes the various functions and status indications of security software, features of web browsers and the website advertisement blocking extensions running within it, and the capabilities of the email client program being used. It also handles the system-level cybersecurity ramifications of any exploits that a user may activate by opening an email or performing other system actions. This is accomplished by checking the exploit's attack strategy against the computer's active system defenses, allowing compromise only if the defense against that exploit is not running or is inadequate. For example, if security software is running but remains unpatched, the exploit will succeed. This checking process is also capable of distinguishing

whether a defense is appropriate for a given exploit. If a user downloads a document that contains a malware payload, the module will check the exploit type against the running defenses. If the system is running an advertisement blocker but does not have fully-patched security software, the exploit will succeed because malware embedded in an email opened within the email client generally cannot be interdicted by a web browser's ad blocker. These conditions are reflected in our model.

To ensure that accurate system behavior has been captured, we made use of several external resources that describe cyberattacker tactics, techniques, and procedures (also known as TTPs) in an effort to maintain correspondence between our formal model and real-world computer security threats. Two major sources of record were used. First, the MITRE PRE-ATT&CK Matrix (2018) details seventeen attack strategy categories and 173 distinct exploit techniques that adversaries use to gather information and establish a foothold in target systems before the actual attack begins. Each technique provides a broader description of its use and intent, an assessment of the difficulty for effective use by an adversary, and an assessment of the difficulty for detection by common defenses.[3] This matrix also provides descriptions for several relevant exploit strategies, including phishing with malicious attachments, phishing with malicious links, and loading malware onto target computers through drive-by downloading or other web browser-delivered exploits. Second, we have also included insights gathered from The Hacker Playbook 2, a compendium of attacks and exploits that can be used for penetration testing and other nefarious purposes (Kim, 2015). Several sections provide descriptions regarding cybersecurity exploits and their use in compromise strategies, including the execution of social engineering and phishing attacks.

In addition to details about phishing strategies and attacks, resources describing the features and functions of modern web browsers and computer systems were also integrated. These resources include components of website advertisement blockers, as well as details about broad classes of modern malware packages that generally accompany emails and the techniques these payloads use to covertly execute once email attachments are downloaded. In

[3]Common defenses here refer to intrusion detection, intrusion prevention, and malware or anti-virus programs installed on computers and networks. Because the ATT&CK Matrix assessments are provided in context of enterprise-grade computer defenses, a home computer user is certainly included as well.

all cases, these details were reviewed and confirmed with appropriate documentation.[4]

After computing the potential changes to resources, the `system` module will update its *system state* variable to either *secure* or *insecure*. This signifies whether the computer system is safe from or vulnerable to cybersecurity exploits based on the user actions that have been taken and whether appropriate defenses are in place to mitigate the threats contained in the opened email. The `system` module then pushes the output of this state change to the `comparator` module. See Appendix B for further details.

5.2.2.3 The Comparator Module

The comparator module was implemented as described generically above. Its calculations regarding whether mismatches exist between the `system` and `user` modules were used to assert specification properties (see Appendix B).

5.2.3 Specification Properties

Using the conditional specification property patterns from (5.3)–(5.7), we asserted specification properties against our formal model. In these analyses we wanted to prove that false vulnerability and false safety mismatches as well as recovery blocking could never occur for each of the four included folk model types and the four events (human actions) that could occur. This results in a *condition* of the form $(model = folktype \wedge actionType = act)$ where *folktype* is the folk model type and *act* is the type of action event. Thus, our five different specification types, four folk model types, and four actions resulted in $5 \times 4 \times 4 = 80$ specification properties.

5.2.4 Apparatus

Each of our 80 specification properties were verified against our formal system model using version 3.3 of SAL's symbolic model checker (SAL-SMC).[5] Analyses were conducted on a

[4] Our browser security behaviors were modeled after security best practices of Mozilla's Firefox browser. Documentation can be found here: https://www.mozilla.org/en-US/security/.

[5] http://sal.csl.sri.com/index.shtml

desktop computer with a 3.7 GHz quad-core Intel Xeon processor and 128 gigabytes of RAM running the Debian 9 Linux distribution.

5.3 Verification Results

Results from our preliminary investigations of the phishing use case are presented in Table 5.2 for false vulnerability mismatch specifications, Table 5.3 for false safety mismatch specifications, and Table 5.4 for recovery blocking specifications. The four supercolumn entries in each table represents the type of specification that was iteratively checked against the four folk models for each of the human action events.

Each supercolumn contains three subcolumns. `States` indicates how many states within the statespace were visited by the model checker. `Time`, in seconds, indicates how long the model checker took to return either a proof or a counterexample. `Results` indicates whether the model checker returned a proof or a counterexample: ✔ indicates that the formal verification process yielded a specification proof, while ✘ or ● indicate that a regular or "ambiguity" counterexample was returned, respectively. Due to the strategy with which our specifications are formulated, a proof signifies that no evidence of the condition under consideration was found. Conversely, a counterexample signifies that that condition was discovered. This approach was taken so that if a condition exists for that folk model, the counterexample returned would provide an execution trace showing exactly how that condition was discovered. Because this trace provides a complete account of all states that led to the violation, this information could be potentially used to suggest user interface or system architecture design remediations. More on this discussion can be found in Section 5.5. We discuss the results in each table in more detail below.

Table 5.2: Folk Model-Cross-Attack Verification Results: False Vulnerability Error States with Potential Ambiguity

	Link			Download			Attachment			No Action		
Folk Model	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result
Graffiti	798	0.06	✘	816	0.06	✘	834	0.06	✘	930	0.06	✔
Burglar	798	0.07	✘	816	0.06	✘	834	0.06	✘	930	0.07	✔
Big Fish	816	0.07	✘	978	0.06	✔	858, 840	0.07, 0.07	✔ ●	1002	0.07	✔
Contractor	810	0.06	✘	978	0.08	✔	858, 846	0.06, 0.05	✔ ●	1002	0.06	✔

Note. A ✔ indicates a generic proof was returned, a ✘ indicates a generic counterexample was returned, and a ● indicates that an "ambiguity" counterexample was returned. Where ● is absent, an "ambiguity" proof was returned.

Table 5.3: Folk Model-Cross-Attack Verification Results: False Safety Error States with Potential Ambiguity

	Link			Download			Attachment			No Action		
Folk Model	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result
Graffiti	846	0.07	✔	834	0.06	✘	858	0.06	✔	906	0.07	✔
Burglar	846	0.06	✔	834	0.06	✘	858	0.06	✔	906	0.08	✔
Big Fish	864, 906	0.07, 0.06	✘ ●	930	0.06	✘	858, 870	0.06, 0.07	✔ ●	960	0.06	✔
Contractor	846	0.07	✔	954	0.07	✘	858, 870	0.06, 0.06	✔ ●	966	0.06	✔

Note. A ✔ indicates a generic proof was returned, a ✘ indicates a generic counterexample was returned, and a ● indicates that an "ambiguity" counterexample was returned. Where ● is absent, an "ambiguity" proof was returned.

Table 5.4: Folk Model-Cross-Attack Verification Results: Blocking States

	Link			Download			Attachment			No Action		
Folk Model	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result	States	Time (s)	Result
Graffiti	834	0.06	✔	834	0.06	✔	834	0.07	✔	834	0.06	✔
Burglar	834	0.07	✔	834	0.05	✔	834	0.07	✔	834	0.06	✔
Big Fish	834	0.07	✔	834	0.06	✔	834	0.06	✔	834	0.05	✔
Contractor	834	0.06	✔	834	0.07	✔	834	0.06	✔	834	0.07	✔

Note. A ✔ indicates a proof was returned, while a ✘ indicates a counterexample was returned.

5.3.1 False Vulnerability Mismatch Verification Results

Table 5.2 summarizes results for verifications of false vulnerability mismatch specifications [see Equations (4.1), (5.1), (5.3), and (5.4)]. Here, a counterexample was returned if the model checker encountered a state where the user erroneously considered the computer system vulnerable to potential cyberattack, but the system was actually safe because it had the proper mitigations in place to handle the threat posed by the received email.

These results indicate that mismatches between user and system security error states occurred for users with `Graffiti` and `Burglar` folk models against all three types of threats, as well as for all folk models against phishing emails leveraging malicious links in their attack strategies. Furthermore, we discovered no counterexamples for situations where no action was taken with the received email. Our framework also returned "ambiguity" counterexamples for `Big Fish` and `Contractor` folk models against phishing attacks leveraging malicious attachments. While no direct path to failure (depicted with ✘) was discovered for these specific instances during model checking, the "ambiguity" counterexamples suggest that direct paths to failure may be possible for the specified attack strategies depending on the specific combination of scenario parameters used. These cases therefore merit further analysis to determine if their combinations sensibly represent these attacker strategies in a realistic manner. Discussion on this point is offered in Section 5.5.1.2.

We also find that proofs and counterexamples were returned quickly, always in less than one tenth of one second while checking around a thousand states during model checking. This offers a promising baseline that can be used to explore the scalability of our model if more complex user actions and phishing attack strategies are developed in the future.

5.3.2 False Safety Mismatch Verification Results

Table 5.3 summarizes results for verifications of false safety mismatch specifications [see Equations (4.2), (5.2), (5.5), and (5.6)]. Here, a counterexample was returned if the model checker encountered a state where the user erroneously considered the computer system safe from potential cyberattack, but the system was actually vulnerable because it lacked the proper

mitigations in place to handle the threat posed by the received email.

Our results suggest that mismatches between user and system security error states occurred for users with `Big Fish` folk models when specifically responding to email phishing attacks leveraging malicious links, as well as for users with all folk model types against attacks using malicious downloads. No vulnerabilities were discovered for users taking no action against a received email.

Our framework also found potential "ambiguity" counterexamples for `Big Fish` folk models against Link and Attachment attacks, as well as for those with `Contractor` folk model types against attachment attacks. Similarly to results in 5.3.1, direct paths to failure may be possible for these attacks depending on the specific combination of scenario parameters used. The discovery of an "ambiguity" mismatch for the `Big Fish` folk model with respect to link-based attack vectors indicates that, while a direct path to failure was discovered with the current model, yet further paths to failure could be detected if further combinations of model parameters are explored. The implications of this finding are discussed in Section 5.5.1.2.

5.3.3 Blocking State Verification Results

Table 5.4 summarizes verification results for recovery blocking specifications [see Equations (4.3) and (5.7)]. Here, a counterexample indicates a state where the user thinks that the software is vulnerable to a given email's attack method and the software is insecure (that is, they are in agreement) and the system can be brought back to a safe state, but the user fails to understand how to do so.

Our results indicate that no folk model-cross-attack interactions result in a counterexample. This means that users opening malicious emails could potentially know how to recover from these dangerous conditions. Further discussion is offered in Section 5.5.1.3.

5.4 Model Validation Using Real-world Data

One way to assess the validity of our method is to explore counterexample execution traces to assess whether the results agree with observed system behaviors and external cybersecurity reference materials. Another is to use data collected from real-world systems and user actions to determine whether our anticipated results correspond with actions taken by users in the wild. Though less targeted than the detailed investigation of counterexamples and the information they contain (see Section 5.4.3.1), these comparisons can be useful for refining models, addressing some concerns over validity, and commenting in a broader sense on the cybersecurity risks that real users face within computer systems. Here we discuss our validation data sets, highlight key findings from our analyses, and identify some limitations of our effort.

5.4.1 Validating the False Security Model Analysis Results

Our false security validation effort used a large corpus of data on user responses to simulated phishing emails collected by Cofense, a company that performs employee phishing response testing and preventive cybersecurity training.[6] Cofense uses a standardized testing platform available to free and paying enterprise customers that offers nearly two dozen phishing email templates, and from these templates the information technology professionals performing phishing assessments can select one or more simulated phishing emails to send to employees. These emails are designed to look very similar to FedEx shipping notifications, attachments from the human resources department, requests to share Google Docs or Dropbox links, password reset requests, and other common phishing email tactics (sometimes called "lures"). Additionally, Cofense continually refines their template emails based on the observed success rates of real lures seen in the wild.

The data used for our validation effort was published by PhishMe prior to acquisition in their "Enterprise Phishing Susceptibility and Resiliency Report" (2016) and clarified through a series of personal communications via email with a technical support staff member. PhishMe collected this data over an 18-month period of time, wherein 40 million simulated phishing

[6]The business that performed and published these results, PhishMe, was acquired by Cofense in early 2018.

emails written in fifteen different languages were sent to employees in 1,000 organizations, including Fortune 500 and public sector entities, that spanned nearly two dozen different industries (p. 2). In addition to a series of metrics on email responses, PhishMe also collected data on user responses, time to click (the delay between users receiving the email in their inbox and taking action on the email), whether those emails were subsequently reported to in-house IT departments, and other data discussed elsewhere in the report.

Results were reported in aggregate, identifying the rates at which users clicked on the most dangerous templates and the attack strategy that each of those template emails used used (p. 5). Each template used one of three different attack strategies:

- **Click-only:** An email that urges the recipient to click on the embedded link;
- **Data entry:** An email with a link to a customized landing page that entices employees to enter sensitive information; or
- **Attachment-based:** An email that includes a seemingly legitimate attachment saved in a variety of file formats.

These results have been summarized in Table 5.5. Here, the thirteen templates with the highest response rates are listed, with each response rate value indicating the percentage of all recipients of that template email who clicked on it. Each template's response rate is categorized under the attack strategy with which it corresponds, and we have computed a relative ratio of response rates so that a general picture of the response landscape could be established. It should be noted that information regarding template attack strategies was taken from each template's description within PhishMe's simulation framework and was verified for accuracy by their technical support staff.

From these data we find that users showed higher response rates for click-only phishing email attacks than the other two attack strategies, and we interpret this to suggest that users are more likely to fall for link-based attacks that only require users to click a link. Furthermore, the click-only emails with the greatest susceptibility values mirror the types of emails that employees commonly see in corporate environments, such as files scanned and delivered from network-connected printer/scanner machines and notifications that packages have been delivered. The relative ratios presented Table 5.5 suggests that lures leveraging data entry and

Table 5.5: Cofense Phishing Simulation Responses by Attack Strategy

Template	Click-only	Data entry	Attachment-based
File from scanner	31.1		
Package delivery—alternate	25.8		
Unauthorized access		24.8	
Digital fax	20.2		
Package delivery	19.7		
Scanned file			18.4
RSA phish	13.1		
Password survey		12.9	
Secure email			12.2
Adobe security update	7.0		
Restaurant gift receipt			6.7
Google docs link	6.0		
Funny pictures			3.4
Relative ratios	.610	.187	.202

Note. The values reported here indicate the percentage of users that clicked on the email out of all users who received it. For example, 31.1% of all users receiving the "file from scanner" phishing email clicked on it.

attachment-style attacks were clicked on less frequently than click-only lures, suggesting that users fell for emails containing links than the other two attack strategies.

Because our results explore situations where users unwittingly fell for phishing emails of different types, our effort to validate the method should compare the PhishMe email phishing data in Table 5.5 with the verification results summarized in Table 5.3. To do so, we must first acknowledge the discrepancy between Wash's attack strategy framework (link-, download-, and attachment-based attack strategies) and PhishMe's framework (click-only, data entry, and attachment-based attack strategies). While the first and third strategies in both sets are highly similar, the second strategies do not align. Though such a dissimilarity is unfortunate, it may be simply due to a difference between the modeling and data collection entities used in this work rather than a fundamental disagreement over the cybersecurity threat landscape. We therefore suggest that comparisons should not be made between the two incongruous attack

strategies, instead focusing our validation efforts on the two remaining and highly similar attack strategies.

When these adjustments are made we find significant agreement between our verification results and PhishMe's collected data. Three factors contribute to these findings. First, our method's only generic counterexample for the link- and attachment-based attack strategies occurs for malicious links, but there exist no generic counterexamples for attachment-based attacks. While a proof indicates that there are no possible modes of failure, a counterexample indicates that the model checker has discovered at least one path to failure. Additional paths could exist, signifying an analysis space where further failure modes (and, within the context of our model, additional ways in which users could erroneously believe their system to be secure and click on a phishing email link) could be found once the failure mode identified in the counterexample is addressed. This signals the potential for a more complex analysis space than just a single mode of failure for a single folk model-cross-attack interaction.

Second, this conclusion is reinforced through the discovery of "ambiguity" counterexamples at particular intersections of folk models and attack strategies. While two exist for the `Big Fish` and `Contractor` folk models under attachment-based attack strategies, these occur alongside generic proofs. The more interesting case exists in concert with the `Big Fish`-cross-Link generic counterexample. The generic counterexample signifies additional possible modes of failure, but these are constrained to the different ways in which only the currently-selected model parameters can interact. The "ambiguity" counterexamples, on the other hand, signify the potential existence of other modes of failure with *different model parameters*. This indicates the potential for other modes of failure if different model parameters are explored, and could also account for the disparity between the more significant PhishMe response rates of link-based emails and the lesser, yet still observed, response rates for attachment-based attacks.

Finally, Wash included counts of the individuals who possessed each of the four folk models, establishing the prevalence of these models among the sampled individuals (Wash, 2010, p. 8). The `Burglar` folk model was most common, followed by `Big Fish`, `Graffiti`, and `Contractor` being the least common. Our counterexamples therefore associate potential

modes of failure with one of the more popular folk models, an observation that lends support to the weight of the phishing response distribution resting on the click-only attack strategy. When taken together, these results seem highly favorable towards an initial validation of our method and offer promising directions for future exploration (see Section 6.4).

5.4.2 Validating the False Vulnerability Model Analysis Results

We also attempted to validate our false vulnerability findings in Table 5.2 through real-world data collected by PhishMe. This data was published in their "Enterprise Phishing Susceptibility and Defense Report" (2017) and discussed a more expansive testing and monitoring procedure, summarizing the aggregate results of 52.4 million simulated emails from 1,400 organizations that included the 40 million sent during the 2016 report's testing period with data from further simulations conducted during a follow-on phase (p. 3). The 2017 report also discussed real-world phishing email reporting statistics gathered from PhishMe's enterprise clients, which included "over half of the Fortune 100" and covered organizations in more than 50 countries (p. 2).

Our method's false vulnerability analysis explored the problem space for situations where users erroneously believed the system was potentially vulnerable to suspected phishing emails, even though the system would have been secure. These conditions would have therefore resulted in an overaggressive attitude towards identifying these harmful emails, perhaps resulting in higher rates of phishing email reporting to IT security teams, among other deleterious effects. Data describing the rates at which users report phishing emails could therefore lend at least partial empirical support to our false vulnerability validation efforts.

Results from the 2017 technical report were presented in aggregate, summarizing trends extracted from 216,000 emails reported to PhishMe in 2017. These are summarized in Figures 5.5 and 5.6 and have been adapted from Figures 11 and 12, respectively, in PhishMe's report. Figure 5.5 shows that, of all emails reported, a vast majority (85%) were either non-malicious or spam emails. As a policy, PhishMe considers spam and phishing emails distinct from one another: spam is a generally harmless "unsolicited commercial message," while phishing emails

"[attempt] to lure a victim to a dangerous link, an attachment, or give up a password," even though spam and phishing emails may use the same bulk email delivery techniques (Higbee, 2017). The remaining 15% is comprised of what PhishMe terms "crimeware," which encompasses malware or dangerous links that are targeted and directly harmful. Most reported emails were therefore generally benign in nature.

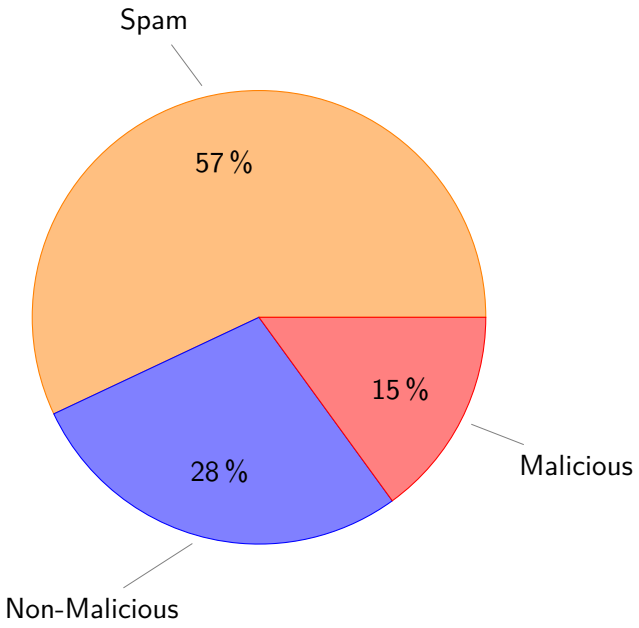

Figure 5.5: Categorization of emails reported to PhishMe in their 2017 report.

Figure 5.6 disaggregates the 15% of truly malicious reported emails into their prominent attack types. These results indicate that a majority of those emails (71%) leveraged malicious URLs, a small number (7%) used attachments, and a middling number (17%) used a combination of these attack types. Here, PhishMe reports that emails within the "combination" category could use malicious links or attachments that leverage hostile Microsoft Office macros. The smallest selection, Business Email Compromise (BEC), typically target C-suite executives and use more sophisticated attack types to authorize the release of corporate information, authorize wire transfers, or otherwise request additional business transactions or activities. It is not immediately clear from their technical report whether this is the reason PhishMe has chosen to break BEC attacks out into their own category.

We interpret these data to signify that users are likely to report emails as malicious when

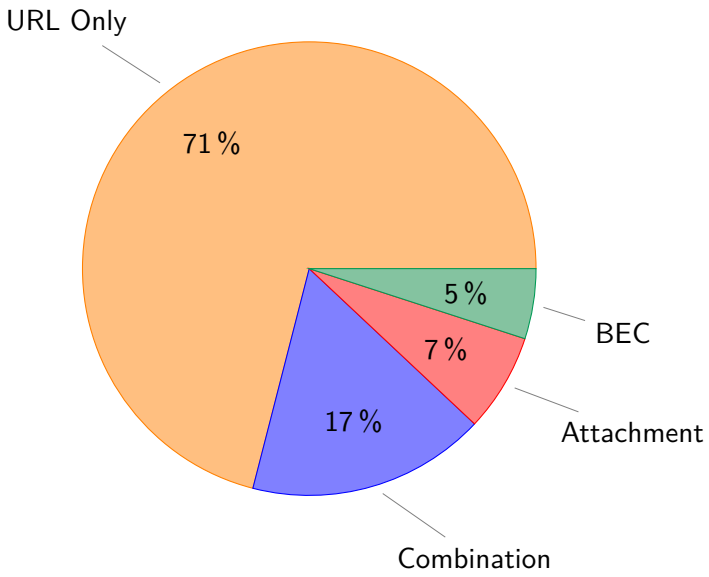

Figure 5.6: Confirmed malicious emails by attack type.

they actually are not, and of those reported emails most of them contain links as compared to other attack types. Therefore, our results presented in Table 5.2 strongly agree with the data provided by PhishMe in Figures 5.5 and 5.6. We draw these conclusions for two reasons. First, because spam emails are much more likely to contain a link that users could easily click on versus other types of information or exploits, such as attachments, we argue that at least a majority of the emails reported as non-malicious plausibly contain links. This conclusion agrees with our external cybersecurity reference materials (Camp et al., 2009; Kim, 2015). Second, of the confirmed-malicious reported emails, at least 71% of them use a link-based attack strategy to attempt user compromise. Together, these findings indicate that our method's assertions regarding the overreporting of potential link-based phishing emails are reflected in the real-world data collected by PhishMe. Users are therefore perhaps wise to be cautious of emails that contain links, but this caution can lead to overly aggressive reporting conditions highly similar to those identified by our method.

5.4.3 Potential Limitations of the Validation Efforts

While our efforts to validate the method's results using real-world phishing data have yielded strong initial indicators of agreement between our results and real-world conditions, there yet remain some potential limitations to these efforts.

5.4.3.1 Using Aggregate Data to Comment on Formal Models

Using large-scale data analysis and comparisons to comment on the accuracy of a formal model risks diverging from the classic use and intent of formal methods. While the use of real-world data can provide a window into the formal model's correspondence with the world, this data may emphasize broad findings that fall out of aggregate analysis at the peril of conclusions with lesser significance.

This stands somewhat in opposition to the historical use and computational power of formal methods techniques, where emphasis is instead placed on the discovery of unforeseen, specific combinations of events that lead to modes of failure. Generalizing from these specific events to large-scale, trend line analyses may not perform very well, in turn potentially underplaying the power of the method when anticipated results are not discovered in large sets of data. Though our validation efforts look favorably upon the method's results, any model explored with formal methods techniques need not live or die by the hand of validation through statistical analysis.

5.4.3.2 Alignment Between Modeling and Validation Sources of Record

Collecting data on user responses to cybersecurity threats can pose unique ethical and legal challenges, as the types and methods by which this data is recorded must remain legal under state and federal laws such as the Computer Fraud and Abuse Act (CFAA). Violating these laws can result in felony charges for a broad range of activities involving the access of information (both deliberately or unintentionally) on nearly any computer or electronic device, including information which may be deemed "personally sensitive" or for which the investigating party should have no expectation of access; furthermore, the CFAA has been written in a way that

allows for substantial flexibility in its interpretation and the activities falling within its purview (18 USC § 1030, 2008; Urban, 2011).

Using aggregate data already published by a third-party organization, rather than information collected specifically for use in the dissertation, thus ameliorates potential legal hazards incurred while verifying our method's results. However, this secondary data may not align as closely with the model when compared to data primarily collected for model validation, reducing the apparent strength of the agreement between our model's results and the data used for validation. While it remains unclear whether this misalignment affected our validation efforts, further investigation is needed to definitively address these concerns.

5.4.3.3 Strategies to Address these Limitations

One way to address these limitations is to explore the actions that users take when responding to phishing emails at a more granular level. Our method is capable of producing detailed counterexamples that highlight the direct paths to failure possible for each folk model-cross-attack interaction. These results are powerful, but may not appropriately scale up in such a way as to be apparent in aggregate, *post hoc* data that cannot be explored in ways congruent with the method. Collecting more specific data could offer human analysts a way to determine how often the exact situations predicted in counterexamples are seen among real user responses. This may be possible in more controlled corporate environments where computer use and activities are monitored with greater fidelity, and is certainly possible in experimental environments. Pairing click-level system and user activity recording with PhishMe's realistic phishing simulation environment would offer a unique opportunity to collect data better suited for extended validation efforts.

Another way is to conduct more targeted validation exercises specifically gathering the types of data needed to validate our method. For example, it may be that the phishing data collection and reporting framework used by PhishMe gathers an adequate amount of information on the types of threats that users face, how often phishing events occur, and detailed reporting information, among other concerns. Researchers with broader access to this infor-

mation could plumb its depths more thoroughly, making more precise comparisons between the attack strategies identified in the method and the attacks suffered by users in the wild. Validation through aggregate reported data is useful for evaluating initial developments and identifying targets for future work, but it is much more difficult to use these data for detailed analyses. Access to more complete data sets could be very useful for later targeted investigations that could be conducted in a manner less intrusive (and more realistic) than high fidelity controlled user monitoring or experimentation.

5.5 Discussion

The work presented here demonstrates a number of theoretical and applied contributions to the formal methods, human factors, and cybersecurity communities. Broadly speaking, our work offers a new approach for discovering potentially unanticipated cybersecurity vulnerabilities stemming from user responses to email phishing attacks.This approach incorporates several novel implementations of existing work from these domains. First, we have extended Degani's architecture and formal analysis concepts developed in Chapter 4 to the problem of email phishing attacks and cybersecurity, introducing a strategy for modeling and investigating the effects of human-automation interaction and erroneous user actions in response to phishing emails. Second, we have developed a method for the re-implementation of existing folk models as runnable constructs, enabling their use in more rigorous applications that lend themselves to exploration using rigorous formal analysis protocols. Third, we have rolled these developments into a novel framework for exploring a realistic, prescient end-user cybersecurity application.

We offer a discussion of our results and their implications below.

5.5.1 Interpretation of the Findings

5.5.1.1 False Vulnerability Error State Results

The results summarized in Table 5.2 characterize situations where users erroneously believe the system to be vulnerable when it is actually secure. In practical terms, counterexamples indicate scenarios where users could be overly aggressive in assuming that their computer is vulnerable to an exploit enclosed in the received email, perhaps refusing to open them or needlessly flagging them as dangerous when in fact they are benign or the system would be secure against their payloads. While it is generally recommended that all email users err on the side of caution with respect to potential phishing emails, if users overzealously characterize emails as hostile then it could result in reduced productivity as important emails could be missed. It could also result in increased resource allocation from helpdesk tickets and information technology (IT) administrator overhead, because certain details extracted from emails reported as phishing attempts in a corporate environment are added to spam filters and used to create firewall rules. This is undesirable.

The distribution of counterexamples presented in our results suggest that recipients with all four folk models risk being excessively suspicious of emails that contain links, requests to download files, or attachments. However, this is more likely for users receiving emails containing links as compared to the other two phishing attack types, with users thereby potentially reporting emails containing links as phishing attacks to corporate IT helpdesks more often than the others.[7] One potential way to verify these results would be to examine phishing email reporting statistics for differences between rates associated with each of the attack strategies, and we would expect to see emails containing links more frequently reported as potential phishing attacks than emails asking users to download documents or view attached files (see Section 5.4.2).

However, it is important to note that the magnitude of these results do not necessarily signal dramatically lower rates of compromise from successful link-based attacks as compared to the

[7]Note that this interpretation could be different if detailed future investigation of the "ambiguity" counterexamples yields a direct path to failure.

others. A user may attempt to manage the risk posed by these emails and open them anyway, cognizant of the risk but perhaps succumbing to the persuasiveness of the subject line ("Wall Street Journal story: Our company just filed for bankruptcy!") or placing sufficient trust in the apparent sender to override her suspicion ("Read this message from your retirement account manager about your account's closure"). Cybersecurity specialists should therefore not assume that users are less likely to fall for phishing attacks using certain strategies simply because users are highly suspicious of those emails.

Finally, we find these results to have high face validity with respect to our external resources, particularly with respect to situations where users with all folk model types could incorrectly believe their system to be vulnerable to emails that contain a link. Because it can be difficult for users to identify hostile emails in overloaded inboxes, "never click a link," "if in doubt, report it," and other categorical, easy to remember maxims are popular in corporate cybersecurity training and awareness programs. It is therefore somewhat expected to find results suggesting that users may be overly suspicious of any links, regardless of their actual hostility.

5.5.1.2 False Safety Error State Results

The results summarized in Table 5.3 characterize situations where users erroneously believe the system to be safe when it is actually vulnerable. We interpret these results to indicate that users are less likely to recognize threats posed by download- and link-based attack emails, potentially resulting in higher rates of compromise for these attack strategies. We would therefore expect to see higher rates of effectiveness for these attack strategies in real-world phishing data collected from actual users.

Our results indicate that download-based phishing attacks are potentially the most effective vector that threat actors could use. Phishing attacks leveraging this strategy often ask users to download a necessary file ("Adobe Flash Player is out of date, click here to update it") by either clicking a link in a desktop popup, a website advertisement, or in a received email or document. Because this vector's medium can be highly refined and appear rather convincing to an unwary user, it is unsurprising that downloads pose a significant threat to the types of

mismatches identified in our results and therefore seem to have high face validity with respect to our external cybersecurity resources.

We also note the discovery of an "ambiguity" counterexample associated with the `Big Fish` folk model and the link-based attack vector. While other "ambiguity" counterexamples were associated only with specifications that returned generic proofs, here we find one associated with a generic counterexample. This discovery indicates further potential counterexamples for this specific folk model-cross-attack combination and signals that a potentially deeper and worthwhile analysis space may exist at this intersection.

We therefore feel that these results merit further investigation because, in addition to further `Big Fish`-cross-Link analysis opportunities, real-world examples of download attacks can simultaneously leverage link-based attack vectors to accomplish their work. From a theoretical perspective, this can make it difficult for our model's results to clearly convey differences between link- and download-based attack vectors with absolute certainty, meaning that some of the counterexamples found for download attacks may in fact result from false security mismatches attributable to link-based attack vector characteristics. This can be due, in part, to a lack of clarity within the original data collected by Wash (2010) that has been integrated into our approach to model representation, and only through rigorous investigation of the folk models with our newly-developed ability to make them runnable is this murkiness exposed.[8] From a practical perspective, this means that link-based attack vectors may be more dangerous than Table 5.3's graphical representation seems to imply at a glance. Cybersecurity experts should therefore not discount the danger posed by these link-based email phishing attacks.

5.5.1.3 Blocking Error State Results

The results summarized in Table 5.4 indicate that users with all four folk models could potentially recover from dangerous conditions initiated by opening a phishing email.

[8] Though it may be possible to add clarity to our formal model and eliminate this murkiness, doing so would require the introduction of assumptions and risk departure from Wash's original findings. A perhaps more responsible approach would be to explore these findings with follow-on investigations that instead augment the paper's results.

5.5.2 Implications for Cybersecurity Training Programs

Our results demonstrate that the method is capable of determining how users with different mental models of cybersecurity threats may respond to various phishing email attack techniques. As researchers and practitioners search for ways to improve the security of digital systems, these results shed light on ways in which user mental models could offer targets for increased system security. For example, we found that users may be more vulnerable to phishing emails asking users to download files (such as an Adobe Flash update) as compared to other vectors, and that user vulnerability spanned all four folk models. At a minimum, this means that organizations supporting employee or member cybersecurity training should perhaps increase their focus on these types of threats. More broadly, our results offer evidence suggesting that those responsible for conducting cybersecurity training programs should allocate time and resources differently to different threats, as opposed to one or a few training sessions treating all threats equally.

Our results also suggest that training efforts could be more effective, both in value and efficiency, if training regimens could be customized based on the "folk model threat profile" of users. For example, suppose that an organization had designed and administered questionnaires or surveys (similar to the folk model elicitation techniques used in Wash, 2010) to all employees to assess their particular folk model of cybersecurity threats, perhaps during their hiring process. Security professionals could then tailor each employee's awareness training program to their specific folk model, encouraging them to look more closely at links, downloads, or attachments before clicking. This may prime employees to be more mindful of specific threats and improve the long-term effectiveness of these training programs (Kumaraguru et al., 2009; Sheng et al., 2010; Siadati et al., 2017).

It is possible that our findings could support other improvements in user cybersecurity, particularly where changes to network security protocols (as opposed to end-user training programs) are concerned. We offer further discussion of these implications in Section 6.4.

6 | Discussion and Future Work

> We predict the future. And the best way to predict it is to invent it.
>
> *The Well-Manicured Man*
> *The X-Files S3E1, "The Blessing Way"*

6.1 Revisiting our Research Objectives

This work investigated the role of erroneous user actions in the inadvertent creation of cybersecurity vulnerabilities, where these erroneous actions can result from mismatches between the user's mental model of a computer system's state and the actual state of the system. In this chapter we first revisit our research questions and describe how our work satisfies each question's intent (Section 6.1). Second, we discuss our planned and already-published literature resultant from the work (Section 6.2). Third, we describe our work's specific contributions to the formal methods and human factors communities (Section 6.3). Finally, we offer targets for future work (Section 6.4).

Our work was motivated throughout by two fundamental research questions (Section 3.1). Here, we demonstrate that our architectural developments and findings satisfy both.

6.1.1 Revisiting Research Question 1

Our results pursuant of RQ1 in Chapter 3 demonstrate that exploring the ways in which users interact with computer systems can help analysts understand the precise manner by which

erroneous human actions and settings misconfigurations can result in unanticipated cybersecurity vulnerabilities. Our method's formal techniques shed light on the specific combinations of actions by which these vulnerabilities can be introduced into a system. For example, results presented in Chapter 4 identified the precise sequences of steps that, when taken by a user, could result in a cybersecurity concern. These results were also trivial to reproduce within the actual target system.

By understanding the sequence of steps users could take to introduce these vulnerabilities, network defenders could identify common missteps committed by users and institute reminders or alerts within the interface to preempt these dangerous conditions before they occur. For instance, if users expect public objects put into a private bucket to remain private, network defenders managing S3 deployments could scan for these conditions and alert users to their potential blunder. Alternatively, Amazon system interface designers could institute an alerting feature (perhaps through a modal popup, or persistent icon) to denote that a private bucket yet contains a public object.

Additional results in Chapter 5 indicated situations where users with different folk models of computer security threats could take unwittingly dangerous actions when interacting with phishing emails, potentially endangering the security and integrity of their personal computer or entire shared corporate network. For example, we discovered that users with the Big Fish folk model—the second most prevalent of all folk models under investigation—could be susceptible to phishing emails leveraging malicious links as the method of compromise. However, users with that particular folk model were less susceptible to emails that leveraged malicious attachments as the lure. Because more users have this type of folk model, systems administrators could improve network security by increasing email scanning or filtering to subject all emails with embedded links to additional scrutiny.

These results become substantially more powerful if a corpus of even generic user folk models could be formulated. Rather than instituting generalized network defenses, cybersecurity countermeasures could be tailored to the "folk model threat profiles" of individual users. For example, if in Chapter 5 GEICO was able to profile the folk models of all corporate employees, network defenders could augment Alice (a Big Fish folk model user)'s personal computer

defenses to scrutinize links more closely, but provide Bob (a Contractor folk model user)'s computer with additional protection against emails that may leverage malicious attachments. Further discussion on potential future work on this topic is offered in Section 6.4.

Ultimately, our method and its results help analysts and network defenders understand the role that human actions play in the security of complex digital systems. Within the cases explored in this work, humans are not willfully negligent in their actions, clicking furiously on every incoming email with the intent of making defenders' lives more difficult. Rather, the modern digital machinery of a 21st century workforce can be difficult for users to completely understand, and within that complexity lies the potential for mismatches between user mental models and system characteristics. Understanding these mismatches, the vulnerabilities they raise, and their potential solutions is critical for a modern, user-focused approach to cybersecurity.

6.1.2 Revisiting Research Question 2

Correspondent with RQ2 in Chapter 3, our method advances the current state of practice in formal methods techniques by synergistically integrating developments from formal mental modeling, folk modeling, and cybersecurity into a formal architecture. The generic formulation of the method (Section 4.2) integrates Degani's *error* and *blocking* concept and extends it to the cybersecurity domain, a first-of-its-kind effort. The generic method also instantiates a generic user mental model based upon established formal mental modeling techniques (Section 2.5.2). This integration results in runnable mental models that, when model checked, yield insights regarding the interaction between user mental models of a computer security tool (in Chapter 4 a cloud data storage service) and the tools themselves.

Results from the first investigation showed that our method successfully identified specific combinations of user actions that could potentially yield unanticipated cybersecurity vulnerabilities. These combinations were described in the counterexample execution traces generated by the model checker. Without formal methods techniques, analysts would not be provided with such clear and complete accounts of these security vulnerabilities and the ways in which

they arise.

Results from the second investigation showed similar results, describing situations where users with specific folk models may be susceptible to emails leveraging certain phishing attack techniques. As in the first case, development of these capabilities within a formal methods framework would may not have yielded such specific insights regarding specific folk model-cross-attack vulnerabilities. This would have reduced the effectiveness of our method and perhaps left hidden some of the ways in which user folk models can lead to specific phishing attack vulnerabilities.

These techniques are also readily extensible to a variety of scenarios. This is possible for two reasons. First, we have implemented our architecture as generically as possible by avoiding the provision of highly-specific computer system components or capabilities. For example, our generic formulation in Chapter 4 eschews any computer- or device-specific characteristics from its architecture (see Figure 4.1). Its computational feature set generically describes nearly any computing device, and only where necessary (for example, when describing how the AWS permissions or encryption features work) are more specific components introduced.

We then completely strip the AWS-related features out of the method in Chapter 5, preserving the core generic architecture and replacing the old content with new capabilities specific to an email program on any computational platform. Though the application is completely different, maintaining the architecture across both examples allows us to use a highly-similar approach to discover examples of mismatches between user mental models of the target system and the system itself. These investigations yield further novel insights, discussed above and in Section 5.3.

6.2 Publications Resultant from the Work

Several publications to discuss our findings and their implications are planned for the immediate future.

6.3 Specific Contributions

6.3.1 Adapting Degani's Method to Cybersecurity Applications

The extension of Degani's human-automation interaction (HAI) analysis concepts to a cybersecurity domain represents a significant contribution to both the formal methods and HAI communities. The product of this extension is a novel architecture for modeling cybersecurity applications in a manner that can shed light on the human's role and function within both of them.

Our work on adapting and extending Degani's notions of *error* and *blocking* states has also led to the development of new specification patterns. These patterns provide formal methods and HAI researchers with generic, broadly-applicable strategies for rigorously describing and discovering Degani's states in a new domain. This means that others can easily leverage our work and extend it in their own manner, fostering further investigations into these and related domains based on our contributions.

By applying our method to a cloud storage domain, we have specifically identified how seemingly benign configuration choices can unexpectedly lead to dangerous system vulnerabilities. Because Amazon dominates the cloud data storage market and user growth rates continue to increase, the rate at which threat actors explore S3 resources with malicious intent is likely to follow suit. This work offers a number of novel contributions that can help network security professionals more effectively lock down system resources and prevent potentially catastrophic cybersecurity vulnerabilities.

These efforts have also led to a series of targeted specifications for analysis within this domain using our novel property patterns. The first one, focused on the ability of users to recover from undesirable security conditions, can be extremely helpful in exploring the future cybersecurity consequences of user missteps, such as clicking on unknown links or changing system settings on their own. Within a modeling context, researchers and practitioners armed with this analytic capability can look forward within modeled conditions and preemptively introduce system or design changes before these missteps become data breaches. Additionally,

we created a series of targeted specifications that offer a suite of analytic capabilities to explore the implications of different resource configuration strategies. These can be quickly adapted and deployed within other approaches to modeling cybersecurity vulnerabilities, offering a strategy for our contributions to affect positive change without requiring the development of a new, extended modeling framework.

6.3.2 Making Folk Models Runnable

Our work makes a second significant contribution by extending folk modeling concepts to Degani's architecture, meaning that they can now be run and explored using formal methods techniques. We believe this to be the first-ever use of folk models in this context, offering a powerful new strategy for using folk models to explore cybersecurity topics. The benefit has been twofold. First, the use of folk models within a cybersecurity context could be a helpful strategy for information technology teams to think about their organization's cybersecurity approaches. It may be possible, following Wash's work, to offer interviews, surveys, questionnaires, or other data collection methods to determine the folk model that best represents user thinking about cybersecurity threats. If true, these results could comment on user folk models in a general sense and from there practitioners would generally need to transform that information into realistic system remediations. Our method offers a new way to explore the ramifications of these folk models in detail and arrive at concrete system improvements through formal model analysis.

Additionally, our work extending analysis capabilities to user folk models allows these analyses to target specific *types* of users, rather than generically modeled users or individual users from which detailed mental models have been elicited. Provided the folk models of users within a sample could be determined, our method could provide a more rapid "triage" capability for assessing the consequences of potential cyberattacks. It would be much more difficult to maintain an effective tempo if more granular mental models were required for formal analysis, and therefore our method offers an interesting approach to this type of broader analysis.

The second contribution offered by extending runnability to folk models provides a way to

improve the rigor of folk models in a general sense, addressing some of the criticism levied by researchers trying to use them in more robust ways. As discussed in Chapter 2, traditional folk models can rely heavily on metaphor to describe the implications of their findings and sometimes have difficulty offering actionable recommendations. Furthermore, folk models have been used elsewhere in human factors research and other domains, including other HAI applications (Kempton, 1986; Lewis, Sycara, & Walker, 2018), folk models of health concerns such as diabetes (Greenhalgh, Helman, & Chowdhury, 1998) and HIV transmission (Kubicek et al., 2008), and education (Torff, 1999), among others. By offering a generic strategy that could extend runnability to other folk modeling work, these contributions could be of direct benefit to research far afield of our original framework and use case explorations. The feasibility of this and other approaches mentioned in this section should be explored in future work.

6.3.3 Discovering Unanticipated Cybersecurity Threats

Our third significant contribution is the discovery of unanticipated cybersecurity consequences from user interactions with phishing email attacks, a finding that aligns with one of our primary goals for this work. For example, we found that users with certain folk models may be more resilient than their counterparts to phishing emails leveraging specific exploits. This was unexpected. Information regarding these differences is generally absent from corporate network defense and user counter-phishing training protocols, perhaps because it is difficult and costly for supervisors to explore their employee base for different folk models, or because it is difficult to implement these findings as actionable steps.

The perspective that users may vary in their susceptibility to different phishing exploits could lead to tailored training procedures for employees, where—if the user's folk model can be determined—the recommendations and activities presented to the user during training could offer increased awareness or other remediations for users more susceptible to malicious attachments versus weaponized hyperlinks. We are so far unaware of any cybersecurity training regimen that offers these capabilities with rigor. Future work should explore the interface between our method and these potential training approaches.

6.3.3.1 Novel perspectives on anticipatory cybersecurity tradecraft

Couched within our third contribution is an additional discrete development: a novel perspective on anticipatory cybersecurity tradecraft (ACT). ACT is derivative from the interest of the US government and other public- and private-sector organizations in anticipatory intelligence. This is the capability to forecast the actions taken by threat actors and estimate their potential national security or defense ramifications (Intelligence Advanced Research Projects Activity, n.d.). In broad strokes, pursuit of an "anticipatory intelligence" doctrine attempts to shift the locus of control over an engagement from the attacker to the defender. In other words, the defender can gain an upper hand within an engagement by thinking and acting in an *anticipatory* manner, rather than *reacting* to the events and operational tempo of an attacker (Office of the Director of National Intelligence, n.d.).

ACT is anticipatory intelligence applied within a cybersecurity context, particularly where cyber-counterintelligence and -counterespionage efforts are concerned (The Department of Defense, 2015). Cyber threat actors often maintain a significant advantage over the targets they compromise, potentially dwelling for weeks or months inside defender networks without being detected (Mandiant, 2018).[1] To move from being reactive to anticipatory, network defenders should understand the tradecraft of their attackers, including their tactics, techniques, and procedures (or TTPs) and how those TTPs interface with the vulnerabilities of the network. Defenders can then preemptively patch systems, shuffle network resources, or take other actions to eliminate attacker opportunities before they are exploited.

Our method offers a novel defensive perspective on ACT by modeling how users may respond to phishing threats or inadvertently introduce vulnerabilities into network-connected resources. The method's results could be used to inform defender actions, identifying potential vulnerabilities that align with attacker TTPs so they can be mitigated before an attacker can exploit them. While cybersecurity professionals are likely to take anticipatory steps to identify and mitigate software or hardware vulnerabilities, our method offers a strategy to address anticipatory cybersecurity tradecraft from the perspective of human "wetware" vulnerabilities.

[1] The global average for dwell time—the time an attacker spends inside the target network without being detected—was 100 days, or about 14 weeks, in 2017.

6.4 Future Work

6.4.1 Further Development of the Method

Though Chapters 4 and 5 have identified potential limitations of our method and ways to address them, those limitations correspond with specific components of the method. There are several ways in which our general method could be improved as well.

6.4.1.1 Additional Recoverability Specifications

Our method currently uses a series of *blocking* specifications to discover situations where the user is blocked from returning the system to a safe state because she is effectively prevented from knowing about this option by the system. Future work could extend this investigation to determine whether, based on the current cybersecurity danger facing the human-computer system, it is *ever* possible to recover from such a condition. While our current specifications check whether recovery is possible but the user is unaware of how to do so, additional recoverability specifications could determine whether recovery is even possible within the system's current capabilities.

To search for this possibility, the analyst would check a property of the form:

$$\textbf{A G}\ ((software = insecure) \Rightarrow \textbf{E F}\ (software = secure)). \tag{6.1}$$

This asserts that for all paths through the model (**A**) it should globally be true that if the software is insecure, then there exists (**E**) at least one path where in the future (**F**) the software should return to a secure state. Its formulation is trivial, but to search for such a condition the analyst must use Computation Tree Logic (CTL), a formulation of mathematical logic that is different from the Linear Temporal Logic (LTL) used by the SAL-SMC model checking tool. Due to differences in the semantics of these logics and how they explore concepts of reachability within either Kripke (as in LTL) or branching-time (as in CTL) computational constructs,[2] LTL and CTL logics are incompatible (Baier & Katoen, 2008, Ch. 6) and cannot be simultaneously

[2]This is abstractly captured in their different names: *Linear* Temporal Logic versus *Computation Tree* Logic.

checked by SAL-SMC (de Moura et al., 2003, p. 29).

However, there do exist solutions to this problem. One could use a model checker that supports CTL or CTL* (a superset of CTL and LTL that combines operators from both logics), such as SPIN (Visser & Barringer, 2000). SAL also offers `sal-wmc`, an experimental witness model checker that could address these specifications' CTL formulations.

6.4.1.2 Incorporating Stochastic Model Checking Capabilities

Our method's current implementation explores the statespace of each checked scenario for conditions that could result in cybersecurity vulnerabilities. Here, the model checker returns a proof if a condition (captured in a formal specification) does not exist, or a counterexample if a condition does exist. This is useful when human analysts must determine with certainty whether potential vulnerabilities exist within the modeled system, regardless of their rarity. However, our model checking technique does not indicate the likelihood of particular vulnerabilities or whether some are more or less common than others. This could be useful if human analysts detect multiple potential vulnerabilities but must prioritize cybersecurity resources to remediate them. A sense as to whether some vulnerabilities are more likely than others to occur could help allocate scarce resources more efficiently.

The formal methods community has developed several stochastic model checking tools and techniques to aid in this process, offering commentary on the likelihood of specific failure modes in addition to determinations whether the observed undesirable characteristics do or do not exist (Baier & Katoen, 2008, Chap. 10). These techniques are usually similar to traditional model checking tools in that they conduct reachability analyses of a given statespace, but they must also calculate the actual likelihoods of system transitions in order to inform the transitions that are executed (Kwiatkowska, Norman, & Parker, 2007). These methods have been successfully used to explore human-automation interaction and the function of systems that rely on human behavior (Feng, Humphrey, Lee, & Topcu, 2016; Feng, Wiltsche, Humphrey, & Topcu, 2016).

While there exist other techniques, variants of Markov chains are commonly used to achieve

these capabilities, for example through the use of discrete-time Markov chains or Markov decision processes. In the former, all transitions are deterministic because successor states are chosen based upon a fixed probability distribution associated with each successor transitions, or edge of the directed graph. In the latter, successor states are still chosen probabilistically, but one or more different probability distributions for system transitions exist; once a probability distribution is chosen nondeterministically, the successor state is chosen deterministically based on the selected distribution (Arnold, Gebler, Guck, & Hatefi, 2014). When used in model checking, these Markov systems are sometimes formally expressed in Probabilistic Computation Tree Logic, or PCTL, a derivative of CTL that offers an extended set of formalisms for representing probabilistic systems (Baier & Katoen, 2008).

Future work should investigate whether our framework could be integrated with probabilistic model checking techniques and tools, such as PRISM (Kwiatkowska, Norman, & Parker, 2011).

6.4.1.3 Expanding Wash's Folk Models to Incorporate Modern Cybersecurity Threats

The cybersecurity threat landscape is constantly shifting as new software is built, vulnerabilities are discovered and mitigated, and both attackers and defenders refine their tactics. This highly-dynamic environment means that even recent research findings can be rendered outdated within a short period of time. Our analysis of Wash (2010) resulted in the inclusion of link-, download-, and attachment-based threats into our expanded method. While we found strong potential agreement between our analysis of user folk model vulnerabilities to phishing attack techniques, the threat landscape has changed since 2010.

Notably absent from Wash's folk models of computer security threats are data-entry attacks, where users are asked to log in to landing pages or web app portals that seem legitimate but are in reality cleverly-designed impersonators (such as those for Gmail, Dropbox, Facebook, or other services). Date-entry attacks are the most common type of link-based phishing email exploit and are potentially different from Wash's link-based folk model attacks, where a user clicks a link, is taken to a malicious web page, and is served harmful content through malver-

tising, drive-by downloading, or other passive means (Proofpoint, 2017). Though both attacks use links as their initial mode of delivery, the ability to break each attack out into its own category could deliver useful additional insights regarding how modern users approach these threats. This would also bring Wash's folk modeling findings in line with more recent data on phishing attacks, including those discussed in Cofense (2016).

To solve this issue, it should be possible to replicate the survey and data analysis procedures documented in Wash (2010) to determine the degree to which folk models of computer security threats have changed. Because popular press coverage regarding cybersecurity vulnerabilities, data breaches, and phishing has generally increased within the past several years, users may be more inclined to take these threats seriously and refine their folk models accordingly. Once these data are collected and integrated into our method, follow-on analysis could explore whether these changes result in new findings and whether these results are borne out through validation.

6.4.1.4 Integrating Complete AWS Bucket Policy Sets

In Section 4.3.3.1 we discuss two methods by which AWS S3 controls access to system resources: policies and Access Control Lists, or ACLs. These control documents are used to establish which users have permission to read, modify, or delete objects and buckets. To demonstrate its capabilities, our method uses a simplified permissions structure that contains bucket and object ACLs. Our method does not address the incorporation of complete bucket policies, which are more complicated, user-generated documents that can contain one or more of up to 68 different permissions and operations types (Amazon Web Services, 2018c). Now that the method has been debugged and we have shown that it can be used to discover real-world vulnerabilities, further method development could focus on the integration of the complete bucket policy permissions set. Though it will add a significant degree of complexity to the method and could pose scalability challenges for formal analysis (see Kneuper, 1997), this development could result in additional discoveries relevant to more advanced AWS S3 resource configuration.

6.4.1.5 Counterexample Visualization

Some of the power afforded model checking for analyzing human-automation interaction is through counterexample generation, where an exact accounting of all system states leading to a specification violation is produced. However, the execution traces can be difficult and time-consuming for human analysts to read, and this problem can be exacerbated if the model uses a large number of system variables to compute state changes (Bolton, 2010). For example, the generic model outlined in Chapter 4 used 43 discrete system variables, and SAL's default Symbolic Model Checker counterexample visualization view displays each variable's assigned value at each analysis step. This means that the method's longest execution traces showed 258 variables over six analysis steps. The human analyst must therefore compare each complete trace with the trace preceding it, searching for differences in variable assignments to discover the source of the error.

Future method development should explore strategies for visualizing counterexample execution traces, and there has been some recent work on these concerns. For example, the consistent ordering of variables across analysis steps means that a computer program may be able to ingest the trace and display the results in an interactive graphical user interface or table (Chechik & Gurfinkel, 2007; Kermelis, 2003). Others have visualized the state sequence diagrams, allowing users to explore discrete nodes and check their temporal logic properties (Aljazzar & Leue, 2008). Differences between the execution steps could be highlighted, directing the human analyst's attention to noteworthy results; it may also be possible to ask the analyst for specific patterns or insights in which she is interested prior to analysis (Simmons & Pecheur, 2000). Because there exist in the literature several examples of counterexample visualization techniques that have helped analysts explore execution traces for interesting insights, it should be possible to extend these findings to our method and provide analysts a tool set for results analysis. These developments could be particularly useful in domains where cybersecurity is a primary focus and our method is used as a specific "COTS" (consumer off-the-shelf) utility, such as government-affiliated laboratories or research programs (see Section 6.4.2.3).

6.4.2 Additional Areas of Exploration

Our method may be capable of investigating other pressing issues within the cybersecurity domain. A selection of these potential developments is outlined below.

6.4.2.1 Investigating Physical Exploit Delivery Strategies

Our method focuses on cybersecurity vulnerabilities that result from mismatches between user mental models of software program functionality and how those programs actually function in the face of potential security threats. This approach has high face validity because most threats that employees or computer users face are so-called "over-the-wire" threats, where attacks are electronically transmitted to victims over the internet through an email account or other means of digital communication.

However, physical attack vectors also pose a significant threat to organizations, particularly where significant or hard to penetrate digital defenses are deployed (Kim, 2015). For example, a common tactic among the most determined attackers is to use a "rubber ducky," which is a slang term for a weaponized USB drive loaded with malware but externally labeled as something potentially enticing to a passerby who may pick it up. When the victim collects it and inserts it into her computer to inspect the contents, the malware is delivered and the attacker is provided a remote foothold in the target system. In practice, attackers may purchase many cheap USB drives, clone malware onto each, then give each an interesting label such as "Holiday bonus list," "Employee layoffs list," or "Webcam screenshots 2013-2017." The attacker will surreptitiously scatter these rubber duckies in the parking lot of the target organization, hoping that just one employee's curiosity will get the better of her and lead to network compromise (Kim, 2015).

There exist other strategies for physical network compromise as well. Attackers could gain physical access to an organization posing as an employee, planting rogue devices or secretly copying access codes left in plain sight (Mitnick & Simon, 2011). Alternatively, attackers could exploit hybrid techniques where a user's physical device is compromised at home and, when taken to a workplace, compromises the employer network and establishes a foothold. In

these cases, user mental models associated with physical network compromise may be much different from those of digital compromise explored with our method.[3] Future work should therefore explore how mental models of computer security vulnerabilities extend from digital to physical exploit strategies, the nature of the aforementioned mismatches with respect to physical attack vectors, and the impact of these mismatches on existing computer security and vulnerability mitigation protocols.

6.4.2.2 Extending the Method to Specific Software Utilities

We explored both the generic and extended formulations of our method through two use cases: the first investigated unanticipated security configuration errors with the AWS S3 cloud data storage service, and the second investigated phishing attacks within a generic email program. We believe that our method should be extensible to other software utilities that offer other storage and control configurations.

Both of these extensions could yield insight regarding unanticipated cybersecurity vulnerabilities within those systems. For example, it may be possible to adapt the generic method to investigate data storage protocols within Dropbox, Google Drive, Microsft Azure Storage, Google Cloud, Alibaba Cloud, Oracle Cloud, and Box, which has an enterprise affiliation with The University at Buffalo. These services are available for public or corporate use and can be used to store a practically unlimited variety of files. Our method could be very helpful in determining whether users risk the same potential vulnerabilities as those who use AWS S3 for data storage.

Similarly, our extended method could be adapted for use with specific email clients and their security configuration protocols. Consider the differences between large closed-source email clients, such as Gmail's web mail interface, Microsoft Outlook, and Apple Mail, and smaller open-source clients such as Thunderbird, Evolution, Kmail, Mutt, and others. Not only could there be significant differences in how each client handles default security settings, potential configuration conditions, and spam or malware email filtering, but there could also be

[3] These differences could surface for a number of reasons, including a lack of cybersecurity training that focuses on these less common physical attack techniques.

significant differences between the closed- and open-source clients by virtue of the customizability of open-source software. Because the developers of each client may have different user control or system security best practices in mind, our method could help provide each email client with a detailed "human security audit" to investigate whether their default settings, user configuration options, or integrated security protocols could result in unanticipated cybersecurity vulnerabilities when their users are confronted with potential phishing emails.

6.4.2.3 Integrating Mental Modeling Concepts into Cybersecurity Testbeds

One of our method's most important contributions is that it offers model-based formal methods techniques for rapidly and efficiently exploring user responses to cybersecurity events. We demonstrated in Chapters 4 and 5, as well as discussed above, strategies by which our formulation could be adapted to a variety of problem spaces and analytic challenges. We believe that the flexibility of our approach means that it could be used to generate novel training scenario components on-the-fly in a repeatable, iterative, and verifiable manner. For example, an analyst could provide the method a target system model and a variety of scenarios; generate a variety of user responses to cybersecurity events with respect to the target system and scenarios; then translate the method's findings into user configuration decisions or actions so as to train network defenders under realistic conditions. The method could also be used to explore how slight system changes impact these simulated user responses under otherwise identical conditions, helping defenders understand the specific implications of changes made to secure a network. We believe these capabilities are well suited for experimental design and environmental simulation efforts.

Future work should explore the integration of our method and its capabilities into cybersecurity testbed platforms. Consider the use of simulation environments known as "cyber test ranges" (NIST, 2018). These environments are used to test and train network security professionals against vulnerabilities, scenarios, and attack and defense strategies too sensitive to conduct on the internet (Northrop Grumman, n.d.). Both private- and public-sector organizations have begun to establish these ranges, including Northrop Grumman, Raytheon

(Raytheon, n.d.), Palo Alto Networks (Palo Alto Networks, n.d.), individual state-level organizations (Merit Network, 2018; *Virginia Cyber Range*, 2018), and the Department of Defense and US Cyber Command (Ferguson, Tall, & Olsen, 2014). In each case, training network defenders to respond to cybersecurity events should focus on system-level user event responses. Integrating realistic sources of erroneous human actions into these simulation environments could improve the rigor with which training exercises are performed. Our method could offer a novel strategy for incorporating these conditions into cyber range environments.

6.4.2.4 Meaningful Insight Regarding Insider Threats

A final target for future work is the refinement of our method's capability to differentiate between deliberate and accidental insider threats. The term "insider threat" generally describes current or former employees, contractors, or business partners that have or had legitimate access credentials and use them to access or steal information for financial profit, sabotage, espionage, or fraud (*Combating the Insider Threat*, 2014). However, there exists a need to make reliable distinctions between deliberately malicious insider threats (those that engage in information theft for nefarious purposes) and inadvertent threats that also compromise sensitive information but without malicious intent. These accidental insider threats can lose sensitive documents or devices, accidentally leave sensitive information publicly exposed online, click "Reply All" rather than "Reply" and copy a sensitive email chain, or perform other actions that result in data disclosure (CMU SEI, 2013). Furthermore, major private sector cybersecurity research and defense firms, as well as federal cybersecurity response teams for the United States, the United Kingdom, the European Union, and others often identify insider threats as a major source of concern but can fail to differentiate deliberate from accidental threats (IBM, 2015; Verizon Enterprise, 2016; SolarWinds, 2016; *National Cyber Security Strategy 2016-2021*, 2016; Samarati, 2016). While distinctions between deliberate and accidental disclosure of information may be blurred, they can mean the difference between internal disciplinary action and felony criminal charges for those who stand accused.

One potential reason for this blurring is that it can be difficult to reliably determine the

deliberate from accidental insider threats, particularly after the breach has occurred and a perpetrator could simply claim ignorance rather than revealing their original nefarious intent. Because our analysis offers counterexamples that capture detailed, explicit traces of user actions and system states that led to the vulnerability that occurred, it could be possible to extract meaningful information from the counterexample trace to determine whether the user's actions were erroneous but innocent or belie more malicious intent. This could offer a powerful technique for exploring insider threat activities and monitoring systems for those who may be intentionally mishandling their access privileges for gain. Future work could help establish patterns of malicious or benign access to sensitive files, system settings configuration, USB or other writeable media usage, web browsing, or other vectors for deliberate espionage and information exfiltration.

7 | Conclusion

> "The work ahead is difficult and without end. Who among you feels daunted?" No one raised a hand. No one made a sound.
>
> "The work ahead will tire us and will frustrate us, and victories will be brief and quickly reversed. Who among you is unwilling?" No one raised a hand. No one made a sound.
>
> *Dave Eggers*
> *The Lifters*

This work has investigated the feasibility of discovering unanticipated cybersecurity risks by identifying, formalizing, and exploring the interactions between user mental models and various computer utilities using formal methods techniques. We have created a framework that extends established analytic concepts from human-automation interaction to the cybersecurity domain, using these techniques to discover and rigorously characterize dangerous user-system interactions within the context of a real-world cloud data storage system. We have also expanded our framework to incorporate folk modeling concepts into the method and investigated the potential hazards faced by the human targets of phishing emails. In both cases, our findings demonstrate that the method is capable of discovering unanticipated cybersecurity vulnerabilities introduced through user actions informed by a mental model of system functionality that is misaligned with the actual state of the world.

Our work has also yielded a significant number of theoretical developments. In addition to the extension of Degani and Heymann (2002)'s *error* and *blocking* state architecture to the cybersecurity domain, we have created several formalisms and specifications that facilitate the investigation of cybersecurity concepts using formal methods techniques. Our work has also yielded a method to bring runnability to folk models, opening the doors to further exploration of folk models through both simulation and formal methods techniques. Establishing the framework necessary for runnable folk models simultaneously improves the rigor of folk modeling constructs and could be used to glean insight from a variety of folk models already published in other fields, such as public health (Greenhalgh et al., 1998; Kubicek et al., 2008) and education (Torff, 1999). Taken together, these developments represent substantial and novel contributions to human factors, formal methods, cybersecurity, and other disciplines.

Finally, we have offered a number of potential improvements to our method and targets for future work. For example, additional efforts could be made to improve the rigor of our method by refining the folk modeling concepts of Wash (2010), as well as implementing more detailed system representations and data handling capabilities. It may also be possible to extend the method to other computer programs and utilities where unanticipated user interactions could result in dangerous cybersecurity conditions, offering a potentially limitless source of use cases and avenues for further refinement. Our method may also offer commercial development opportunities through the integration of our framework into cybersecurity testbeds, cybersecurity assessment and training programs, and other applications. Future would should focus on more fully developing the rigor, capabilities, and potential of our method.

Appendices

A | AWS Formal Model Code

```
awsModel : CONTEXT =
 BEGIN
 userAction:          TYPE = {create_Object, create_Bucket,
    ↪ delete_Object, delete_Bucket, change_BucketPermissions,
    ↪ change_ObjectPermissions, change_BucketEncryption,
    ↪ change_ObjectEncryption};
 encryptionSetting:   TYPE = {none_Option, manual_Option};
 permissionSetting:   TYPE = [# public: BOOLEAN, read : BOOLEAN, write
    ↪ : BOOLEAN, readPermissions : BOOLEAN, writePermissions :
    ↪ BOOLEAN #];

 userState:           TYPE = {safe, vulnerable};
 softwareState:       TYPE = {secure, insecure};
 private:             permissionSetting = (# public := FALSE, read :=
    ↪ FALSE, write := FALSE, readPermissions := FALSE,
    ↪ writePermissions := FALSE #);

 % Functions
 isPublic?(perm : permissionSetting) : BOOLEAN = perm.public AND (perm
    ↪ .read OR perm.write OR perm.readPermissions OR perm.
    ↪ writePermissions);
 isPrivate?(perm : permissionSetting) : BOOLEAN = NOT perm.read AND
    ↪ NOT perm.write AND NOT perm.readPermissions AND NOT perm.
    ↪ writePermissions;
 isAuthenticatedWrite?(perm : permissionSetting) : BOOLEAN = ((NOT
    ↪ perm.public) AND (perm.write OR perm.writePermissions));
 isAuthenticatedRead?(perm : permissionSetting) : BOOLEAN = ((NOT perm
    ↪ .public) AND (perm.read OR perm.readPermissions));
```

```
isAuthenticated?(perm : permissionSetting) : BOOLEAN =
    ↪ isAuthenticatedWrite?(perm) OR isAuthenticatedRead?(perm);
isPublicWritePermissions?(perm : permissionSetting) : BOOLEAN = (perm
    ↪ .public AND perm.writePermissions);


actionGenerator: MODULE =
BEGIN
  OUTPUT action:      userAction
  OUTPUT actionPerm:  permissionSetting
  OUTPUT actionEnc:   encryptionSetting

  DEFINITION
    action      IN  {x: userAction | TRUE};
    actionPerm  IN  IF action = change_ObjectPermissions OR action =
       ↪ create_Bucket OR action = change_BucketPermissions
                THEN {x : permissionSetting | NOT (x.public AND
                   ↪ isPrivate?(x))}
                ELSE {x : permissionSetting | (NOT x.public) AND
                   ↪ isPrivate?(x)} ENDIF;
    actionEnc   IN  IF   action = create_Object
                    OR action = create_Bucket
                    OR action = change_BucketEncryption
                    OR action = change_ObjectEncryption
                THEN {none_Option, manual_Option}
                ELSE {none_Option} ENDIF;
END;

aws: MODULE =
BEGIN
  %-----------------------------------------------
  % AWS Resources
  %-----------------------------------------------
  LOCAL bucketExists:       BOOLEAN
  LOCAL bucketEncryption:   encryptionSetting
  LOCAL bucketACL:          permissionSetting

  LOCAL objectExists:       BOOLEAN
  LOCAL objectACL:          permissionSetting
  LOCAL objectEncryption:   encryptionSetting
```

```
INPUT action:              userAction
INPUT actionPerm:          permissionSetting
INPUT actionEnc:           encryptionSetting

OUTPUT  softwareWrite:     softwareState
OUTPUT  softwareRead:      softwareState

INITIALIZATION
  bucketExists =           FALSE;
  bucketEncryption =       none_Option;
  bucketACL =              private;

  objectExists =           FALSE;
  objectEncryption =       none_Option;
  objectACL =              private;

DEFINITION
  softwareWrite = if objectExists AND objectACL.public AND (
     ↪ objectACL.write OR objectACL.writePermissions) THEN
     ↪ insecure ELSE secure ENDIF;
  softwareRead = if objectExists AND objectACL.public AND (
     ↪ objectACL.read OR objectACL.readPermissions OR objectACL.
     ↪ writePermissions) THEN insecure ELSE secure ENDIF;

TRANSITION
[
  %object transitions
  action = create_Object AND bucketExists AND NOT objectExists -->
    objectExists' =        TRUE;
    objectEncryption' =    IF actionEnc = none_Option then
       ↪ bucketEncryption ELSE actionEnc ENDIF;
    objectACL' =           actionPerm;
  [] action = delete_Object AND objectExists -->
    objectExists' =        FALSE;
    objectEncryption' =    none_Option;
    objectACL' =           private;
  [] action = change_ObjectPermissions AND objectExists -->
    objectACL' =           actionPerm;
  [] action = change_ObjectEncryption AND objectExists -->
```

```
      objectEncryption' =     actionEnc;

    %bucket transitions
    [] action = create_Bucket AND NOT bucketExists -->
      bucketExists' =         TRUE;
      bucketEncryption' =     actionEnc;
      bucketACL' =            actionPerm;
    [] action = delete_Bucket AND bucketExists -->
      bucketExists' =         FALSE;
      bucketEncryption' =     none_Option;
      bucketACL' =            private;
      objectExists' =         FALSE;
      objectEncryption' =     none_Option;
      objectACL' =            private;
    [] action = change_BucketPermissions AND bucketExists -->
      bucketACL' =            actionPerm;
    [] action = change_BucketEncryption AND bucketExists -->
      bucketEncryption' =     actionEnc;

    %no change
    [] ELSE -->
      objectExists' =         objectExists;
  ]
END;

mental: MODULE =
BEGIN
  LOCAL thinkBucketExists:      BOOLEAN
  LOCAL thinkBucketEncryption:  encryptionSetting
  LOCAL thinkBucketACL:         permissionSetting

  LOCAL thinkObjectExists:      BOOLEAN
  LOCAL thinkObjectACL:         permissionSetting
  LOCAL thinkObjectEncryption:  encryptionSetting

  INPUT action:                 userAction
  INPUT actionPerm:             permissionSetting
  INPUT actionEnc:              encryptionSetting

  OUTPUT  userWrite:            userState
```

```
OUTPUT  userRead:              userState

INITIALIZATION
  thinkBucketExists =          FALSE;
  thinkObjectExists =          FALSE;
  thinkBucketEncryption =      none_Option;
  thinkObjectEncryption =      none_Option;




DEFINITION
  userWrite = if thinkObjectExists AND thinkObjectACL.public AND (
     ↪ thinkObjectACL.write OR thinkObjectACL.writePermissions)
     ↪ AND thinkObjectEncryption = none_Option THEN vulnerable
     ↪ ELSE safe ENDIF;
  userRead = if thinkObjectExists AND thinkObjectACL.public AND (
     ↪ thinkObjectACL.read OR thinkObjectACL.readPermissions) AND
     ↪ thinkObjectEncryption = none_Option THEN vulnerable ELSE
     ↪ safe ENDIF;

TRANSITION
[
  %object transitions
  action = create_Object AND thinkBucketExists AND NOT
     ↪ thinkObjectExists -->
    thinkObjectExists' =       TRUE;
    thinkObjectEncryption' =   actionEnc;
    thinkObjectACL' =          actionPerm;
  [] action = delete_Object AND thinkObjectExists -->
    thinkObjectExists' =       FALSE;
    thinkObjectEncryption' =   none_Option;
  [] action = change_ObjectPermissions AND thinkObjectExists -->
    thinkObjectACL' =          actionPerm;
  [] action = change_ObjectEncryption AND thinkObjectExists -->
    thinkObjectEncryption' =   actionEnc;

  %bucket transitions
  [] action = create_Bucket AND NOT thinkBucketExists -->
    thinkBucketExists' =       TRUE;
```

```
      thinkBucketEncryption' =  actionEnc;
      thinkBucketACL' =         actionPerm;
    [] action = delete_Bucket AND thinkBucketExists -->
      thinkBucketExists' =      FALSE;
      thinkBucketEncryption' = none_Option;
      thinkObjectExists' =      FALSE;
      thinkObjectEncryption' =  none_Option;
    [] action = change_BucketPermissions AND thinkBucketExists -->
      thinkBucketACL' =         actionPerm;
      thinkObjectACL' =         actionPerm;
    [] action = change_BucketEncryption AND thinkBucketExists -->
      thinkBucketEncryption' =  actionEnc;
      thinkObjectEncryption' =  IF thinkObjectExists = FALSE THEN
        ↪ none_Option ELSE actionEnc ENDIF;

    %no change
    [] ELSE -->
      thinkObjectExists' = thinkObjectExists;
  ]
END;

comparator: MODULE =
BEGIN
  INPUT   userWrite:          userState
  INPUT   userRead:           userState
  INPUT   softwareWrite:      softwareState
  INPUT   softwareRead:       softwareState

  OUTPUT  safetyWriteMM:      BOOLEAN
  OUTPUT  blockingWriteMM:    BOOLEAN
  OUTPUT  safetyReadMM:       BOOLEAN
  OUTPUT  blockingReadMM:     BOOLEAN

  DEFINITION
    safetyWriteMM = (userWrite = safe AND softwareWrite = insecure);
    blockingWriteMM = (userWrite = vulnerable AND softwareWrite =
      ↪ secure);
    safetyReadMM = (userRead = safe AND softwareRead = insecure);
    blockingReadMM = (userRead = vulnerable AND softwareRead = secure
      ↪ );
```

```
END;

theModel: MODULE = aws || mental || comparator || actionGenerator;
% ------------------------------------------
% Safety and blocking specifications
% ------------------------------------------
noSafetyWriteMM:          THEOREM theModel |- G(NOT safetyWriteMM);
noBlockingWriteMM:        THEOREM theModel |- G(NOT blockingWriteMM);
noRecoveryWriteBlocking:  THEOREM theModel |- G(NOT((softwareWrite =
   ↪ insecure AND userWrite = vulnerable) AND X (blockingWriteMM)));

noSafetyReadMM:           THEOREM theModel |- G(NOT safetyReadMM);
noBlockingReadMM:         THEOREM theModel |- G(NOT blockingReadMM);
noRecoveryReadBlocking:   THEOREM theModel |- G(NOT((softwareRead =
   ↪ insecure AND userRead = vulnerable) AND X (blockingReadMM)));
% ------------------------------------------
% Other properties to check
% ------------------------------------------
% the user thinks their object is private but it's still publicly
   ↪ readable

userThinksObjectPrivate:    THEOREM theModel |- G(NOT((isPublic?(
   ↪ thinkBucketACL) AND isPublic?(thinkObjectACL)) AND X(isPrivate
   ↪ ?(thinkBucketACL) AND isPrivate?(thinkObjectACL) AND (objectACL
   ↪ .read OR objectACL.write OR objectACL.readPermissions OR
   ↪ objectACL.writePermissions))));
% the user thinks the object is encrypted, but it can still be
   ↪ publicly read

encryptedStillReadSimple:   THEOREM theModel |- G(NOT(
   ↪ objectEncryption /= none_Option AND isPrivate?(thinkObjectACL)
   ↪ AND isPublic?(objectACL)));
encryptedStillReadComplex:  THEOREM theModel |- G(NOT(
   ↪ thinkObjectEncryption = manual_Option AND objectEncryption =
   ↪ manual_Option AND X(objectEncryption = manual_Option AND
   ↪ objectACL.read)));

% it should never be the case that if a bucket and object exist
   ↪ unencrypted, then encrypting the bucket will not encrypt the
   ↪ object
```

```
encryptionTricklesDown:     THEOREM theModel |- G(NOT(bucketExists =
    ↪ TRUE AND bucketEncryption = none_Option AND objectExists = TRUE
    ↪  AND objectEncryption = none_Option AND action =
    ↪ change_BucketEncryption AND X(bucketEncryption = manual_Option
    ↪ AND objectEncryption = none_Option)));
END
```

B | Folk Model Formal Model Code

```
folkModel : CONTEXT =
  BEGIN
  userState:       TYPE = {safe,vulnerable,ambiguous};
  userFolkModel:   TYPE = {graffiti,burglar,bigFish,contractor};
  softwareState:   TYPE = {secure,insecure};
  blocker:         TYPE = {enabled,disabled};
  content:         TYPE = {basic,personal};
  action:          TYPE = {websiteLink,websiteDownload,hostileAttachment
     ↪ ,toggleAdBlock,patchSoftware,noAction};

  system: MODULE =
  BEGIN
    %-----------------------------------------------
    % System (Backend) Properties
    %-----------------------------------------------
      LOCAL adBlocker:         blocker
      LOCAL softwarePatched:   BOOLEAN
      OUTPUT sensitiveInfo:    BOOLEAN
    %-----------------------------------------------
    % Incoming/Outgoing Actions or Events
    %-----------------------------------------------
      INPUT actionType:      action
      OUTPUT software:       softwareState

    INITIALIZATION
      software =             secure;
```

```
  TRANSITION
  [
    actionType = websiteLink -->
      software' =   IF softwarePatched = TRUE AND adBlocker = enabled
                    THEN software
                    ELSE insecure
                    ENDIF;
    [] actionType = websiteDownload -->
      software' =   IF softwarePatched = TRUE THEN software
                    ELSE insecure
                    ENDIF;
    [] actionType = hostileAttachment -->
      software' =   IF softwarePatched = TRUE THEN software
                    ELSE insecure
                    ENDIF;
    [] actionType = toggleAdBlock -->
      adBlocker' =  IF adBlocker = enabled THEN disabled ELSE enabled
        ↪  ENDIF;
    [] actionType = patchSoftware -->
      software' IN  IF software = secure
                    THEN {secure}
                    ELSE {secure, insecure}
                    ENDIF;
    [] ELSE -->
      software' =   software;
  ]
END;

user: MODULE =
BEGIN
  %--------------------------------------------
  % Userspace (Frontend) System Properties
  %--------------------------------------------
    LOCAL model:          userFolkModel
  %--------------------------------------------
  % Incoming/Outgoing Actions or Events
  %--------------------------------------------
    INPUT actionType:     action
```

```
    INPUT software:        softwareState
    INPUT sensitiveInfo:   BOOLEAN
    LOCAL siteContent:     content
    OUTPUT user:           userState

   INITIALIZATION
    user =                 safe;

  DEFINITION
    siteContent = IF sensitiveInfo = FALSE
                          THEN basic
                          ELSE personal
                          ENDIF;

  TRANSITION
  % ratings: 3 = very important, 2 = somewhat important, 1 = not
     ↪ important, emptyset = not enough data to comment one way or
     ↪ another (see Wash, 2010)
  [
    % rating = 3
    model = graffiti AND actionType = websiteLink -->
      user' =   vulnerable;
    % rating = 2
    [] model = graffiti AND actionType = websiteDownload -->
      user' =   IF siteContent = basic AND software = secure THEN
         ↪ user ELSE vulnerable ENDIF;
    % rating = 3
    [] model = graffiti AND actionType = hostileAttachment -->
      user' =   vulnerable;
    % rating = 3
    [] model = burglar AND actionType = websiteLink -->
      user' =   vulnerable;
    % rating = 2
    [] model = burglar AND actionType = websiteDownload -->
      user' =   IF siteContent = basic AND software = secure THEN
         ↪ user ELSE vulnerable ENDIF;
    % rating = 3
    [] model = burglar AND actionType = hostileAttachment -->
      user' =   vulnerable;
    % rating = 2
```

```
      [] model = bigFish AND actionType = websiteLink -->
        user' =   IF siteContent = basic AND software = secure THEN
            ↪ user ELSE vulnerable ENDIF;
      % rating = 1
      [] model = bigFish AND actionType = websiteDownload -->
        user' =   user;
      % rating = emptyset
      [] model = bigFish AND actionType = hostileAttachment -->
        user' =   ambiguous;
      % rating = 3
      [] model = contractor AND actionType = websiteLink -->
        user' =   vulnerable;
      % rating = 1
      [] model = contractor AND actionType = websiteDownload -->
        user' =   user;
      % rating = emptyset
      [] model = contractor AND actionType = hostileAttachment -->
        user' =   ambiguous;
      [] actionType = patchSoftware -->
        user' =   safe;
      [] actionType = toggleAdBlock -->
        user' =   safe;
      [] ELSE -->
        user' =   user;
    ]
  END;

  comparator: MODULE =
  BEGIN
    INPUT user:           userState
    INPUT software:       softwareState
    OUTPUT mismatch:      BOOLEAN
    OUTPUT safetyMM:      BOOLEAN
    OUTPUT blockingMM:    BOOLEAN
    OUTPUT safetyAmb:     BOOLEAN
    OUTPUT blockingAmb:   BOOLEAN

    DEFINITION
      safetyMM =          (user = safe AND software = insecure);
      blockingMM =        (user = vulnerable AND software = secure);
```

```
    mismatch =          (safetyMM OR blockingMM);
    safetyAmb =         (user = ambiguous AND software = insecure);
    blockingAmb =       (user = ambiguous AND software = secure);
END;

theModel: MODULE = system || user || comparator;


% ---------------------------------
% Graffiti model specifications
% ---------------------------------
GraffitiFalseSafetyLinkSafety:          THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = websiteLink AND NOT safetyMM)
   ↪  AND X(safetyMM)));
GraffitiFalseSafetyDownloadSafety:      THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = websiteDownload AND NOT
   ↪ safetyMM) AND X(safetyMM)));
GraffitiFalseSafetyAttachmentSafety:    THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = hostileAttachment AND NOT
   ↪ safetyMM) AND X(safetyMM)));
GraffitiFalseSafetyNoActionSafety:      THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = noAction) AND NOT safetyMM
   ↪ AND X(safetyMM)));

GraffitiFalseSafetyLinkAmb:             THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = websiteLink AND NOT safetyAmb
   ↪ ) AND X(safetyAmb)));
GraffitiFalseSafetyDownloadAmb:         THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = websiteDownload AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
GraffitiFalseSafetyAttachmentAmb:       THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = hostileAttachment AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
GraffitiFalseSafetyNoActionAmb:         THEOREM theModel |- G(NOT((
   ↪ model = graffiti AND actionType = noAction AND NOT safetyAmb)
   ↪ AND X(safetyAmb)));

GraffitiFalseVulnerabilityLinkSafety:       THEOREM theModel |- G(NOT
   ↪ ((model = graffiti AND actionType = websiteLink AND NOT
   ↪ blockingMM) AND X(blockingMM)));
```

```
GraffitiFalseVulnerabilityDownloadSafety:   THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = websiteDownload AND NOT
    ↪ blockingMM) AND X(blockingMM)));
GraffitiFalseVulnerabilityAttachmentSafety: THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = hostileAttachment AND NOT
    ↪ blockingMM) AND X(blockingMM)));
GraffitiFalseVulnerabilityNoActionSafety:   THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = noAction AND NOT blockingMM
    ↪ ) AND X(blockingMM)));

GraffitiFalseVulnerabilityLinkAmb:          THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = websiteLink AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
GraffitiFalseVulnerabilityDownloadAmb:      THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = websiteDownload AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
GraffitiFalseVulnerabilityAttachmentAmb:    THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = hostileAttachment AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
GraffitiFalseVulnerabilityNoActionAmb:      THEOREM theModel |- G(NOT
    ↪ ((model = graffiti AND actionType = noAction AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));

GraffitiLinkNRB:          THEOREM theModel |- G((model = graffiti AND
    ↪  actionType = websiteLink) => NOT((software = insecure AND user
    ↪  = vulnerable) AND X(blockingMM)));
GraffitiDownloadNRB:      THEOREM theModel |- G((model = graffiti AND
    ↪  actionType = websiteDownload) => NOT((software = insecure AND
    ↪ user = vulnerable) AND X(blockingMM)));
GraffitiAttachmentNRB:    THEOREM theModel |- G((model = graffiti AND
    ↪  actionType = hostileAttachment) => NOT((software = insecure
    ↪ AND user = vulnerable) AND X(blockingMM)));
GraffitiNoActionNRB:      THEOREM theModel |- G((model = graffiti AND
    ↪  actionType = noAction) => NOT((software = insecure AND user =
    ↪ vulnerable) AND X(blockingMM)));

% ---------------------------------
% Burglar specifications
% ---------------------------------
```

```
BurglarFalseSafetyLinkSafety:           THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = websiteLink AND NOT safetyMM)
   ↪ AND X(safetyMM)));
BurglarFalseSafetyDownloadSafety:       THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = websiteDownload AND NOT
   ↪ safetyMM) AND X(safetyMM)));
BurglarFalseSafetyAttachmentSafety:     THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = hostileAttachment AND NOT
   ↪ safetyMM) AND X(safetyMM)));
BurglarFalseSafetyNoActionSafety:       THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = noAction AND NOT safetyMM) AND
   ↪  X(safetyMM)));

BurglarFalseSafetyLinkAmb:              THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = websiteLink AND NOT safetyAmb)
   ↪  AND X(safetyAmb)));
BurglarFalseSafetyDownloadAmb:          THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = websiteDownload AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
BurglarFalseSafetyAttachmentAmb:        THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = hostileAttachment AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
BurglarFalseSafetyNoActionAmb:          THEOREM theModel |- G(NOT((
   ↪ model = burglar AND actionType = noAction AND NOT safetyAmb)
   ↪ AND X(safetyAmb)));

BurglarFalseVulnerabilityLinkSafety:        THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = websiteLink AND NOT
   ↪ blockingMM) AND X(blockingMM)));
BurglarFalseVulnerabilityDownloadSafety:    THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = websiteDownload AND NOT
   ↪ blockingMM) AND X(blockingMM)));
BurglarFalseVulnerabilityAttachmentSafety:  THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = hostileAttachment AND NOT
   ↪ blockingMM) AND X(blockingMM)));
BurglarFalseVulnerabilityNoActionSafety:    THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = noAction AND NOT blockingMM)
   ↪  AND X(blockingMM)));
```

```
BurglarFalseVulnerabilityLinkAmb:           THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = websiteLink AND NOT
   ↪ blockingAmb) AND X(blockingAmb)));
BurglarFalseVulnerabilityDownloadAmb:       THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = websiteDownload AND NOT
   ↪ blockingAmb) AND X(blockingAmb)));
BurglarFalseVulnerabilityAttachmentAmb:     THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = hostileAttachment AND NOT
   ↪ blockingAmb) AND X(blockingAmb)));
BurglarFalseVulnerabilityNoActionAmb:       THEOREM theModel |- G(NOT
   ↪ ((model = burglar AND actionType = noAction AND NOT blockingAmb
   ↪ ) AND X(blockingAmb)));

BurglarLinkNRB:           THEOREM theModel |- G((model = burglar AND
   ↪ actionType = websiteLink) => NOT((software = insecure AND user
   ↪ = vulnerable) AND X(blockingMM)));
BurglarDownloadNRB:       THEOREM theModel |- G((model = burglar AND
   ↪ actionType = websiteDownload) => NOT((software = insecure AND
   ↪ user = vulnerable) AND X(blockingMM)));
BurglarAttachmentNRB:     THEOREM theModel |- G((model = burglar AND
   ↪ actionType = hostileAttachment) => NOT((software = insecure AND
   ↪  user = vulnerable) AND X(blockingMM)));
BurglarNoActionNRB:       THEOREM theModel |- G((model = burglar AND
   ↪ actionType = noAction) => NOT((software = insecure AND user =
   ↪ vulnerable) AND X(blockingMM)));

% --------------------------------
% Big Fish specifications
% --------------------------------
BigFishFalseSafetyLinkSafety:           THEOREM theModel |- G(NOT((
   ↪ model = bigFish AND actionType = websiteLink AND NOT safetyMM)
   ↪ AND X(safetyMM)));
BigFishFalseSafetyDownloadSafety:       THEOREM theModel |- G(NOT((
   ↪ model = bigFish AND actionType = websiteDownload AND NOT
   ↪ safetyMM) AND X(safetyMM)));
BigFishFalseSafetyAttachmentSafety:     THEOREM theModel |- G(NOT((
   ↪ model = bigFish AND actionType = hostileAttachment AND NOT
   ↪ safetyMM) AND X(safetyMM)));
BigFishFalseSafetyNoActionSafety:       THEOREM theModel |- G(NOT((
   ↪ model = bigFish AND actionType = noAction AND NOT safetyMM) AND
```

```
    ↪  X(safetyMM)));

BigFishFalseSafetyLinkAmb:              THEOREM theModel |- G(NOT((
    ↪ model = bigFish AND actionType = websiteLink AND NOT safetyAmb)
    ↪  AND X(safetyAmb)));
BigFishFalseSafetyDownloadAmb:          THEOREM theModel |- G(NOT((
    ↪ model = bigFish AND actionType = websiteDownload AND NOT
    ↪ safetyAmb) AND X(safetyAmb)));
BigFishFalseSafetyAttachmentAmb:        THEOREM theModel |- G(NOT((
    ↪ model = bigFish AND actionType = hostileAttachment AND NOT
    ↪ safetyAmb) AND X(safetyAmb)));
BigFishFalseSafetyNoActionAmb:          THEOREM theModel |- G(NOT((
    ↪ model = bigFish AND actionType = noAction AND NOT safetyAmb)
    ↪ AND X(safetyAmb)));

BigFishFalseVulnerabilityLinkSafety:    THEOREM theModel |- G(NOT((
    ↪ model = bigFish AND actionType = websiteLink AND NOT blockingMM
    ↪ ) AND X(blockingMM)));
BigFishFalseVulnerabilityDownloadSafety:    THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = websiteDownload AND NOT
    ↪ blockingMM) AND X(blockingMM)));
BigFishFalseVulnerabilityAttachmentSafety:  THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = hostileAttachment AND NOT
    ↪ blockingMM) AND X(blockingMM)));
BigFishFalseVulnerabilityNoActionSafety:    THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = noAction AND NOT blockingMM)
    ↪  AND X(blockingMM)));

BigFishFalseVulnerabilityLinkAmb:           THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = websiteLink AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
BigFishFalseVulnerabilityDownloadAmb:       THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = websiteDownload AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
BigFishFalseVulnerabilityAttachmentAmb:     THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = hostileAttachment AND NOT
    ↪ blockingAmb) AND X(blockingAmb)));
BigFishFalseVulnerabilityNoActionAmb:       THEOREM theModel |- G(NOT
    ↪ ((model = bigFish AND actionType = noAction AND NOT blockingAmb
    ↪ ) AND X(blockingAmb)));
```

```
BigFishLinkNRB:           THEOREM theModel |- G((model = bigFish AND
   ↪ actionType = websiteLink) => NOT((software = insecure AND user
   ↪ = vulnerable) AND X(blockingMM)));
BigFishDownloadNRB:       THEOREM theModel |- G((model = bigFish AND
   ↪ actionType = websiteDownload) => NOT((software = insecure AND
   ↪ user = vulnerable) AND X(blockingMM)));
BigFishAttachmentNRB:     THEOREM theModel |- G((model = bigFish AND
   ↪ actionType = hostileAttachment) => NOT((software = insecure AND
   ↪  user = vulnerable) AND X(blockingMM)));
BigFishNoActionNRB:       THEOREM theModel |- G((model = bigFish AND
   ↪ actionType = noAction) => NOT((software = insecure AND user =
   ↪ vulnerable) AND X(blockingMM)));

% --------------------------------
% Contractor specifications
% --------------------------------
ContractorFalseSafetyLinkSafety:          THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = websiteLink AND NOT
   ↪ safetyMM) AND X(safetyMM)));
ContractorFalseSafetyDownloadSafety:      THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = websiteDownload AND NOT
   ↪ safetyMM) AND X(safetyMM)));
ContractorFalseSafetyAttachmentSafety:    THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = hostileAttachment AND NOT
   ↪ safetyMM) AND X(safetyMM)));
ContractorFalseSafetyNoActionSafety:      THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = noAction AND NOT safetyMM)
   ↪ AND X(safetyMM)));

ContractorFalseSafetyLinkAmb:             THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = websiteLink AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
ContractorFalseSafetyDownloadAmb:         THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = websiteDownload AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
ContractorFalseSafetyAttachmentAmb:       THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = hostileAttachment AND NOT
   ↪ safetyAmb) AND X(safetyAmb)));
```

```
ContractorFalseSafetyNoActionAmb:          THEOREM theModel |- G(NOT((
   ↪ model = contractor AND actionType = noAction AND NOT safetyAmb)
   ↪  AND X(safetyAmb)));

ContractorFalseVulnerabilityLinkSafety:       THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = websiteLink AND NOT
   ↪ blockingMM) AND X(blockingMM)));
ContractorFalseVulnerabilityDownloadSafety:   THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = websiteDownload AND
   ↪ NOT blockingMM) AND X(blockingMM)));
ContractorFalseVulnerabilityAttachmentSafety: THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = hostileAttachment AND
   ↪ NOT blockingMM) AND X(blockingMM)));
ContractorFalseVulnerabilityNoActionSafety:   THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = noAction AND NOT
   ↪ blockingMM) AND X(blockingMM)));

ContractorFalseVulnerabilityLinkAmb:          THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = websiteLink AND NOT
   ↪ blockingAmb) AND X(blockingAmb)));
ContractorFalseVulnerabilityDownloadAmb:      THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = websiteDownload AND
   ↪ NOT blockingAmb) AND X(blockingAmb)));
ContractorFalseVulnerabilityAttachmentAmb:    THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = hostileAttachment AND
   ↪ NOT blockingAmb) AND X(blockingAmb)));
ContractorFalseVulnerabilityNoActionAmb:      THEOREM theModel |- G(
   ↪ NOT((model = contractor AND actionType = noAction AND NOT
   ↪ blockingAmb) AND X(blockingAmb)));

ContractorLinkNRB:          THEOREM theModel |- G((model = contractor
   ↪  AND actionType = websiteLink) => NOT((software = insecure AND
   ↪ user = vulnerable) AND X(blockingMM)));
ContractorDownloadNRB:      THEOREM theModel |- G((model = contractor
   ↪  AND actionType = websiteDownload) => NOT((software = insecure
   ↪ AND user = vulnerable) AND X(blockingMM)));
ContractorAttachmentNRB:    THEOREM theModel |- G((model = contractor
   ↪  AND actionType = hostileAttachment) => NOT((software =
   ↪ insecure AND user = vulnerable) AND X(blockingMM)));
```

```
  ContractorNoActionNRB:      THEOREM theModel |- G((model = contractor
     ↪  AND actionType = noAction) => NOT((software = insecure AND
     ↪ user = vulnerable) AND X(blockingMM)));
END
```

Bibliography

18 USC § 1030. (2008). *United States Government, Provisions of the Computer Fraud and Abuse Act*. Retrieved from https://www.law.cornell.edu/uscode/text/18/1030

Aljazzar, H., & Leue, S. (2008). Debugging of dependability models using interactive visualization of counterexamples. In *Quantitative evaluation of systems, 2008. qest'08. fifth international conference on* (pp. 189–198).

Amazon Web Services. (2018a). *AWS Cloud Products* (Tech. Rep.). Retrieved from https://aws.amazon.com/products/

Amazon Web Services. (2018b). *AWS Customer Success* (Tech. Rep.). Retrieved from https://aws.amazon.com/solutions/case-studies/

Amazon Web Services. (2018c). *AWS Policy Generator* (Tech. Rep.). Retrieved from https://awspolicygen.s3.amazonaws.com/policygen.html

Amazon Web Services. (2018d). *How Amazon S3 Authorizes a Request for a Bucket Operation* (Tech. Rep.). Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/dev/access-control-auth-workflow-bucket-operation.html

Amazon Web Services. (2018e). *How Amazon S3 Authorizes a Request for an Object Operation* (Tech. Rep.). Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/dev/access-control-auth-workflow-object-operation.html

Amazon Web Services. (2018f). *How Do I Enable Default Encryption for an S3 Bucket?* (Tech. Rep.). Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/user-guide/default-bucket-encryption.html

Amazon Web Services. (2018g). *Introduction to Amazon S3* (Tech. Rep.). Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html

A.P. Møller-Mærsk A/S. (2017). *Interim Report Q3 2017* (Tech. Rep.). Retrieved from http://files.shareholder.com/downloads/ABEA-3GG91Y/5744307541x0x962977/A3CA75AA-14E3-4C99-9D57-F184196BADA6/Maersk_Interim_Report_Q3_2017.pdf

APWG. (2017). *Phishing activity trends report* (Tech. Rep.). Anti-Phishing Working Group. Retrieved from https://docs.apwg.org/reports/apwg_trends_report_q4_2016.pdf

Arnold, F., Gebler, D., Guck, D., & Hatefi, H. (2014). A tutorial on interactive markov chains. In A. Remke & M. Stoelinga (Eds.), *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems: International Autumn School, ROCKS 2012, Vahrn, Italy, October 22-26, 2012, Advanced Lectures* (pp. 26–66). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-662-45489-3_2

Asay, M. (2018, February 6). State of the cloud union: How AWS, Microsoft, Google, and Alibaba stack up. *Tech Republic*. Retrieved from https://www.techrepublic.com/article/state-of-the-cloud-union-how-aws-microsoft-google-and-alibaba-stack-up/

Asgharpour, F., Liu, D., & Camp, L. J. (2007). Mental models of security risks. In *International Conference on Financial Cryptography and Data Security* (p. 367-377).

Ashok, I. (2017, December 1). National credit federation data leak: Over 100gb of sensitive customer data was left exposed online. *International Business Times*. Retrieved from http://www.ibtimes.co.uk/national-credit-federation-data-leak-over-100gb-sensitive-customer-data-was-left-exposed-online-1649709

Baier, C., & Katoen, J. P. (2008). *Principles of Model Checking*. MIT press.

Basuki, T. A., Cerone, A., Griesmayer, A., & Schlatte, R. (2009). Model-checking user behaviour using interacting components. *Formal Aspects of Computing*, *21*(6), 571-588.

Bennett, K. B., Posey, S. M., & Shattuck, L. G. (2008). Ecological interface design for military command and control. *Journal of Cognitive Engineering and Decision Making*, *2*(4), 349-385.

Bibel, W. (2007). Early history and perspectives of automated deduction. *KI 2007: Advances in Artificial Intelligence*, 2-18.

Blythe, J., & Camp, L. J. (2012). Implementing mental models. In *2012 IEEE Symposium on Security and Privacy Workshops (SPW)* (p. 86-90).

Bolton, M. L. (2010). *Using task analytic behavior modeling, erroneous human behavior generation, and formal methods to evaluate the role of human-automation interaction in system failure* (Unpublished doctoral dissertation). The University of Virginia.

Bolton, M. L., Bass, E. J., & Siminiceanu, R. I. (2013). Using formal verification to evaluate human-automation interaction: A review. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *43*(3), 488-503.

Bredereke, J. (2003). On preventing telephony feature interactions which are shared-control mode confusions. In *Feature Interaction Workshop 2003* (p. 159-176).

Bredereke, J., & Lankenau, A. (2002). A rigorous view of mode confusion. In *International Conference on Computer Safety, Reliability, and Security* (p. 19-31).

Bredereke, J., & Lankenau, A. (2005). Safety-relevant mode confusions—modelling and reducing them. *Reliability Engineering & System Safety*, *88*(3), 229-245.

Cameron, D. (2017, May 31). Top defense contractor left sensitive pentagon files on amazon server with no password. *Gizmodo*. Retrieved from https://gizmodo.com/top-defense-contractor-left-sensitive-pentagon-files-on-1795669632

Camp, L. J. (2004). Mental models of computer security. *Financial Cryptography*, *3110*, 106-111.

Camp, L. J. (2009). Mental models of privacy and security. *IEEE Technology and Society Magazine*, *28*(3), 37-46.

Camp, L. J., Feamster, N., Feigenbaum, J., Forrest, S., Kotz, D., Lee, W., ... Stolfo, S. (2009). *Data for cyber security: Process and "wish list"*. Retrieved from http://www.ljean.com/files/data-wishlist.pdf

Carroll, J. M., Anderson, N. S., Olson, J. R., Council, N. R., et al. (1987). *Mental models in human-computer interaction: Research issues about what the user of software knows* (No. 12). National Academies.

Chan, W., Anderson, R. J., Beame, P., & Notkin, D. (1998). Improving efficiency of symbolic model checking for state-based system requirements. In *ACM SIGSOFT Software Engineering Notes* (Vol. 23, p. 102-112).

Chechik, M., & Gurfinkel, A. (2007). A framework for counterexample generation and exploration. *International Journal on Software Tools for Technology Transfer*, *9*(5-6), 429-445.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, *5*(2), 121-152.

Clarke, E., Grumberg, O., & Peled, D. (1999). *Model checking*. MIT Press. Retrieved from https://books.google.com/books?id=Nmc4wEaLXFEC

Clement, J. (1983). A conceptual model discussed by galileo and used intuitively by physics students. In *Mental Models* (p. 325-338). Erlbaum: Hillsdale, NJ.

CMU SEI. (2013). *Unintentional insider threats: A foundational study* (Tech. Rep.). Carnegie Mellon University's Software Engineering Inistitute. Retrieved from https://resources.sei.cmu.edu/asset_files/TechnicalNote/2013_004_001_58748.pdf

Cofense. (2016). *Enterprise Phishing Susceptibility and Resiliency Report* (Tech. Rep.). PhishMe. Retrieved from https://cofense.com/enterprise-phishing-susceptibility-report/

Cofense. (2017). *Enterprise Phishing Susceptibility and Defense Report* (Tech. Rep.). PhishMe. Retrieved from https://cofense.com/whitepaper/enterprise-phishing-resiliency-and-defense-report/

Collett, S. (2014, May 21). Five new threats to your mobile device security. *CSO Online*. Retrieved from http://www.csoonline.com/article/2157785/data-protection/five-new-threats-to-your-mobile-device-security.html

Combating the Insider Threat (Tech. Rep.). (2014). U.S. Department of Homeland Security National Cybersecurity and Communications Integration Center. Retrieved from

https://www.us-cert.gov/sites/default/files/publications/Combating%20the%20Insider%20Threat_0.pdf

Combéfis, S., Giannakopoulou, D., & Pecheur, C. (2014). State event models for the formal analysis of human-machine interactions. *Formal Verification and Modeling in Human-Machine Systems: Papers from the AAAI Spring Symposium State*, 15-20.

Combéfis, S., Giannakopoulou, D., & Pecheur, C. (2016). Automatic detection of potential automation surprises for adept models. *IEEE Transactions on Human-Machine Systems*, *46*(2), 267-278.

Combéfis, S., Giannakopoulou, D., Pecheur, C., & Feary, M. (2011). Learning system abstractions for human operators. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering* (p. 3-10).

Combéfis, S., & Pecheur, C. (2009). A bisimulation-based approach to the analysis of human-computer interaction. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (p. 101-110).

Craik, K. (1943). *The Nature of Explanation*. Cambridge University Press.

Curzon, P., Rukšėnas, R., & Blandford, A. (2007). An approach to formal verification of human-computer interaction. *Formal Aspects of Computing*, *19*(4), 513-550.

Cybersecurity Ventures. (2016, Q4). *Cybersecurity Ventures projects $1 trillion will be spent globally on cybersecurity from 2017 to 2021* (Tech. Rep.). Retrieved from http://cybersecurityventures.com/cybersecurity-market-report/

d'Andrade, R. (1987). A folk model of the mind. In *Cultural Models in Language and Thought* (p. 112-148). Cambridge University Press.

Day, M. C., & Boyce, S. J. (1993). Human factors in human-computer system design. *Advances in Computers*, *36*, 333-430.

Degani, A., & Heymann, M. (2002). Formal verification of human-automation interaction. *Human Factors*, *44*(1), 28-43.

Degani, A., Shafto, M., & Kirlik, A. (1999). Modes in human-machine systems: Constructs, representation, and classification. *The International Journal of Aviation Psychology*, *9*(2), 125-138.

DeKeyser, V. (1986). Cognitive development of process industry operators. In *New Technology and Human Error*. Wiley, Chichester.

Dekker, S., & Hollnagel, E. (2004). Human factors and folk models. *Cognition, Technology & Work*, *6*(2), 79–86.

de Kleer, J., & Brown, J. S. (1981). Mental models of physical mechanisms and their acquisition. In *Cognitive Skills and their Acquisition* (p. 285-309). Erlbaum: Hillsdale, NJ.

de Moura, L., Owre, S., & Shankar, N. (2003). *The SAL language manual* (Tech. Rep.). SRI International. Retrieved from http://staffwww.dcs.shef.ac.uk/people/A.Simons/z2sal/saldocs/SALlanguage.pdf

Department of Homeland Security. (2009). *A roadmap for cybersecurity research* (Tech. Rep.). Retrieved from https://www.dhs.gov/sites/default/files/publications/CSD-DHS-Cybersecurity-Roadmap.pdf

Dourish, P., Grinter, R. E., De La Flor, J. D., & Joseph, M. (2004). Security in the wild: User strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing*, *8*(6), 391-401.

Dutton, J. M., & Starbuck, W. (1971). Finding charlie's run-time estimator. In *Computer Simulation of Human Behavior* (p. 218-242). Wiley, New York.

FBI: Cyber Crime. (2016). Retrieved 15 February 2017, from https://www.fbi.gov/investigate/cyber

Federal Avaiation Administration. (2017, November 14). Air traffic by the numbers. Retrieved from https://www.faa.gov/air_traffic/by_the_numbers/

Feng, L., Humphrey, L., Lee, I., & Topcu, U. (2016). Human-interpretable diagnostic information for robotic planning systems. In *IEEE International Conference on Intelligent Robots and Systems (IROS) 2016* (p. 1673-1680).

Feng, L., Wiltsche, C., Humphrey, L., & Topcu, U. (2016). Synthesis of human-in-the-loop control protocols for autonomous systems. *IEEE Transactions on Automation Science and Engineering*, *13*(2), 450–462.

Ferguson, B., Tall, A., & Olsen, D. (2014). National cyber range overview. In *IEEE Military Communications Conference (MILCOM) 2014* (p. 123-128).

Finnan, K., & Melrose, J. (2017, August 18). Cybersecurity for pipelines, other SCADA systems. *Control Engineering*. Retrieved from https://www.controleng.com/single-article/cybersecurity-for-pipelines-other-scada-systems/681927df34de445ea2ee8ae186518395.html

Flanagan, C., & Godefroid, P. (2005). Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices* (Vol. 40, pp. 110–121).

Furman, S., Theofanos, M. F., Choong, Y.-Y., & Stanton, B. (2012). Basing cybersecurity training on user perceptions. *IEEE Security & Privacy*, *10*(2), 40-49.

Gandel, S. (2015, Jan 23). Lloyd's CEO: Cyber attacks cost companies $400 billion every year. *Fortune*. Retrieved from http://fortune.com/2015/01/23/cyber-attack-insurance-lloyds/

Gartner. (2015, Sep 23). *Gartner says worldwide information security spending will grow almost 4.7 percent to reach $75.4 billion in 2015* (Tech. Rep.). Retrieved from https://www.gartner.com/newsroom/id/3135617

Gentner, D., & Gentner, D. R. (1983). Flowing waters or teeming crowds: Mental models of electricity. In *Mental Models* (p. 99-129). Erlbaum: Hillsdale, NJ.

Gentner, D., & Stevens, A. L. (1983). *Mental models*. Psychology Press.

Graham, B. (1998, May 24). U.S. studies a new threat: Cyber attack. *The Washington Post*, p. A01.

Greenhalgh, T., Helman, C., & Chowdhury, A. M. (1998). Health beliefs and folk models of diabetes in British Bangladeshis: A qualitative study. *BMJ*, *316*(7136), 978-983.

Harrison, J. (2013). *A survey of automated theorem proving*. Retrieved from https://www.cl.cam.ac.uk/~jrh13/slides/stpetersburg-28sep13/slides.pdf (Lecture for the St. Petersburg Computer Science Club)

Hayes, P. J. (1989). Naïve physics I: Ontology for liquids. In *Readings in Qualitative Reasoning about Physical Systems* (p. 484-502). Morgan Kaufmann Publishers Inc.

Heckle, R., Lutters, W. G., & Gurzick, D. (2008). Network authentication using single sign-on:

The challenge of aligning mental models. In *Proceedings of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology* (p. 6).

Hendershott, R. J. (2004). Net value: wealth creation (and destruction) during the internet boom. *Journal of Corporate Finance*, *10*(2), 281-299.

Heymann, M., & Degani, A. (2007). Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. *Human Factors*, *49*(2), 311-330.

Higbee, A. (2017, April 3). Spam is spam, phishing is phishing, but phishing is not spam. *Cofense*. Retrieved from https://cofense.com/spam-spam-phishing-phishing-phishing-not-spam/

Hoc, J.-M. (1995). Planning in diagnosing a slow process. *Z Psychol*, *203*(11995), 101-115.

Hollnagel, E., & Woods, D. D. (2005). *Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*. CRC Press.

Hong, J. (2012). The state of phishing attacks. *Communications of the ACM*, *55*(1), 74-81.

IBM. (2015). *2015 IBM Cyber Security Intelligence Index* (Tech. Rep.). IBM X-Force. Retrieved from https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEW03073USEN.PDF

IBM. (2016). *2016 IBM Cyber Security Intelligence Index* (Tech. Rep.). IBM X-Force. Retrieved from https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEJ03320USEN.PDF

Intelligence Advanced Research Projects Activity. (n.d.). Anticipatory intelligence. Retrieved from https://www.iarpa.gov/index.php/about-iarpa/anticipatory-intelligence

Johnson-Laird, P. N. (1970). The perception and memory of sentences. *New Horizons in Linguistics*, 261-270.

Johnson-Laird, P. N. (2005). Mental models and thought. *The Cambridge Handbook of Thinking and Reasoning*, 185-208.

Johnson-Laird, P. N., Byrne, R. M., & Schaeken, W. (1992). Propositional reasoning by model. *Psychological Review*, *99*(3), 418.

Juniper Research. (2015, May 12). *Cybercrime will cost businesses over $2 trillion by 2019*

(Tech. Rep.). Retrieved from https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion

Kamp, H. (2008). A theory of truth and semantic representation. *Formal semantics: The essential readings*, *1*, 189-222.

Kang, R., Dabbish, L., Fruchter, N., & Kiesler, S. (2015). "My data just goes everywhere:" User mental models of the internet and implications for privacy and security. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS) 2015* (p. 39-52). USENIX Association.

Kauer, M., Günther, S., Storck, D., & Volkamer, M. (2013). A comparison of American and German folk models of home computer security. In *International conference on human aspects of information security, privacy, and trust* (p. 100-109).

Kempton, W. (1986). Two theories of home heat control. *Cognitive Science*, *10*(1), 75-90.

Kermelis, M. (2003). Towards an improved understanding of model-checking traces by visualisation. *Master's thesis, University of York, York*.

Kiesler, S., & Goetz, J. (2002). Mental models of robotic assistants. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems* (p. 576-577).

Kim, P. (2015). *The Hacker Playbook 2: Practical Guide to Penetration Testing*. Secure Planet LLC.

Kneuper, R. (1997). Limits of formal methods. *Formal Aspects of Computing*, *9*(4), 379-394.

Koenig, A. (1995). Patterns and antipatterns. *Journal of Object-Oriented Programming*, *8*(1), 46–48.

Kubicek, K., Carpineto, J., McDavitt, B., Weiss, G., Iverson, E. F., Au, C.-W., ... Kipke, M. D. (2008). Integrating professional and folk models of HIV risk: YMSM's perceptions of high-risk sex. *AIDS Education & Prevention*, *20*(3), 220-238.

Kumaraguru, P., Cranshaw, J., Acquisti, A., Cranor, L., Hong, J., Blair, M. A., & Pham, T. (2009). School of phish: A real-world evaluation of anti-phishing training. In *Proceedings of the 5th symposium on usable privacy and security* (p. 3).

Kwiatkowska, M., Norman, G., & Parker, D. (2007). Stochastic model checking. In

International School on Formal Methods for the Design of Computer, Communication and Software Systems (pp. 220–270).

Kwiatkowska, M., Norman, G., & Parker, D. (2011). Prism 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification* (p. 585-591).

Lewis, M., Sycara, K., & Walker, P. (2018). The role of trust in human-robot interaction. In *Foundations of Trusted Autonomy* (p. 135-159). Springer.

Leyden, J. (2017, September 4). Leaky s3 bucket sloshes deets of thousands with us security clearance. *The Register*. Retrieved from https://www.theregister.co.uk/2017/09/04/us_security_clearance_aws_breach/

Mandiant. (2018). *M-Trends 2018 Threat Report* (Tech. Rep.). FireEye. Retrieved from https://www.fireeye.com/current-threats/annual-threat-report/mtrends.html

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. MIT Press.

Mathieu, J. E., Heffner, T. S., Goodwin, G. F., Salas, E., & Cannon-Bowers, J. A. (2000). The influence of shared mental models on team process and performance. *Journal of Applied Psychology*, *85*(2), 273-283.

Merit Network. (2018). *Michigan Cyber Range.* (https://www.merit.edu/cyberrange/)

Microsoft. (2014). *Microsoft 2014 Computing Safety Index* (Tech. Rep.). Microsoft. Retrieved from https://www.microsoft.com/en-US/security/resources/mcsi.aspx

Militello, L. G., Klein, G., Lee, J., & Kirlik, A. (2013). Decision-centered design. In *The Oxford Handbook of Cognitive Engineering* (p. 261-271). Oxford University Press, Oxford, UK.

Mitnick, K. D., & Simon, W. L. (2011). *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons.

MITRE PRE-ATT&CK: Adversarial tactics, techniques, and common knowledge for left-of-exploit. (2018). Retrieved 31 January 2018, from https://attack.mitre.org/pre-attack/index.php/Main_Page

Moher, T. G., & Dirda, V. (1995). Revising mental models to accommodate expectation

failures in human-computer dialogues. In *Design, Specification and Verification of Interactive Systems 1995* (p. 76-92). Springer.

Moray, N. (1999). Mental models in theory and practice. In *Attention and Performance XVII: Cognitive Regulation of Performance: Interaction of Theory and Application* (p. 223-258). MIT Press, Cambridge, MA.

Morgan, S. (2016, Mar 9). Worldwide cybersecurity spending increasing to $170 billion by 2020. *Forbes*. Retrieved from http://www.forbes.com/sites/stevemorgan/2016/03/09/worldwide-cybersecurity-spending-increasing-to-170-billion-by-2020/#7cde23d576f8

Muncaster, P. (2017, September 25). Verizon hit by another Amazon S3 leak. *InfoSecurity Magazine*. Retrieved from https://www.infosecurity-magazine.com/news/verizon-hit-by-another-amazon-s3/

National Cyber Security Strategy 2016-2021 (Tech. Rep.). (2016). Government of the United Kingdom. Retrieved from https://www.enisa.europa.eu/topics/national-cyber-security-strategies/ncss-map/national_cyber_security_strategy_2016.pdf

Newman, L. H. (2017, June 19). The scarily common screw-up that exposed 198 million voter records. *Wired*. Retrieved from https://www.wired.com/story/voter-records-exposed-database/

NIST. (2018). *Cyber Ranges* (Tech. Rep.). Retrieved from https://www.nist.gov/sites/default/files/documents/2018/02/13/cyber_ranges.pdf

Norman, D. A. (1983). Some observations on mental models. In *Mental Models* (p. 7-14). Erlbaum: Hillsdale, NJ.

Norman, D. A. (2013). *The design of everyday things: Revised and expanded edition*. Basic Books.

Northrop Grumman. (n.d.). *Cyber Test Range.* Retrieved from http://www.northropgrumman.com/Capabilities/CyberTestRange/Pages/default.aspx

Novet, J. (2018, February 1). Amazon cloud revenue jumps 45 percent in fourth quarter.

CNBC Enterprise. Retrieved from https://www.cnbc.com/2018/02/01/aws-earnings-q4-2017.html

Obregon, L. (2015). *Infrastructure security architecture for effective security monitoring* (Tech. Rep.). SANS Institute Whitepaper. Retrieved from https://www.sans.org/reading-room/whitepapers/bestprac/infrastructure-security-architecture-effective-security-monitoring-36512

Office of the Director of National Intelligence. (n.d.). What is intelligence? Retrieved from https://www.dni.gov/index.php/what-we-do/what-is-intelligence

Oishi, M., Hwang, I., & Tomlin, C. (2003). Immediate observability of discrete event systems with application to user-interface design. In *Proceedings of the 42nd IEEE Conference on Decision and Control, 2003* (Vol. 3, p. 2665-2672).

Oishi, M., Mitchell, I., Bayen, A. M., & Tomlin, C. J. (2008). Invariance-preserving abstractions of hybrid systems: Application to user interface design. *IEEE Transactions on Control Systems Technology*, *16*(2), 229-244.

Oliveira, R., Palanque, P., Weyers, B., Bowen, J., & Dix, A. (2017). State of the art on formal methods for interactive systems. In *The Handbook of Formal Methods in Human-Computer Interaction* (p. 3-55). Springer.

O'Sullivan, D. (2017, November 28). Black box, red disk: How top secret NSA and Army data leaked online. *UpGuard*. Retrieved from https://www.upguard.com/breaches/cloud-leak-inscom

Palo Alto Networks. (n.d.). *Cyber Range*. Retrieved from https://www.paloaltonetworks.com/solutions/initiatives/cyberrange-overview

Paloma, J. (2007). *Windows server 2008 in an organization's defense in depth strategy* (Tech. Rep.). Microsft TechNet. Retrieved from https://technet.microsoft.com/en-us/library/cc512681.aspx

Plato. (2007). *Theaetetus, translated by Benjamin Jowett*. Bibliobazaar.

Prettyman, S. S., Furman, S., Theofanos, M., & Stanton, B. (2015). Privacy and security in the brave new world: The use of multiple mental models. In *International Conference on Human Aspects of Information Security, Privacy, and Trust* (p. 260-270).

Proofpoint. (2017). *The Human Factor* (Tech. Rep.). Retrieved from https://www.proofpoint.com/us/thank-you-human-factor-report-2017

Pynadath, D. V., & Marsella, S. (2007). Minimal mental models. In *Proceedings of the 22nd National Conference on Artificial Intelligence* (p. 1038-1044). AAAI.

Raja, F., Hawkey, K., Hsu, S., Wang, K.-L., & Beznosov, K. (2011). Promoting a physical security mental model for personal firewall warnings. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems* (p. 1585-1590).

Rashid, F. Y. (2015, Jan 26). Enterprises overly reliant on perimeter-based defenses: Survey. *Security Week*. Retrieved from http://www.securityweek.com/enterprises-overly-reliant-perimeter-based-defenses-survey

Rasmussen, J. (1986). *Information processing and human-machine interaction: An approach to cognitive engineering*. North-Holland.

Rasmussen, J., & Rouse, W. B. (2013). *Human detection and diagnosis of system failures* (Vol. 15). Springer Science & Business Media.

Raytheon. (n.d.). *Cyber Range.* Retrieved from https://www.raytheon.com/cyber/capabilities/range

Reynolds, R. E., Sinatra, G. M., & Jetton, T. L. (1996). Views of knowledge acquisition and representation: A continuum from experience centered to mind. *Educational Psychologist*, *31*(2), 93-104.

Rouse, W. B., & Hunt, R. M. (1986). *Human problem solving in fault diagnosis tasks* (Tech. Rep.). DTIC Document.

Rouse, W. B., & Morris, N. M. (1986). On looking into the black box: Prospects and limits in the search for mental models. *Psychological Bulletin*, *100*(3), 349.

Rushby, J. (2001a). Analyzing cockpit interfaces using formal methods. *Electronic Notes in Theoretical Computer Science*, *43*, 1-14.

Rushby, J. (2001b). Modeling the human in human factors. In *International Conference on Computer Safety, Reliability, and Security* (p. 86-91).

Rushby, J. (2002). Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering & System Safety*, *75*(2), 167-177.

Rushby, J. (2007). *Introduction to SMT solving and infinite bounded model checking.* Retrieved from http://www.csl.sri.com/users/rushby/slides/jaist07.pdf (Presentation for SRI International)

Rushby, J., Crow, J., & Palmer, E. (1999). An automated method to detect potential mode confusions. In *Proceedings of the 18th Digital Avionics Systems Conference, 1999* (Vol. 1, p. 4-B).

Samarati, M. (2016, November 15). Accidental or malicious insider threat: Staff awareness makes the difference. *IT Governance*. Retrieved from https://www.itgovernance.co.uk/blog/accidental-or-malicious-insider-threat-staff-awareness-makes-the-difference/

Sanderson, P. M. (1990). Knowledge acquisition and fault diagnosis: Experiments with PLAULT. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*(1), 225-242.

Schneier, B. (2011, January 28). Whitelisting vs. blacklisting. *Schneier on Security*. Retrieved from https://www.schneier.com/blog/archives/2011/01/whitelisting_vs.html

Schwartz, D. L., & Black, J. B. (1996). Analog imagery in mental model reasoning: Depictive models. *Cognitive Psychology*, *30*(2), 154-219.

Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L. F., & Downs, J. (2010). Who falls for phish? A demographic analysis of phishing susceptibility and effectiveness of interventions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (p. 373-382).

Sheridan, T. B. (1976). Toward a general model of supervisory control. In *Monitoring Behavior and Supervisory Control* (p. 271-281). Springer.

Siadati, H., Palka, S., Siegel, A., & McCoy, D. (2017). Measuring the effectiveness of embedded phishing exercises. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*.

Simmons, R., & Pecheur, C. (2000). Automating model checking for autonomous systems. In *AAAI Spring Symposium on Real-Time Autonomous Systems.*

Singh, S. (2000). *The code book: The science of secrecy from ancient egypt to quantum cryptography*. Anchor.

SolarWinds. (2016). *SolarWinds Federal Cybersecurity Survey Summary Report* (Tech. Rep.). SolarWinds. Retrieved from http://www.solarwinds.com/assets/surveys/cybersecurity-slide-deck.aspx

Symantec. (2016). *2016 internet security threat report* (Tech. Rep.). Symantec. Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

Taleb, N. N. (2007). *The black swan: The impact of the highly improbable*. Random House.

The Department of Defense. (2015). *The DoD Cyber Strategy* (Tech. Rep.). Retrieved from https://www.defense.gov/Portals/1/features/2015/0415_cyber-strategy/Final_2015_DoD_CYBER_STRATEGY_for_web.pdf

Torff, B. (1999). Tacit knowledge in teaching: Folk pedagogy and teacher education. *Tacit Knowledge in Professional Practice*, 195-214.

UK National Audit Office. (2017). *Investigation: Wannacry cyber attack and the NHS* (Tech. Rep.). UK National Audit Office. Retrieved from https://www.nao.org.uk/wp-content/uploads/2017/10/Investigation-WannaCry-cyber-attack-and-the-NHS.pdf

Urban, G. D. (2011). Causing damage without authorization: The limitations of current judicial interpretations of employee authorization under the Computer Fraud and Abuse Act. *William & Mary Law Review*, *52*(4), 1368-1412.

Verizon Enterprise. (2016). *2016 Data Breach Investigations Report* (Tech. Rep.). Verizon Enterprise. Retrieved from http://www.verizonenterprise.com/verizon-insights-lab/dbir/2016/

Vicente, K. J., & Rasmussen, J. (1992). Ecological interface design: Theoretical foundations. *IEEE Transactions on Systems, Man, and Cybernetics*, *22*(4), 589-606.

Virginia Cyber Range. (2018). Retrieved from https://virginiacyberrange.org/

Visser, W., & Barringer, H. (2000). Practical CTL* model checking: Should SPIN be extended? *International Journal on Software Tools for Technology Transfer*, *2*(4), 350-365.

Voas, J. M., & Schaffer, K. B. (2016). Insights on formal methods of cybersecurity. *Computer (IEEE Computer)*, *49*.

Warner, M. (2012). Cybersecurity: A pre-history. *Intelligence and National Security*, *27*(5), 781-799.

Wash, R. (2010). Folk models of home computer security. In *Proceedings of the Sixth Symposium on Usable Privacy and Security (SOUPS) 2010* (p. 11).

Weirich, D., & Sasse, M. A. (2001). Pretty good persuasion: A first step towards effective password security in the real world. In *Proceedings of the 2001 Workshop on New Security Paradigms* (p. 137-143).

Whittaker, Z. (2017, July 12). Millions of verizon customer records exposed in security lapse. *InfoSecurity Magazine*. Retrieved from http://www.zdnet.com/article/millions-verizon-customer-records-israeli-data/

Wickens, C. D., Hollands, J. G., Banbury, S., & Parasuraman, R. (2015). *Engineering Psychology & Human Performance*. Psychology Press.

Williams, M. D., Hollan, J. D., & Stevens, A. L. (1983). Human reasoning about a simple physical system. In *Mental Models* (p. 131-154). Hillsdale, NJ: Lawrence Erlbaum.

Wilson, J. R., & Rutherford, A. (1989). Mental models: Theory and application in human factors. *Human Factors*, *31*(6), 617-634.

Wing, J. M. (1990). A specifier's introduction to formal methods. *Computer*, *23*(9), 8-22.

Woods, D. D., & Hollnagel, E. (2006). *Joint cognitive systems: Patterns in cognitive systems engineering*. CRC Press.

Woods, D. D., & Roth, E. (1988). Cognitive systems engineering. In *Handbook of Human-Computer Interaction* (p. 3-44). Elsevier.

Young, L. (1969). On adaptive manual control. *Ergonomics*, *12*(4), 635-674.