KNOWLEDGE REUSE AS A FOUNDATION FOR SECURITY METRICS

by

Nilofar Mansourzadeh

Submitted in partial fulfillment of the requirements
for the degree of PhD of Computer Science

at

Carleton University
Ottawa, Ontario
May 2024

Table of Contents

List of Figures

Abstract

The recurring question within the cybersecurity landscape is—why do systems keep getting successfully attacked? This relative ease of system breaches necessitates a reevaluation of what it means for attacks to be "hard". Existing security metrics quantify narrow aspects of attack difficulty but do not give insight into why attacks remain feasible. Diversity is frequently discussed as a strategy to mitigate the risks associated with cybersecurity monoculture. However, determining the optimal level and type of diversity in security presents complex challenges for which we have little basis.

We propose a new way of conceptualizing attack difficulty in terms of knowledge reuse. To this end, we present two models—one abstract model inspired by artificial intelligence search and another model, product family algebra, inspired by software components. Using these models we propose three new metrics: knowledge obfuscation (KOSM), defense evolution (DESM), and population fragility. We then analyze the practical implications of our proposed models for an information security analyst. By adopting these models, information security analysts can better navigate the complexities of the cybersecurity landscape, tailor security measures more precisely to their organization's needs, and ultimately contribute to a more resilient and secure digital environment.

Acknowledgements

This thesis is dedicated to two remarkable individuals who have been my pillars of support, inspiration, and encouragement throughout this journey.

To Dr. Anil Somayaji, my esteemed supervisor, whose wisdom, guidance, and unwavering belief in my potential have been instrumental in shaping not only this research but also my growth as a scholar. Your mentorship has been a beacon of light, guiding me through the challenges and celebrating the milestones. I am deeply grateful for your invaluable contributions to my academic and personal development.

And to my husband, Dr. Amir Aghasharif, who has been my steadfast companion in every sense. Your love, patience, and unwavering support have been my sanctuary. Your belief in me and my aspirations has been a source of strength and motivation, making this journey not only possible but also joyful. I am forever thankful for your companionship, understanding, and endless encouragement.

This work is a testament to the profound impact you both have had on my life. Thank you for being my guiding stars.

Chapter 1

Introduction

This dissertation explores the premise that models of knowledge reuse can significantly bolster the foundation of computer security. In an era where cyber threats evolve with alarming speed and ingenuity, traditional defensive mechanisms often fall short. Through this work, I endeavor to bridge this gap by presenting an analysis that not only highlights the critical role of knowledge reuse in the persistence of cyber threats but also proposing a novel framework for integrating this concept into the development of more resilient security strategies. The motivation behind this approach stems from the observation that despite significant advancements in defensive technologies, attackers continue to outpace these measures by leveraging and innovating upon existing knowledge. By understanding and modeling this process of knowledge reuse, we can anticipate and counteract emerging threats more effectively.

The dissertation unfolds this narrative through a detailed examination of the underlying motivations, a strategic approach that leverages both theoretical and practical methodologies, significant contributions that push the boundaries of current security paradigms, and a structured outline that guides the reader through the intricate layers of this argument. This introductory chapter sets the stage for a deep dive into these components, laying the groundwork for a nuanced discussion on enhancing computer security in the face of evolving cyber threats.

1.1 Motivation

The persistent ease with which cyber-attacks are executed poses a formidable challenge in the realm of computer security, undermining the efficacy of even the most advanced defense mechanisms. This paradoxical scenario, where significant technological advancements in security seem insufficient to deter attackers, forms the core motivation for this dissertation. The question arises: why do these advancements fail to make a substantial impact?

To understand this, we must consider the nature of the defenses deployed. Recent years have seen the introduction of sophisticated encryption algorithms, intrusion detection systems capable of real-time monitoring and response, and comprehensive malware analysis tools that employ machine learning to predict and neutralize threats. Yet, these measures often fall short. The reason lies not in their lack of sophistication, but in the adaptive and innovative capabilities of attackers. For every new defense introduced, attackers find novel ways to circumvent it. For instance, against advanced encryption, attackers have developed ransomware that encrypts user data with their keys, effectively turning a defense mechanism against the user. Intrusion detection systems are bypassed through the use of polymorphic malware that changes its signature with each replication, making detection notoriously difficult.

The evolution of attack strategies further complicates this landscape. Cyber-attacks have grown from simple viruses and worms to complex, multi-vector assaults that exploit a range of vulnerabilities. Consider the progression from basic phishing scams to spear-phishing and whaling, where attackers tailor their deception to specific targets with alarming precision. Similarly, distributed denial-of-service (DDoS) attacks have evolved from mere bandwidth flooding to sophisticated application-layer attacks that mimic legitimate user requests, making them harder to detect and mitigate.

This continuous innovation in attack strategies is fueled by a culture of knowledge reuse among attackers. Once a new technique is developed, it doesn't remain in the shadows for long. Detailed analyses and proof-of-concept codes often find their way into academic publications, hacker forums, and dark web marketplaces. This information dissemination transforms singular innovations into reusable tools and techniques that can be adapted and applied by a wide range of malicious actors. Vulnerability marketplaces commodify exploits, making them accessible to anyone willing to pay, while "as-a-service" models for deploying malware, such as botnets or ransomware, offer turnkey solutions for inflicting damage.

This ecosystem of shared knowledge and tools underpins the concept of the "security arms race," a never-ending cycle of action and reaction between defenders and attackers. Recognizing this dynamic is crucial, but it also begs the question: how can we break this cycle, or at least, gain the upper hand?

1.2 Approach

This dissertation proposes that the key to a more secure future lies in understanding and modeling the knowledge reuse practices of attackers. By conceptualizing and quantifying how attackers learn from each other and build upon existing techniques, we can develop a foundation for computer security that not only anticipates emerging threats but also evaluates the long-term efficacy of defense strategies. Such models can offer insights into the most resilient forms of defense, those that not only block current attacks but also inhibit the future reuse of knowledge, thereby elevating the cost and complexity of launching successful cyber-attacks. Through this lens, we aim to shift the balance of the security arms race, crafting a strategy that is proactive rather than reactive, and grounded in a deep understanding of the adversary's playbook.

In the pursuit of a more robust computer security framework, understanding and modeling knowledge reuse becomes pivotal. Knowledge, in the context of cybersecurity, is not monolithic; it is multifaceted, encompassing everything from theoretical concepts and tutorials to practical applications like code snippets and fully-fledged software tools. Each of these elements plays a role in how knowledge is transferred, adapted, and applied in new contexts, particularly among attackers seeking to exploit system vulnerabilities.

To grasp the complexity of knowledge reuse, we can draw parallels from the field of artificial intelligence (AI), where knowledge generation is often conceptualized as a search process. In AI, the search space represents the universe of possible solutions to a problem, and the process of search involves navigating this space to find a solution that meets the desired criteria. This solution, once found, constitutes the "knowledge" that was sought. This fundamental concept is well-established in AI literature and provides a useful framework for understanding knowledge reuse in cybersecurity.

Applying this framework, we can envision attackers as operating within a search space defined by the system's architecture and its defenses. Each layer of defense, each algorithm, and each protocol sets boundaries for this search space, guiding or constraining the attacker's exploration. The effectiveness of these defenses, therefore, can be assessed by how well they limit the attacker's ability to navigate the search space and identify exploitable vulnerabilities.

This concept is not entirely dissimilar to the principles of encryption, where the

search space is constituted by the possible keys, and the attacker's goal is to find the correct key that decrypts the desired information. In the broader context of system security, however, the "key" is metaphorical, representing any potential vulnerability or exploit that could be used to breach the system. This includes not just cryptographic weaknesses but any aspect of the system that could be exploited, from hardware and software vulnerabilities to human factors such as social engineering.

Within this extended search space, knowledge reuse among attackers becomes a tool for efficiency. Rather than exploring the search space from scratch, attackers build upon existing knowledge—be it conceptual theories that provide a deeper understanding of potential vulnerabilities, tutorials that guide less experienced attackers through complex attack methodologies, or code examples and software tools that can be directly employed or adapted for new attacks.

For example, a conceptual understanding of buffer overflow vulnerabilities might be documented in academic papers or technical blogs, providing a theoretical foundation that attackers can leverage to understand and identify similar weaknesses in other systems. Tutorials available on various forums and websites might offer step-by-step guides on executing a buffer overflow attack, reducing the barrier to entry for less skilled attackers. Similarly, code examples shared in repositories or found in malware databases can be repurposed with slight modifications to exploit the same or similar vulnerabilities in different contexts. Fully-fledged software tools, like exploit frameworks, offer a more direct application, enabling attackers to launch sophisticated attacks without a deep understanding of the underlying vulnerabilities.

This process of knowledge reuse among attackers is facilitated by a variety of platforms, from open academic publications and white-hat disclosures to darker corners of the internet like hacker forums and dark web marketplaces. Each of these platforms contributes to the shared pool of knowledge, making it easier for attackers to find and apply information relevant to their current objectives.

By modeling this process of knowledge reuse, we can gain insights into how attackers navigate the search space of system vulnerabilities and how they leverage existing knowledge to expedite this process. Understanding these dynamics is crucial for developing defenses that not only secure systems against known threats but also anticipate and mitigate the ways in which attackers might adapt and apply existing

knowledge to circumvent new security measures. This approach calls for a security paradigm that is dynamic and adaptive, capable of evolving in response to the changing landscape of threats and the continuous reuse of knowledge within the attacker community.

1.3 Contributions

The key contributions of this dissertation are as follows.

The cornerstone of this dissertation is the development of a systematic model to assess the difficulty of launching successful cyber attacks, a new approach that challenges traditional reactive security paradigms by proposing a proactive framework. This model offers a dynamic perspective that captures the ongoing interplay between attacker capabilities and system defenses, providing insights into how to understand and mitigate threats effectively. It marks a shift from static to dynamic evaluations of security, emphasizing the need for adaptive and forward-thinking defense strategies in response to evolving cyber threats.

Central to our methodology is the introduction of an abstract model that conceptualizes attack strategies within the framework of search spaces—a novel idea adapted from artificial intelligence. This model suggests that attackers navigate through a "search space" of potential vulnerabilities and attack vectors, influenced by the system's architecture, security measures, and the attackers' knowledge and resources. By quantifying this search space, the model facilitates a structured analysis of potential attack paths and evaluates the effectiveness of different defensive strategies in either expanding or constraining these paths.

A significant contribution of this dissertation is the introduction of new security metrics that provide a more nuanced and strategic assessment of security postures. Traditional metrics often focus on easily quantifiable indicators such as the number of vulnerabilities patched, mean time to detect (MTTD), or compliance checklists, which do not fully capture the complexities and evolving nature of modern security challenges. The proposed metrics in this research address these gaps by evaluating not just the presence of vulnerabilities, but also the context in which security controls operate and how changes in those controls impact the overall difficulty and dynamics of potential attacks.

These metrics add substantial value by providing a multi-dimensional view of security that considers factors like system adaptability, resilience, and the shifting nature of threats. They allow organizations to measure how various defense mechanisms contribute to reducing attack success rates, rather than merely counting occurrences or events. This deeper insight helps to prioritize security investments and optimize response strategies in ways that traditional metrics cannot.

Even if these new metrics cannot always be directly calculated due to the complexity or dynamic nature of systems, they still offer critical benefits. They serve as conceptual models that guide security decisions by highlighting areas where defenses are most effective and where they might fail under evolving conditions. For instance, they can be used to simulate potential outcomes, conduct what-if analyses, and anticipate how attackers might adapt to new defenses. In cases where direct measurement is impractical, the metrics provide a framework for estimating risk and resilience, encouraging a proactive rather than reactive security posture.

Ultimately, these metrics push the boundaries of how we assess security by emphasizing the quality and effectiveness of security measures over simple quantitative benchmarks. They shift the focus from just knowing the number of risks to understanding the strategic implications of those risks and the effectiveness of the mitigations in place. This approach not only fills a critical gap in current security evaluations but also promotes a more adaptive and resilient defense strategy.

Further bridging the gap between theory and practice, this work presents a new model using product family algebra (PFA) [35] to simulate how knowledge reuse, specifically the reuse of exploits, impacts software ecosystems. This model offers a concrete framework to understand the propagation of exploits through software components and their potential mitigation. It provides crucial insights into the vulnerabilities of complex software systems and the interdependencies within software ecosystems, aiding in the development of strategies to enhance overall security.

Finally, the dissertation proposes a novel methodology for security analysis that leverages the developed models to assess the robustness of security measures. This methodology offers a systematic approach to evaluate not just the immediate effectiveness of security controls but also their long-term impact on the attack landscape.

It aims to equip security practitioners with tools for informed decision-making, effective resource prioritization, and the design of resilient security architectures that can withstand evolving cyber threats.

Collectively, these contributions represent a significant advancement in the field of computer security, offering new perspectives and tools for addressing the complex and dynamic challenges posed by cyber threats. By integrating theoretical insights with practical models and methodologies, this dissertation aims to pave the way for more strategic, effective, and adaptive security practices.

1.4 Chapter Outline

The structure of this dissertation is designed to methodically unravel the complexities of computer security through the lens of knowledge reuse and the innovative modeling of attack difficulty. Each chapter builds upon the previous, creating a cohesive narrative that not only presents new concepts but also integrates them into a comprehensive framework for understanding and addressing cyber threats.

The Background chapter lays the foundational knowledge necessary for navigating the subsequent discussions. It provides an overview of existing computer security paradigms, the history and evolution of cyber threats, and the traditional responses to these threats. This chapter sets the stage by highlighting the limitations of current approaches, thereby underscoring the need for new perspectives and methodologies in tackling security challenges. It contextualizes the research within the broader field of computer security, ensuring that readers have a firm grounding in the key concepts and terminologies that underpin the dissertation's arguments.

In the Motivation chapter, the focus shifts to the core issue that this dissertation seeks to address: the seemingly effortless ability of attackers to circumvent advanced defense mechanisms. By delving into the reasons behind the persistent success of cyber-attacks, this chapter elucidates the critical role of knowledge reuse among attackers and the consequent need for a paradigm shift in how defenses are conceptualized and deployed. Through examples and analyses, it demonstrates how the current reactive security measures fall short, setting the stage for the introduction of new models and methodologies that aim to preempt and counteract the evolving threat landscape.

The Modelling Attack Difficulty Using Knowledge Reuse chapter introduces the dissertation's central proposition: the modeling of attack difficulty through the framework of AI-derived search spaces. This chapter breaks down the abstract model, explaining how it represents the interaction between attackers and defenses as a navigational challenge within a metaphorical search space of vulnerabilities and exploits. It elaborates on how this model provides a structured method to analyze potential attack paths and the effectiveness of defenses in constraining these paths. This chapter is pivotal as it lays the theoretical foundation for the subsequent application and analysis of the model in practical scenarios.

Building on the theoretical framework established in chapter 4, A Case Study Using Product Family Algebra chapter presents a concrete application of these concepts through the development of a probabilistic finite automata model for software components. This chapter explains how the PFA model simulates the dynamics of exploit propagation and mitigation within software ecosystems, offering insights into the vulnerabilities and interdependencies that characterize these systems. It demonstrates the practical utility of the abstract models in understanding and managing the specific challenges posed by software vulnerabilities and exploit reuse.

The Knowledge Reuse and Security Analysis chapter outlines a novel approach to evaluating the security of systems based on the previously introduced models. It details a structured methodology that allows for the assessment of defense mechanisms not just in terms of their immediate efficacy but also their strategic impact on the broader attack landscape. This chapter provides a bridge between theory and practice, offering guidelines for applying the dissertation's concepts to real-world security challenges and decision-making processes.

The final chapter, Conclusion, synthesizes the insights gained throughout the dissertation, reflecting on the implications of the research findings for the field of computer security. It evaluates the strengths and limitations of the proposed models and methodologies, considering their practical applicability and potential areas for further research. This chapter aims to encapsulate the dissertation's contributions to the ongoing discourse on computer security, highlighting how the new perspectives and tools introduced can inform and enhance future security strategies.

Together, these chapters weave a narrative that progresses from identifying the

shortcomings of current security paradigms to proposing and elaborating on new models and methodologies designed to address these challenges. The dissertation culminates in a discussion that not only reflects on the journey but also looks forward to the path ahead in the quest for more resilient and adaptive security frameworks.

Chapter 2

Background

In the evolving landscape of computer security, understanding and increasing the difficulty of attacks has become a central theme. This chapter delves into the multifaceted approaches that security experts have adopted to fortify systems against ever-more sophisticated threats. It begins with an exploration of threat modeling, a foundational aspect where defenses are meticulously constructed around the potential threats a system must withstand. This section discusses various threat modeling techniques, including the use of attack trees, and underscores the inherent assumptions about attacker strength and capabilities, which often overlook the dynamic nature of threat actors' adaptability.

Next, this chapter examines the role of trust in computer security and then transitions into the realm of cryptography, where the quest for unassailable security through mathematical proofs has led to the concept of provable security, aiming for an ideal state where attack difficulty is theoretically infinite. This section evaluates the strengths and limitations of cryptographic defenses, from the reliance on entropy as a measure of security to the practical challenges that arise when theoretical security models meet the realities of implementation. It highlights the discrepancies between the assumed and actual attack difficulties, exacerbated by advances in attack methodologies and unforeseen vulnerabilities within cryptographic systems.

Following the discussion on cryptography, the chapter progresses to explore the realm of software and systems modeling, specifically focusing on the concept of feature modeling. This methodology enables the identification and representation of commonalities and variations between computer systems and their components, offering a structured approach to understanding and managing the complexity inherent in modern software systems.

The chapter then addresses the concept of security metrics, tools designed to quantitatively assess the security posture of a system. It critically examines the challenges

in developing metrics that accurately reflect the complexity of attack difficulty and the limitations in using these metrics to achieve a comprehensive understanding of a system's security.

Finally, this chapter discusses software diversity and efforts to measure software diversity for security. From this discussion it becomes clear that although diversity has been identified as an important security strategy, past work has not established viable metrics for security and indeed has not established precise rationale for when diversity is a useful defense strategy.

Through this structured exploration, the chapter aims to provide a holistic view of the strategies employed to enhance attack difficulty, highlighting the successes, challenges, and ongoing efforts in the field of computer security.

2.1 Threat Modeling

In the intricate and ever-evolving realm of cybersecurity, the formulation of a detailed and comprehensive threat model emerges as a cornerstone for establishing effective and tailored defensive strategies [12, 81]. For example, consider a financial institution facing threats from both sophisticated cybercriminals aiming to breach data and insider threats from employees with access to sensitive information. This model is paramount not only in pinpointing potential threats but also in serving as a criterion for gauging the sufficiency and efficacy of the security measures that are put into place. Taking this financial institution as an example, the threat model would need to identify potential phishing attacks, DDoS attacks aimed at disrupting service, and the risk of insider fraud or data theft.

The process is inherently iterative, demanding continuous reassessment and refinement of defenses to adeptly confront any new or inadequately mitigated threats. For instance, after identifying a new malware variant targeting the banking sector, the institution must update its threat model and enhance its antivirus and intrusion detection systems to counter this specific threat. This proactive stance fortifies the system's defenses, ensuring robustness against a dynamic array of cybersecurity challenges.

The foundation of threat modeling lies in the assumptions made about potential threats and the system's inherent security features. These assumptions are critical for

simplifying complex security analyses but also introduce a risk of narrowing the focus, potentially overlooking unconventional or evolving threats. The STRIDE framework, for instance, offers a comprehensive categorization of threats but also acknowledges the possibility of internal threats, emphasizing a holistic approach to threat modeling that considers both internal and external threat vectors [35, 14].

The challenge lies in balancing the simplification of complex security analyses with the need for a comprehensive view that accounts for the adaptability and sophistication of potential attackers. For cybersecurity professionals, leveraging models like STRIDE while remaining vigilant to their limitations involves ensuring that threat models are periodically revisited and updated in response to the changing threat landscape. As an example, a tech company may initially focus on protecting intellectual property from external cyber-espionage but, upon reassessment, realize the growing risk of supply chain attacks, prompting an update to their threat model to include this vector.

The process of threat modeling is inherently based on a set of assumptions that aim to simplify the threat landscape for more manageable analysis. These assumptions, however, can sometimes inadvertently narrow the scope of the analysis, potentially leaving certain vulnerabilities unaddressed. This limitation is particularly evident in models like STRIDE, which, despite its comprehensive approach to categorizing threats, may not fully account for the dynamic nature of insider threats or the evolving tactics of sophisticated attackers. The challenge for cybersecurity professionals is to leverage these models while remaining vigilant to the limitations imposed by their underlying assumptions, ensuring that threat models are periodically revisited and updated in response to the changing threat landscape [15, 42, 81].

The evolution of threat modeling reflects the cybersecurity community's ongoing efforts to enhance the efficiency and comprehensiveness of this critical process. The advent of automated tools, such as the fuzzy logic-based technique and the SDL threat model tool, represents a significant advancement in threat modeling methodologies. These tools incorporate sophisticated concepts like uncertainty modeling and collaborative analysis, aiming to broaden the scope of threat modeling beyond the limitations of traditional manual processes. However, the reliance on automated tools introduces new challenges, particularly in their ability to stay abreast of the latest threats and

attack vectors, underscoring the need for continuous refinement and adaptation to ensure their effectiveness [59, 72, 12].

The proliferation of diverse threat modeling methodologies, such as Abuser Stories and attack trees, highlights the cybersecurity community's response to the complex and evolving nature of cyber threats. For instance, an e-commerce platform might use Abuser Stories to imagine scenarios where attackers exploit web application vulnerabilities to conduct SQL injection attacks [17], while utilizing attack trees to break down the steps of a complex cross-site scripting (XSS) attack [82]. These methodologies offer perspectives on threat modeling, emphasizing the importance of understanding the system from an attacker's viewpoint and breaking down complex attack scenarios into manageable components. The challenge lies in leveraging these diverse methodologies to construct a comprehensive threat model that accurately reflects the multifaceted nature of the threat landscape, ensuring that security measures are both relevant and robust [1, 34, 36, 46, 70].

2.2 Trust

In computer security, the concept of trust is used to discuss how security failures in certain components can compromise the security of other parts of a system. Fragility can be seen as a measure of trust arising from the use of shared components, and thus can be seen as trust metric. Here we review different approaches to trust in computer security.

Trust is a foundational concept in computer security, going back to the earliest work in the field. Trust is used in many contexts; here we attempt to briefly survey them in the context of our work by highlighting work on distinct areas related to trust in security. For a more detailed review, see [31].

Trusted vs. Trustworthy: While the concept of trust is central, the term is used in two distinct ways: components that are *trusted*, and components that are *trustworthy*. While closely related, their definitions are often confused. A trusted component is one whose failure will result in the failure of the entire security of the system; however, a trustworthy component is one that does not fail [5].

Reference Monitors, TCB: Reference monitors, components that centralize access control decisions, are a classic example of a trusted component. If a reference

monitor fails, all access control decisions are potentially invalidated. The Orange Book [48] details strategies for making trusted components trustworthy. Specifically, it details methodologies for assurance, i.e., assuring trusted components are trustworthy. Because assurance is often expensive and requires a significant amount of effort, the parts of the system that are trusted—the trusted computing base (TCB)—should be minimized in terms of their size and complexity. The Orange Book defined six evaluation classes that would give successively higher levels of assurance; the later Common Criteria [61] offered a more flexible model of assurance, supporting different predefined protection profiles. While the concept of a TCB has been criticized as being an inappropriate basis for designing secure systems [11], it remains a foundational idea in the field.

Critical Infrastructure: Work on trust at an organizational level has been used to better secure critical infrastructure [8, 10], secure access controls, and secure information communications [52, 80]. These approaches focus on the idea that to maintain computer systems trusted in an organization and to manage system disruptions, dependencies and interdependencies should be secured. Otherwise, a successful attack on one computer system could result in large numbers of consequences which could severely affect the entire organization or network.

One of the key motivations for determining what components or systems are trusted is to limit the scope of what needs to be secured. Today, however, systems rarely have a small TCB and thus many components must be considered "trusted" even when they may not be trustworthy. As a result, most of the time a full security analysis has to consider most if not all of a system.

2.3 Cryptography

Cryptography, a cornerstone of modern digital security, intertwines the realms of mathematics and computer science to safeguard communication and data against unauthorized access and tampering. This intricate field is underpinned by a set of rigorous assumptions—ranging from the complexity of mathematical puzzles to the unpredictability of random number generation—forming the bedrock of its defensive mechanisms. Over the years, the cryptographic landscape has been marked by

a dynamic evolution, characterized by the proposal, scrutiny, and eventual obsolescence of various algorithms and protocols in response to the advancing capabilities of adversaries and the relentless progress of computational technology.

This narrative not only delineates the contours of what it means to be secure in a digital age but also reflects the profound impact of knowledge creation on the very fabric of cybersecurity. As we delve into the cryptographic domain, we traverse a journey from foundational principles and the quest for provable security to the nuanced interplay between theoretical assurances and practical vulnerabilities, all the while considering the pivotal role of entropy as a measure of attack difficulty and the complexities that transcend apparent security measures.

At the heart of cryptographic security are mathematical challenges such as factoring large prime numbers, which is central to RSA encryption, or computing discrete logarithms used in elliptic curve cryptography (ECC) [58]. These tasks are deemed difficult based on current mathematical knowledge and computational capabilities, forming a solid wall against attackers. For example, RSA encryption relies on the difficulty of factoring the product of two large prime numbers, a task that becomes exponentially harder as the size of the numbers increases [58]. This mathematical complexity ensures that, without the private key, decrypting the information becomes practically impossible for the attacker.

Moreover, the integrity of cryptographic defenses is deeply intertwined with the unpredictability inherent in random number generation, a critical element for key generation processes, cryptographic protocols, and encryption schemes. The security of numerous cryptographic algorithms is predicated on the premise that the random generation or prediction of keys is beyond the reach of attackers, thus safeguarding the confidentiality and integrity of encrypted data [2].

Another pivotal assumption underpinning these defenses is the impracticality of inverting cryptographic functions without access to the corresponding secret keys. Functions such as hash algorithms are engineered to be unidirectional, implying that deducing the original input from the output should be computationally unfeasible, thereby protecting data from unauthorized reversals and upholding its integrity.

These foundational principles underpin the cryptographic safeguards that enable secure data exchange over vulnerable digital channels, ensuring the confidentiality,

integrity, and authenticity of communications. However, the quest for absolute security is perpetual, as advancements in computational power and analytical techniques pose continual challenges to cryptographic systems. This reality necessitates a relentless cycle of assessment, refinement, and evolution of cryptographic practices, with the goal of staying ahead of potential threats. The ongoing development and adaptation of cryptographic algorithms and protocols are crucial for maintaining robust defenses against an ever-evolving array of cybersecurity threats, illustrating the dynamic interplay between theoretical security models and practical cryptographic applications [58, 2].

2.4 Modeling Commonalities in Software and Systems

Feature modeling enables us to represent the commonalities and variabilites between computer systems and their components. In this section, we discuss different feature modeling techniques and we discuss their strengths and weaknesses.

Feature models (FMs) were first introduced and used by Kang et al. in 1990 [44]. They have since become well accepted and applied in both academic and industrial projects. A feature model represents a combination of system characteristics called features such that each combination corresponds to a member in a product family. A product family is a group of products that normally share common features. Moreover, it describes the commonalities, variabilities, and dependencies between features. The main challenges of feature modeling include the development, maintenance, and evolution of complex and large feature models. Specifically, it is difficult to handle unanticipated changes in large feature models [19, 57, 84]. A number of notable feature modeling methodologies and notations have been proposed and are described in detail below.

Feature Oriented Domain Analysis (FODA) was originally introduced by Kang et al. in 1990 [44]. FODA made an important contribution to the popularity of model-driven analysis by proposing to use several complementary perspectives of a domain to transfer complete information about that domain. This method shows the common functionality and architecture of applications in a domain that is related to requirements analysis and high-level design.

Pohjalainen [64] explained that although it is a good method, it is unclear how

the FODA expressions must be expanded to solve real world problems. Moreover, the main concern is how to express domain knowledge of particular configurations, such as default values, configuration implications, and constraints while maintaining the simplistic approach of the core FODA expression language.

Feature-Oriented Reuse Method (FORM), an extension of FODA, is based on commonalities and differences of applications in a domain in terms of *features* and employing the analysis results to develop domain architectures and components. A domain is described and analyzed in terms of common and different *units* of computation to generate various *feasible* configurations of reusable architectures [45]. They agree that FORM only provides language constructs to model architectures and integrate architectural components, and rigorous analysis of models can not be performed with the method.

Reuse-Driven Software Engineering Business (RSEB) is a systematic, model-driven aspect for large-scale software reuse introduced by Griss et al. [32]. RSEB is based on Jacobson's object oriented (OO) Software Engineering [39] and OO Business Engineering [37], applied to an organization working with building sets of related applications from sets of reusable components.

RSEB explains several model-driven software development processes including Architecture Family Engineering (develops a layered architecture), Component System Engineering (develops systems of reusable components), and Application System Engineering (develops selected applications). These processes optimize for robustness and reuse [33]; however, RSEB does not provide required domain analysis methods including domain scoping and feature modeling. In addition, it does not explain a systematic way to accomplish the asset development [38].

Featured Reuse-Driven Software Engineering Business (FeatuRSEB) is a combination of FODA and RSEB and a model-driven concept proposed by Griss et al. [33], with domain information taken from several different models showing different views of the domain. It has three extensions including that proposed by van Gurp et al. [77], Product Line Use case modeling for System and Software (PLUSS) [25], and that from Riebisch et al. [65].

Although FeatuRSEB provides a strong, and easy to use, way of characterizing the many hundreds of reuse-oriented features and use case variants [33], its features have

a binding time flag showing whether or not a variation point is bound at runtime. In addition, the FeatuRSEB method does not distinguish between various availability sites, i.e., when, where, and to whom a feature can be available.

Generative Programming (GP) is a software engineering paradigm introduced by Czarnecki et al. [18] based on modeling software system families. It can employ configuration information to automatically build highly customized and optimized intermediate and end-products from elementary reusable implementation components if it has a specific requirements specification. GP does not compete with the current paradigms, but instead supplements them [23].

Azimi and Hosseini believe it is not clear how GP works for concurrent and real-time systems in which there are a lot of delicate timing limitations. Such limitations are normally addressed by careful crafting the low-level facilities of the underlying hardware and operating system. In addition, the integration of manually and automatically created code could become too complicated. Therefore, this might be a particularly serious challenge in the maintenance phase in which new features are expected to be added to the system that modifies the system model, and then modifies the generated code that might be a mixture of automatically generated code and manually developed software [7].

Product Line Use case modeling for System and Software (PLUSS) proposed by Eriksson et al. [25] is a tool to visualize variants in abstract product family use case models. One proper use case model for the entire system family is kept and the feature model is employed as a tool for instantiating that abstract family model into concrete product use case models for each system built within the family.

One issue of the PLUSS feature modeling notation, compared to Czarnecki and Kim's more expressive Cardinality-Based notation [20], is the inability to model $n..m$ multiplicity. Moreover, for PLUSS to be successfully used, stronger configuration management and product planning functions are required.

Product Family Algebra (PFA) introduced by Hofner et al. [35] is a mathematical structure of product families. It focuses on the mathematical structure of an idempotent and commutative semiring. It aims capturing and analyzing the commonalities and variabilities of a product family. In addition, it allows mathematical description and manipulation of product family specifications.

Compared to other product family specification formalisms, such as FODA and FORM, there is a large body of theoretical results for idempotent commutative semiring, and for algebraic techniques in general, with strong impact for research related to problems of consistency, correctness, compatibility and reusability.

Cardinality-Based Feature Modeling (CBFM) proposed by Czarnecki and Kim is a hierarchical feature model integrates a number of extensions to the original FODA notation and each feature has cardinality [21]. CBFM is an extension of original FODA and is more expressive to describe commonality and variability in software product lines (SPL) with introducing cardinality concept in feature models [13].

Gomez and Ramos [30] showed the classical definition of configuration of a feature model is an issue since this definition tries to define the configuration as a copy mechanism instead of as an instantiating mechanism. This definition can be intuitive when working with traditional feature models; however it can be confusing when dealing with cardinality-based ones. Moreover, the second challenge is how to deal with model constraints when features can be cloned in the models and such features can have an attribute type.

PFA can help taking and analyzing the commonalities and variabilities of a product family and allows us to perform calculations on features, products, and product families that is not provided by other feature models. In addition, by using PFA, it is possible to build new families, find new products, and exclude unwanted combinations [3].

In this research, we employ PFA as a way to capture patterns of component reuse in populations, as described in Chapter 5.

2.5 Security Metrics

Security metrics is a complex topic, with books [41, 28] and surveys of different areas including systems security metrics [63], network security metrics [78], embedded security metrics [50] among others. Despite this variety, the core objective of employing security metrics within an organization is to establish a tangible and measurable way to assess the cybersecurity posture. This quantification allows organizations to gauge how secure their systems are, with a higher metric indicative of a robust defense mechanism making cyber-attacks arduous.

The large number of metrics arises from the numerous ways security can be quantified, capturing virtually any aspect of system configuration or history that could be related to security. Some measures are historical, such as how often vulnerabilities have been found, how long it took for vulnerabilities to be patched, and the time between patches being made available to being applied to a given system. Some are empirical, arising from lab or field studies where a defense system's ability to detect malware or intrusions are tested. Others are more theoretical, such as password strength or address space randomization entropy, where the construction of the system should give certain security properties, but its real-world security is impacted by many external factors. (Cryptographic security measures are virtually all theoretical.) No matter the type of metrics, however, each is only capturing a small aspect of system security.

2.6 Moving Target Defense (MTD)

Moving Target Defense (MTD) is an innovative cybersecurity strategy designed to increase attack difficulty by continuously changing the attack surface of a system [40]. Unlike traditional static defenses that rely on fixed configurations and predictable behaviors, MTD introduces variability and unpredictability, making it significantly harder for attackers to gain a foothold.

MTD operates by dynamically altering system configurations, such as IP addresses, network ports, software versions, or even code execution paths at unpredictable intervals. This continuous shifting of the defensive landscape disrupts the attacker's reconnaissance and exploitation phases, as previously gathered intelligence quickly becomes obsolete. By forcing attackers to adapt their strategies in real-time, MTD increases the time, effort, and resources required to launch successful attacks.

A common example of MTD is address space layout randomization (ASLR), which randomizes memory locations of key data areas in a program, complicating buffer overflow attacks. Another MTD technique is network address shuffling, where server IP addresses are periodically changed to disrupt targeted attacks. The primary benefit of MTD is that it reduces the window of opportunity for attackers, enhancing system resilience. However, implementing MTD can also introduce complexity in system management and potential performance overheads, necessitating careful planning and

execution.

Despite its advantages, current measures of MTD effectiveness often fail to capture the dynamic and time-sensitive nature of its security benefits. This study extends the understanding of MTD by proposing metrics that better reflect the evolving security landscape. For instance, while techniques like ASLR can temporarily disrupt attack strategies, they do not fully account for the adaptive capabilities of attackers who may eventually overcome these defenses. Cases have shown how MTD can be bypassed, such as targeted attacks adapting to randomized environments [74], highlighting the need for metrics that can dynamically assess MTD's security advantages over time, providing a more realistic evaluation of its long-term effectiveness.

2.7 Software Reuse

In the context of software engineering, reuse is a widely adopted practice for improving efficiency and reducing development costs [55]. However, this research addresses the critical gap in how traditional software engineering metrics fail to capture the security risks associated with software reuse. When components are reused across multiple systems, vulnerabilities embedded in those components can propagate, increasing the attack surface and risk exposure. Existing metrics often overlook these risks, focusing instead on performance or maintainability. This work introduces new metrics that specifically account for the hidden security liabilities of reused components, thereby providing a more comprehensive assessment of software security.

2.8 Software Diversity

Software diversity has been researched since the late 1970s [49]. A key development was N-version programming, a method where multiple functionally equivalent programs (the "N versions") are developed from the same initial specification. The idea is that, since the programs are developed independently, it is unlikely that they will share the same faults. Diversity has also been proposed as a strategy for reducing the impact of security vulnerabilities [27]. The idea is that if systems are different, then vulnerabilities will only impact a subset of systems at any time.

Many researchers have criticized software monocultures—environments with minimal software diversity—as being especially insecure, especially in the context of dominant operating systems [29]. Software monocultures, however, have significant benefits in terms of development time and administrative overhead, as fewer types of systems translates to fewer systems for administrators to learn how to configure, maintain, and secure.

Most past work in metrics for software diversity is focused on diversity from the perspective of fault tolerance [51]. Larsen (2014) noted that the field of automated software diversity needed better metrics [47]. Subsequent to this we know of two works that present diversity metrics for security, Zhang (2016) [83] and Tong (2019) [75]. Both works are inspired by diversity metrics used in ecology, unlike ours which is inspired by work in software engineering. The focus of Zhang's work is on network-level diversity using an attack graph-related approach; however, their work, although they also examine how to determine resource similarity using file and modification-level similarity [83]. In contrast, Tong's work presents an attribute matrix-based metric that captures variations such as differences in hardware, operating system, and applications [75].

Note that the metrics presented here are different from these past works in that 1) they are inspired by work in AI and software engineering, not biology, and 2) these metrics have explicit connections to security rather than simply being measurements of software and system diversity.

The motivation behind our study, as detailed in the following chapter, arises from the gaps. We aim to build upon the foundation set by our predecessors, delve deeper into specific areas that have been less scrutinized, and contribute novel insights to the existing body of knowledge.

Chapter 3

Motivation

In the rapidly evolving domain of cybersecurity, understanding the multifaceted nature of cyber threats and the strategies to mitigate them is crucial. This knowledge forms the bedrock upon which robust security architectures are constructed. Traditional security models often fall short in capturing the nuanced interactions between cyber attackers and defenders. Particularly, these models struggle in areas such as the recycling of knowledge among attackers and the processes involved in vulnerability discovery. This chapter underscores the need for a paradigm shift, advocating for a nuanced approach that examines the complexity of cyber attacks through the lens of knowledge reuse and advocates for a search-based framework to understand how attackers systematically probe for system weaknesses.

The dynamic landscape of cybersecurity is marked by a constant tug-of-war between cybercriminals and defenders. Attackers continuously evolve their strategies, leveraging sophisticated techniques and shared knowledge to identify and exploit system vulnerabilities. In contrast, defenders endeavor to anticipate and counteract these threats by fortifying their systems. This interaction underlines the importance of understanding the strategies and knowledge crucial to both launching and thwarting cyber attacks. By delving into these aspects, this discussion seeks to reveal strategic insights that can guide the development of stronger defenses, thereby making it more challenging for attackers to breach systems.

This foundational perspective sets the stage for a comprehensive exploration of the essential knowledge for conducting cyber attacks, highlighting the significance of epistemological considerations in software development, and the intricate relationship between vulnerabilities, exploits, and the broader cybersecurity landscape. It is through this lens that the chapter aims to foster a deeper understanding of the cyber threat ecosystem. This approach not only highlights the limitations of traditional defense models but also emphasizes the need for a more sophisticated and

knowledge-driven strategy in cybersecurity.

The concept of knowledge reuse among attackers is particularly noteworthy. It suggests that attackers do not always need to invent new techniques; instead, they often repurpose existing knowledge and tools to discover and exploit new vulnerabilities. This efficiency in knowledge recycling underscores the necessity for defenders to adopt proactive and innovative defense mechanisms that can adapt to the evolving tactics of attackers.

Furthermore, the proposition of a search-based paradigm for understanding the systematic exploration undertaken by attackers to find vulnerabilities introduces a new perspective in cybersecurity defense strategies. This model implies a more dynamic and iterative approach to security, where the focus is not only on defending against known threats but also on actively searching for potential vulnerabilities that could be exploited in the future.

This chapter advocates for a shift towards a more informed and adaptive approach to cybersecurity. By emphasizing the importance of understanding the complexities of cyber attacks through knowledge reuse and a systematic exploration of vulnerabilities, it sets the groundwork for developing more resilient and effective security infrastructures. This perspective not only challenges conventional security models but also encourages a deeper strategic engagement with the ongoing cyber warfare, with the ultimate goal of staying one step ahead of attackers.

3.1 Knowledge and Attacks

The capacity of an individual or group to launch a cyber attack is intrinsically linked to their knowledge and expertise in various technical domains [71]. In the context of this work, knowledge refers to the information and skills required to identify, exploit, and manipulate vulnerabilities within systems. This includes understanding specific vulnerabilities, such as memory management flaws, as well as broader techniques like social engineering, password cracking, or network intrusion methods. Knowledge encompasses technical details, procedural steps, and contextual information that collectively enable attackers to mount successful cyber attacks. In essence, knowledge is the intellectual and practical content that drives the actions of both attackers and defenders in the cybersecurity landscape.

Sophisticated cyber attacks, such as code injections, demand a comprehensive understanding of multiple aspects of computing and software engineering. For instance, executing a successful code injection attack requires not just familiarity with programming but a nuanced comprehension of CPU architecture to influence execution flows, an intimate knowledge of low-level operating system APIs to run arbitrary code, and advanced techniques to navigate around security measures like address space layout randomization (ASLR) and protections against executable space exploitation.

When delving into a specific example, such as a buffer overflow attack [24], the depth of knowledge required becomes even clearer. In these scenarios, attackers must first identify a buffer overflow vulnerability within the target application. This step alone requires a detailed understanding of the application's memory layout and the ability to recognize potential points of failure. Following this, crafting a payload that, upon execution, confers unauthorized access or control over the system involves sophisticated skills in manipulating stack or heap memory and utilizing shellcode or return-oriented programming (ROP) techniques.

It's important to highlight that while certain knowledge areas are specific to individual attack types, much of this expertise is transferable and can be applied across a range of cyber threats. Skills in bypassing ASLR, for example, or in executing shellcode, are invaluable across many forms of exploits [73, 6]. Mastery in these areas means that attackers can leverage their existing knowledge to facilitate new attacks, streamlining the process for launching future exploits. This ability to apply knowledge flexibly not only makes it easier for attackers to adapt but also increases the challenge for cybersecurity professionals tasked with defending against these threats.

This concept of knowledge reuse among attackers emphasizes the critical need for a dynamic and informed approach to cybersecurity defense. Understanding the breadth and depth of knowledge that attackers bring to their efforts can guide the development of more effective defense mechanisms. By anticipating the strategies and techniques that attackers might employ, cybersecurity professionals can better protect systems against a wide array of potential threats [43].

The relationship between knowledge reuse and attacks is fundamental in cybersecurity. Attackers often leverage existing knowledge—such as techniques, scripts, and known vulnerabilities—rather than inventing new methods for each attack. This

knowledge reuse significantly lowers the barriers to executing attacks, allowing even less skilled attackers to pose serious threats. For example, once a vulnerability like a buffer overflow is understood and exploited, the specific exploit knowledge can be reused across different systems with similar vulnerabilities. This reuse of knowledge enables rapid and widespread exploitation, amplifying the impact of attacks.

Examples of knowledge reuse in attacks are prevalent across various threat landscapes:

- Script Kiddies: These are individuals with limited technical skills who rely on pre-existing scripts or tools created by more knowledgeable attackers. Script kiddies exemplify knowledge reuse by using publicly available exploits to launch attacks, often without fully understanding the underlying vulnerabilities or methods.

- Worms: A classic example of knowledge reuse is seen in worms, such as the Morris Worm or WannaCry, which exploit known vulnerabilities to propagate autonomously. These worms reuse exploit code and strategies to infect multiple systems rapidly, demonstrating how prior knowledge is encoded into malware that can operate without human intervention.

- Advanced Persistent Threats (APTs): APT groups often use previously successful attack techniques across multiple campaigns. For instance, once an APT discovers a zero-day vulnerability, they may continue to exploit that vulnerability across different targets until it is patched, continuously reusing the same knowledge to maximize their impact.

These examples illustrate that knowledge reuse is not confined to a single level of sophistication; it is a strategic advantage that attackers at all levels exploit to optimize their efforts and extend their reach.

In designing cybersecurity defenses, it's crucial to consider not just the technical countermeasures but also the human element involved in these attacks [71]. Education and training in cybersecurity must, therefore, encompass a broad spectrum of knowledge, from the specifics of coding and system architecture to the psychological

and strategic aspects of cyber warfare. Only through a comprehensive and multi-faceted approach can the cybersecurity community hope to stay one step ahead of those seeking to exploit technological vulnerabilities for malicious purposes.

This detailed exploration into the attacker's knowledge base underscores the importance of continuous learning and adaptation in the field of cybersecurity. As attackers evolve and refine their techniques, so too must the defenses designed to thwart them. This ongoing cat-and-mouse game between attackers and defenders highlights the dynamic nature of cybersecurity and the constant need for innovation and vigilance in the digital age.

A theoretical approach is reasonable and necessary for this work because it allows us to abstract and generalize the dynamics of knowledge reuse in a way that is not limited to specific technologies or fleeting vulnerabilities. By modeling the attacker-defender interaction through formal frameworks, we can identify fundamental patterns and principles that apply across diverse contexts. This approach provides a structured way to analyze how knowledge reuse affects attack strategies and defensive postures, offering insights that are robust against the rapidly evolving nature of technology. Theoretical models serve as a foundation for developing practical tools and strategies that can adapt to new threats, guiding the design of resilient cybersecurity systems even as the landscape changes.

3.2 Epistemology and Software Development

Epistemology, or the philosophical study of knowledge, has a profound impact on software development and cybersecurity, shaping how practitioners understand, utilize, and protect information within digital systems [60]. In software development, epistemology is central to the process of creating, testing, and maintaining software. This field requires a deep understanding of abstract concepts such as algorithms, data structures, and system architectures. Developers must navigate the intricate landscape of existing codebases, anticipating how changes may affect overall system behavior. This task is inherently epistemic, as it involves the manipulation and application of knowledge in various forms.

The process of software testing serves as a clear example of applied epistemology in technology [56]. It resembles the scientific method, involving hypothesis formation

about software behavior under diverse conditions, followed by systematic experimentation to validate these hypotheses. Through unit testing, integration testing, stress testing, and other methods, developers engage in an epistemic cycle of theory, experimentation, and revision. This iterative process is crucial for ensuring software correctness, security, and performance, mirroring the broader scientific pursuit of knowledge through empirical inquiry.

In the domain of cybersecurity, the significance of epistemology is magnified. Security experts must grapple with the knowledge embedded within software systems as well as the tactics and strategies employed by attackers [54]. This dual focus requires a comprehensive understanding of both the knowledge that constitutes a system (e.g., functionalities, vulnerabilities) and the ways in which this knowledge can be leveraged, altered, or compromised by potential attackers. Cybersecurity thus emerges as a battleground over knowledge, with defenses designed around anticipatory knowledge of potential attacks and attackers continuously seeking to extend their understanding to circumvent these protections.

This epistemic perspective illuminates the continuous arms race between cybersecurity professionals and attackers. Defenders aim to build and maintain secure systems by applying their understanding of potential vulnerabilities and attack vectors, while attackers exploit gaps in this knowledge to breach defenses. The dynamic and ever-evolving nature of this contest underscores the importance of ongoing education and research in the field. For both sides, success is predicated on the ability to acquire, apply, and innovate upon knowledge, making epistemology a foundational aspect of cybersecurity [5].

Moreover, the epistemic challenges in software development and cybersecurity highlight the importance of methodologies and tools that can aid in the acquisition and application of knowledge [79]. From code analysis tools that help developers understand complex systems to advanced threat intelligence platforms that enable security professionals to predict and prevent attacks, technology plays a crucial role in mediating the epistemic activities of professionals in these fields.

The intersection of epistemology and technology, particularly in software development and cybersecurity, is a rich area of inquiry and practice [26]. By examining the ways in which knowledge is created, shared, and contested within these domains,

professionals can gain deeper insights into the complexities of digital systems and the ongoing efforts to secure them. This understanding not only enhances the effectiveness of technical solutions but also contributes to the broader discourse on the nature of knowledge and its role in the modern world.

3.3 Vulnerabilities and Exploits

Within the cybersecurity landscape, the understanding and management of vulnerabilities and their associated exploits are of paramount importance. A vulnerability represents a defect or weakness within a system that can be exploited by an attacker to undermine the system's integrity, availability, or confidentiality [73]. Conversely, an exploit is essentially a tool—whether it be software, data, or a series of commands—that utilizes a vulnerability to induce undesired system behavior, often resulting in unauthorized control over system resources or access to confidential information.

The journey of a vulnerability from discovery to resolution is complex and multifaceted. Initially, vulnerabilities may be unearthed by security researchers, malicious attackers, or inadvertently by end-users during routine usage [68]. Upon identification, the next step often involves the development of an exploit that can harness the identified vulnerability to launch attacks. This phase of exploit development is critical as it transforms theoretical vulnerabilities into practical tools for attackers. Subsequently, these exploits may be disclosed publicly, prompting vendors to create and disseminate patches or mitigation strategies. However, there exists a window of opportunity for attackers between the disclosure of an exploit and the widespread application of patches, during which the exploit can be used to target unpatched or slowly updated systems [67].

An illustrative example of the significant impact that knowledge of vulnerabilities and exploits can have in the cybersecurity realm is the Heartbleed bug. This critical vulnerability in the OpenSSL cryptography library enabled attackers to read the memory of affected servers [66]. This breach could reveal highly sensitive data, including private keys, user passwords, and personal information. OpenSSL's pivotal role in securing web communications amplified the consequences of Heartbleed, highlighting the crucial nature of timely knowledge regarding vulnerabilities and the swift

development of working exploits.

Heartbleed exemplified the urgency and necessity of a proactive approach in the cybersecurity community towards vulnerability management [9]. It underscored the ongoing arms race between those seeking to protect digital assets and those aiming to exploit vulnerabilities for malicious purposes. The incident also emphasized the importance of rapid response mechanisms, the value of open communication between vendors and the security community, and the critical need for regular system updates and patches by end-users.

The dynamics surrounding the discovery, exploitation, and mitigation of vulnerabilities underscore a fundamental aspect of cybersecurity: knowledge is power [76]. The timely identification of vulnerabilities, coupled with the swift development of effective exploits or patches, can significantly influence the security posture of digital systems. As such, continuous vigilance, research, and collaboration within the cybersecurity community are essential to defend against the ever-evolving threat landscape.

3.4 Modeling Knowledge Reuse Dynamics

In the ongoing battle between cybersecurity defenders and attackers, the strategic reuse and adaptation of knowledge is a central tactic. This conflict is characterized by a continuous cycle where both attackers and defenders leverage their cumulative knowledge—attackers to find new vulnerabilities and defenders to seal these breaches and fortify systems against future incursions. This iterative process can be conceptualized through the framework of search-based strategies, which encapsulates how attackers methodically explore a system's architecture for exploitable weaknesses—a structure initially crafted by defenders to be as impenetrable as possible [69].

Attackers, drawing upon their experiences and the collective intelligence of their communities, often recycle and refine tactics from past breaches. They employ a variety of probing techniques akin to a sophisticated search algorithm, navigating through the system's architecture in search of any vulnerability that can be leveraged [62]. Each point of weakness, once found, provides a potential entry point or method of attack. The evolution of these attack strategies is driven by a continuous feedback loop, where each attempt, whether successful or thwarted, contributes to the attacker's knowledge base and strategic approach.

Conversely, defenders are tasked with the complex challenge of modifying the search space—the theoretical landscape of potential attack vectors—by deploying security measures designed to obscure, relocate, or entirely eliminate vulnerabilities [5]. This not only makes the attacker's search more arduous but also requires a deep understanding of potential attack methodologies and the foresight to predict where new vulnerabilities may emerge.

The interplay between attackers' search strategies and defenders' architectural designs can be analyzed and modeled using AI and machine learning techniques [16]. These technologies offer a sophisticated means of simulating potential attack paths and identifying system vulnerabilities before they can be exploited. By applying machine learning models, defenders can gain valuable insights into likely attack vectors, enabling them to reinforce system defenses preemptively. This approach facilitates a more proactive defense posture, potentially staying one step ahead of attackers by anticipating their moves and strengthening vulnerabilities before they can be exploited.

This modeling and anticipatory strategy underscore a crucial aspect of modern cybersecurity: it is not enough to react to attacks as they occur. Instead, a predictive, knowledge-based approach is essential for maintaining robust security. By understanding the cyclical nature of knowledge reuse among attackers and leveraging advanced modeling techniques, cybersecurity professionals can better protect against the dynamic and ever-evolving threats posed by malicious actors. This proactive stance is critical in an era where the sophistication and frequency of cyberattacks continue to escalate, demanding equally sophisticated and dynamic defense mechanisms.

3.5 Search and Knowledge in AI

The concept of search AI offers a powerful framework for generating knowledge, especially in the domain of cybersecurity. AI search algorithms embark on a journey through extensive spaces of possibilities, aiming to unearth solutions to complex problems. This process mirrors the approach of attackers who meticulously explore various methods and strategies to identify vulnerabilities within systems.

The evolution of search in AI is highlighted by the development of sophisticated algorithms designed to navigate complex problem spaces with efficiency and accuracy.

Notable examples include the A* algorithm, which is renowned for its effectiveness in pathfinding and graph traversal problems, genetic algorithms that excel in solving optimization issues by mimicking the process of natural selection, and deep learning algorithms that have revolutionized pattern recognition through their ability to learn from large datasets [53].

These advancements illustrate a fundamental principle of AI: by methodically searching through a given space, it is possible to generate new knowledge and discover solutions that might not have been immediately apparent. This capability of AI to sift through vast amounts of data and identify patterns can be particularly beneficial in the context of cybersecurity.

In cybersecurity, the search space encompasses an array of potential attack vectors, system vulnerabilities, and techniques for exploitation. By leveraging AI-driven search techniques, cybersecurity experts can effectively simulate the exploratory processes employed by attackers. This proactive approach allows for the identification and mitigation of vulnerabilities before they can be exploited, enhancing the security of systems.

Moreover, AI-driven search methods in cybersecurity facilitate a more dynamic and adaptive defense mechanism. For instance, machine learning models can continuously learn from new data, improving their ability to predict and prevent future attacks [22]. This ongoing learning process ensures that cybersecurity defenses evolve in tandem with emerging threats, maintaining a strong security posture against an ever-changing landscape of vulnerabilities.

Furthermore, the integration of AI search techniques in cybersecurity tools enables automated and efficient scanning of networks and systems for vulnerabilities, reducing the time and resources required for manual testing. This automation not only accelerates the vulnerability identification process but also allows cybersecurity professionals to focus on developing and implementing robust defense strategies.

The application of AI search algorithms in cybersecurity represents a strategic advancement in the fight against cyber threats. By harnessing the power of AI to systematically explore and analyze potential security weaknesses, cybersecurity professionals can stay ahead of attackers, securing digital infrastructures against an array of vulnerabilities. This fusion of AI and cybersecurity underscores the critical role of

innovative technologies in safeguarding information and systems in the digital age.

3.6 Search and Security

In cybersecurity, the analogy of attackers as search algorithms provides a compelling framework for understanding the dynamics of cyber threats. Attackers systematically sift through the search space of a system's architecture, employing their accumulated knowledge and experience to uncover and exploit vulnerabilities. This methodical search is not random but targeted, with attackers directing their efforts towards areas most susceptible to compromise, thereby optimizing their chances of success.

On the flip side, defenders possess the capability to intricately alter this search space through the deployment of various security measures. Techniques such as encryption, which scrambles data into an unreadable format without a specific key; obfuscation, which deliberately makes code or system configurations confusing to interpret; and segmentation, which divides network resources into separate segments to contain potential breaches, collectively serve to complicate the attackers' search efforts. These strategies effectively increase the cost of the search for attackers, necessitating greater time, resources, and specialized knowledge to navigate and identify exploitable weaknesses.

The strategic interplay between attackers' search strategies and defenders' modifications to the search space underlines the importance of understanding the methods and motivations behind cyber attacks. By comprehensively analyzing the tactics employed by attackers, cybersecurity professionals can more accurately predict potential attack vectors and strengthen their defenses accordingly. This proactive and informed approach to cybersecurity leverages the principles of search and knowledge dynamics, granting defenders a strategic edge in safeguarding against cyber threats.

Moreover, the continuous evolution of both attack strategies and defense mechanisms highlights the cat-and-mouse nature of cybersecurity. As attackers refine their methods and discover new vulnerabilities, defenders must similarly advance their techniques and tools to protect against these emerging threats. This ongoing cycle necessitates a vigilant and adaptive security posture, emphasizing the critical role of continuous learning, threat intelligence sharing, and the implementation of advanced security technologies.

The conceptual framework of search in cybersecurity provides valuable insights into the tactics and countermeasures at play in the digital security landscape. By viewing attackers as sophisticated search algorithms and understanding the ways in which the search space can be manipulated by defenders, cybersecurity professionals can devise more effective strategies to thwart cyber threats. This understanding not only enhances the security of individual systems but also contributes to the broader goal of creating a safer and more resilient digital environment.

3.7 Formalizing Knowledge Reuse

To deepen our understanding of cybersecurity dynamics, the development and utilization of formal models for knowledge reuse is pivotal. These models offer an abstract yet precise representation of how knowledge, once acquired, can be repurposed and applied to novel contexts, such as varying attack scenarios or defense strategies. This abstract representation is crucial for encapsulating the essence of knowledge transferability within the cybersecurity domain.

One method to formalize knowledge reuse involves constructing models that delineate the relationships between distinct pieces of knowledge and their potential applications across diverse situations. For example, a model could illustrate the process by which understanding a particular exploit technique might be abstracted and then applied to exploit analogous vulnerabilities in disparate systems or software. This kind of modeling not only aids in generalizing specific knowledge for broader applications but also enhances the efficiency of both offensive and defensive cybersecurity strategies by leveraging past insights for future engagements.

Moreover, an essential facet of formalizing knowledge reuse is the development of models that encapsulate the progression of knowledge over time. This includes how attackers and defenders adapt and evolve through continuous learning from each cyber engagement. Such models are instrumental in forecasting the impact of newly acquired knowledge from a cyber attack or defensive maneuver on subsequent actions and strategies employed by both attackers and defenders. This predictive capability is vital for preempting adversary moves and fortifying cybersecurity measures.

Formalizing the reuse of knowledge transitions intuitive understandings into explicit, actionable insights. These insights, encapsulated within models, become invaluable resources for security analysts. They enable the prediction of attacker behaviors, the proactive identification of vulnerabilities, and the formulation of potent defense mechanisms, thereby elevating the strategic planning and response capabilities within cybersecurity operations.

The exploration into the dynamics of knowledge in cybersecurity, spanning from the basic prerequisites of cyber attacks to the sophisticated formal modeling of knowledge reuse, emphasizes the indispensable role of comprehending and anticipating adversary strategies. As the cybersecurity landscape perpetually shifts, propelled by technological progress and the creativity of attackers, possessing an in-depth understanding of these dynamics becomes increasingly crucial.

Leveraging AI and search-based models presents promising pathways to bolster our defensive posture, offering structured methodologies for vulnerability detection and defense strengthening. Embracing such innovative approaches and promoting collaboration within the cybersecurity community are key steps towards constructing more robust digital infrastructures. These infrastructures are better equipped to withstand the dynamic threats posed by constantly evolving adversaries.

In essence, the cybersecurity arms race transcends the technological realm to encompass an intellectual struggle where knowledge, its acquisition, refinement, and strategic application, form the core of cybersecurity endeavors. Moving forward, our effectiveness in protecting digital assets will depend on our vigilance, adaptability, and informed strategic foresight, ensuring we remain a step ahead in the ongoing dance between attackers and defenders.

In the upcoming chapter, we will delve into the formalization of exploring a domain of possibilities, commonly referred to as a search space. We will elucidate the systematic approach of navigating this space to uncover solutions that effectively challenge and invalidate the underlying assumptions. This detailed exploration will provide a comprehensive understanding of the problem-solving process in this context.

Chapter 4

Modelling Attack Difficulty Using Knowledge Reuse

In this chapter, we present a model that shows the dynamics of computer security are best understood as a pattern of knowledge reuse, one in which attackers develop knowledge of how to compromise systems. Most importantly, we believe our model can help us to understand why defenders continue to lose and what must change if we are to thwart attackers.

An attacker has to create knowledge in order to develop an exploit and attack a system, but how much knowledge has to be created and how we model the knowledge creation? In AI knowledge creation is modeled as search over an appropriate search space. Here, we are doing the same thing. Search leads to knowledge. We define **Attack** as search for any form of malicious action and **Attacker** as an agent who performs the search.

Attackers always build knowledge based on past knowledge like personal experience or some works someone else has been done. The more background they have the easier knowledge creation is. Knowledge reuse becomes a restriction on the search space. Search spaces are used in security most commonly in cryptography. In this research, we are generalizing this to apply to all security things by using more abstract search space.

We think it is an abstract search space because it has to represent all possible attacks. For example, for buffer overflow attack, an attacker has to have the search space of all possible address offset. Address space randomization does not increase the search space, but it reduces the ability to do knowledge reuse because an attacker cannot run it on one system and say I have the address and use it on next system.

4.1 Definitions

In this section, we begin by defining some terms that will be used in our model. We begin by understanding the foundational concept of an attack search space.

Imagine you have a house. The attack search space for a burglar would be all the possible entry points into your house—the front door, back door, windows, the chimney, etc. In the context of cybersecurity, if your house is a computer system or network, the attack search space would include vulnerabilities in the software, weak passwords, open ports, and so on.

The attack search space is a term used in cybersecurity to describe all the possible ways an attacker could attempt to compromise a system. It's essentially the universe of all potential avenues an attacker might explore to achieve their malicious intent.

Definition 4.1 (**Attack Search Space**)**.** *Let A be the attack search space. Each element $a \in A$ represents a possible attack vector. For instance, in our house analogy, a_1 might be the front door, a_2 the back door, and so on.The size of this search space is represented by $|A|$, which is the number of elements in the set A.*

$$A = \{a_1, a_2, ...\} \tag{4.1}$$

To better visualize the concept of the attack search space, let's consider a practical example.

Example 4.1 (Attack Search Space)**.** *Consider a simple login form. If an attacker wants to brute-force the password, the attack search space consists of all possible password combinations. If the password is a 4-digit PIN, then $|A| = 10^4 = 10,000$ possible combinations.*

Example 4.2 (Buffer Overflow Vulnerabilities)**.** *Consider a software application that takes user input without properly limiting its size. The attack search space for exploiting buffer overflow vulnerabilities in this application involves finding inputs that not only exceed the buffer's allocated space but also successfully execute arbitrary code. Unlike a finite set of PIN combinations, this space includes a vast range of inputs varying in length, content, and structure.*

Example 4.3 (Authentication Verification Errors)**.** *Consider a web application with multiple endpoints requiring user authentication. The attack search space related to authentication verification errors encompasses all possible ways an attacker might bypass authentication or escalate privileges without valid credentials. This space is*

abstract and multifaceted, involving various methods such as session hijacking, forging authentication tokens, exploiting logic flaws, and more.

The attack search space is a crucial concept in cybersecurity. By understanding the size and scope of this space, defenders can better prepare and protect systems, and attackers can determine the feasibility of their potential methods. The bigger the attack search space, the harder (and often longer) it generally is for an attacker to successfully compromise a system by brute force methods. However, attackers often look for ways to reduce this space by finding vulnerabilities or using other intelligent methods.

The mathematical assumptions underlying our model involve several key elements:

- Attack Search Space Assumption: We assume that the attack search space A encompasses all possible attack vectors, represented as discrete elements within the set. Each vector corresponds to a distinct combination of method, target, and conditions. This abstraction helps generalize attack strategies beyond specific technical details, allowing us to model the difficulty of attacks in a broad, systemic way.

- Independence of Vectors: The model assumes that each attack vector is independent, meaning the success of one vector does not directly affect the others. This independence simplifies calculations but also highlights a limitation—real-world scenarios might have interdependencies between vulnerabilities, which this model does not directly account for.

- Probabilistic Outcomes: Probabilities assigned to each vector reflect the likelihood of success given no prior knowledge, equating to a uniform distribution without additional intelligence guiding the attacker. Adjustments in probability due to knowledge reuse reflect a concentration of attack effort in known areas, thus reducing the effective size of the attack search space.

These assumptions provide a foundational framework but also present boundaries where real-world complexities might diverge from the modeled scenarios, particularly in cases where dependencies between vectors or evolving attacker strategies might play a significant role.

Building on the concept of the attack search space, we now delve into the specific pathways or methods an attacker might employ, termed as attack vectors.

Definition 4.2 (**Attack Vector**). *An Attack Vector is a representation of the way or approach an attacker uses to exploit a system. It encapsulates three main elements that are crucial to understanding the nature of the attack.*

Given an attack vector a:

$$a = (M, T, E) \tag{4.2}$$

Where

- *M represents the method or technique used by the attacker. It's the how of the attack.*
- *T denotes the target of the attack. It's the where or what of the attack.*
- *E encapsulates the conditions that need to be in place for the attack to be successful. These could be vulnerabilities in the system, user behaviors that can be exploited, or even external conditions like a natural disaster that an attacker is taking advantage of.*

To further illustrate this idea, let's examine a real-world scenario involving an SQL Injection attack.

Example 4.4 (Attack Vector). *In the provided example, we are looking at one of the most common web attack vectors: the SQL Injection attack.*

- *M: The method being used by the attacker is SQL Injection. This technique involves injecting malicious SQL code into input fields to manipulate or query the database in unintended ways.*
- *T: The target of this attack is the Web application's user login page. This means the attacker is specifically trying to exploit the login page of a web application, possibly to gain unauthorized access.*

- *E: The exploit conditions here are that the application doesn't sanitize user input, and the database directly processes the raw input. This is a common vulnerability in web applications where user inputs are not checked or sanitized for malicious content before being processed. If an application doesn't sanitize inputs, it might execute malicious SQL code provided by the attacker.*

For this particular attack vector, we can represent it as:

$a_{SQLinjection}$ = *(SQL Injection, User login page, No input sanitation)*

Understanding attack vectors is crucial in cybersecurity. It not only helps in identifying how an attacker might exploit a system but also in devising strategies to defend against such attacks. By breaking down an attack into its method, target, and exploit conditions, defenders can better understand the threat and work on specific countermeasures.

It is important to distinguish between different forms of attack vectors and how they are represented in our model. Attack vectors are defined by their unique combination of method, target, and conditions rather than minute variations in execution, such as different buffer overflow addresses. For instance, multiple buffer overflow exploits targeting different memory addresses are not considered distinct attack vectors in this context; rather, they represent variations within a single vector defined by the method (buffer overflow), the target (vulnerable application), and the exploit conditions (input size and content). This distinction ensures that the model accurately captures the strategic choices of attackers rather than inflating the search space with variations that do not fundamentally change the attack pathway.

Now that we've explored the specifics of attack methods, it's crucial to consider the knowledge attackers might possess about vulnerabilities, and how they might reuse this knowledge.

Definition 4.3 (**Attacker's Knowledge**)**.** *The attacker's knowledge about specific vulnerabilities refers to the information they have about potential weaknesses in the system. Let K be the set of vulnerabilities the attacker knows. Each vulnerability $k \in K$ represents a specific weakness in the system.*

Definition 4.4 (**Knowledge Reuse**)**.** *Reusing knowledge means leveraging insights or methods from past attacks for new attacks. Knowledge reuse can be formalized as a function R, which maps a known vulnerability to potential attack vectors in the search space.*

$$K = \{k_1, k_2, ...\} \tag{4.3}$$

$$R : K \to A \tag{4.4}$$

To contextualize the concept of knowledge reuse, let's envision a scenario where an attacker leverages vulnerabilities from past exploits.

Example 4.5 (Knowledge Reuse)**.** *Imagine a hacker who, in the past, exploited a vulnerability in System A. Now, they come across System B and recognize that it has the same vulnerability. Instead of starting from scratch, the hacker can reuse the knowledge from their past exploit on System A to quickly and efficiently attack System B.*

This is analogous to a locksmith who knows how to pick a specific type of lock. If they encounter the same lock type on a different door, they can use their prior knowledge to open it without having to figure out the mechanism all over again.

The idea of an attacker's knowledge and knowledge reuse underscores the importance of regularly updating systems and fixing known vulnerabilities. If attackers can reuse their methods from past exploits, it makes their job easier and faster. On the defense side, understanding these concepts helps in predicting potential threats and deploying appropriate countermeasures. If a vulnerability is known and fixed in one system, it's essential to ensure that other similar systems are also patched to prevent knowledge reuse by attackers.

With the understanding of how knowledge reuse functions, we now investigate its direct implications on the nature of the attack search space.

Proposition 4.1 (Reduction of Attack Search Space via Knowledge Reuse)**.** *Knowledge reuse reduces the size of the search space an attacker needs to explore.*

Proof. Let A denote the full attack search space and K represent the set of known vulnerabilities within this space. The application of knowledge reuse in identifying attack vectors reduces the search space from A to a smaller subset A', where $A' = R(K)$ and R maps known vulnerabilities to their respective attack vectors. This reduced search space A' has a size that is less than or equal to the size of A, formally expressed as $|A'| \leq |A|$. Moreover, the subset A', being informed by prior knowledge (K), possesses a higher density of exploitable vulnerabilities compared to the remaining portion of A not covered by K.

- Initial Assumption: Assume the full attack space A consists of all possible attack vectors, while K consists of known vulnerabilities. The mapping function $R : K \to A'$ translates these vulnerabilities into specific attack vectors, forming the reduced attack space A'.

- Reduction of Search Space: By definition, A' is constructed solely from the known vulnerabilities K, which implies A' is a subset of A. Thus, by construction, $|A'| \leq |A|$.

- Inside K-informed Space (A'): By focusing on K, attackers leverage historical data and known vulnerabilities, which are inherently more likely to be exploitable due to their established nature. Hence, the density of viable attack vectors within A' is higher, making the search more efficient and likely to yield fruitful results.

- Outside K-informed Space: While vulnerabilities can indeed exist outside of K, the absence of prior knowledge or evidence suggesting their exploitability means that the search within $A \setminus A'$ (the portion of A not included in A') is more akin to searching in the dark. The probability of discovering a new vulnerability in this area is lower, given the lack of targeted direction, making this effort less efficient and more time-consuming.

- Probabilistic Advantage: The choice to operate within A' is not just about reducing the search space but also about maximizing the probability of finding exploitable vulnerabilities. The knowledge-driven approach inherently concentrates efforts where success is more likely, thus optimizing the search process.

□

Therefore, knowledge reuse effectively reduces the attack search space to a more manageable and potentially more fruitful subset A', where the efficiency of identifying new vulnerabilities is enhanced due to the higher density of known vulnerabilities. This strategic narrowing not only makes the attacker's job quicker but also more efficient, underlining the proposition's premise.

To further emphasize the effects of knowledge reuse on the attack search space, let's consider an illustrative example.

Example 4.6 (Impacts of Knowledge Reuse)**.** *Imagine an attacker has previously exploited 5 different systems. From these exploits, they've learned about 5 vulnerabilities, each corresponding to a specific attack vector. When faced with a new system, instead of considering every possible method of attack (the full search space A), they can narrow their focus to just these 5 known vulnerabilities.*

So, even if the total number of potential attack vectors in the full search space A is, say, 1000, by relying on their previous knowledge, the attacker reduces their search space to just 5. This is a drastic reduction and showcases the efficiency gained through knowledge reuse.

The concept of knowledge reuse and its impact on the attack search space emphasizes the dynamic nature of cybersecurity. As attackers gain more experience and knowledge, they can become more efficient in their malicious endeavors. For defenders, understanding this dynamic is crucial.

Having established the foundational concepts of the attack search space and the role of attacker's knowledge, we now transition to understanding the mathematical relationships governing these concepts, starting with the expected value of success in a scenario where an attacker has no prior knowledge.

Proposition 4.2 (Expected Success Rate Without Prior Knowledge)**.** *If an attacker selects vectors at random without any prior knowledge of their success rates, the expected success rate for a single random choice is* $\frac{1}{n}$.

Proof. Consider an attack scenario with n distinct attack vectors. Let $S(a_i)$ denote the success of selecting attack vector a_i, with $S(a_i) = 1$ if the attack is successful,

and $S(a_i) = 0$ otherwise. The probability of any attack vector a_i being successful, in the absence of any prior knowledge, is uniformly distributed, such that $P(a_i) = \frac{1}{n}$ for all $i \in 1, 2, ..., n$.

- Expected Value Calculation: The expected value of success, denoted as $E(Success|K = \emptyset)$, can be calculated as the sum of the probabilities of success for all attack vectors multiplied by their respective success values:

$$E(Success|K = \emptyset) = \sum_{i=1}^{n} P(a_i)S(a_i) \tag{4.5}$$

Given that $S(a_i) = 1$ for a successful attack and 0 for failure, and the probability of any vector being successful is $\frac{1}{n}$, the formula simplifies to:

$$E(Success|K = \emptyset) = \frac{1}{n} \times 1 + (n-1)(\frac{1}{n} \times 0) = \frac{1}{n} \tag{4.6}$$

Simplifying further, we find that:

$$E(Success|K = \emptyset) = \frac{1}{n} \tag{4.7}$$

□

Thus, it is proven that without any prior knowledge ($K = \emptyset$) to inform their decisions, an attacker's expected value of success when randomly choosing from n attack vectors is $\frac{1}{n}$, highlighting the challenges faced when attempting to succeed without targeted knowledge or strategies in a security context.

The Expected Value of Success without Knowledge quantifies the likelihood of an attacker's success when they're just guessing. This concept underscores the importance of knowledge and strategy in both offensive and defensive cyber operations. The more an attacker knows, the more they can refine their approach and increase their chances of success. Conversely, the more defenders can obfuscate, vary, and secure their systems, the closer they can push an attacker's chances to this random guessing scenario, making breaches less likely.

When an attacker possesses knowledge about the system vulnerabilities, the expected success rate changes. Let's delve deeper into this scenario.

Proposition 4.3 (Expected Success Rate with Knowledge Reuse)**.** *If an attacker knows one vector has a higher success rate than the uniform rate of the others, the expected success rate is:*

$$E_{a_j}(Success|K \neq \emptyset) = P(a_j) + (1 - P(a_j)) \times E(Success|K = \emptyset) \tag{4.8}$$

where $E(Success|K = \emptyset)$ is the expected value of success without knowledge, as defined previously.

Proof. Assume there are multiple attack vectors available to an attacker, among which one specific vector a_j has a known higher success rate, $P(a_j)$, compared to the others. The rest of the vectors share the baseline success rate $E(Success|K = \emptyset)$. Given that the attacker has knowledge about a_j's higher probability of success:

- The attacker uses a_j with probability $P(a_j)$. If a_j is successful, the attack succeeds.
- If a_j fails, which occurs with probability $1 - P(a_j)$, then the attacker resorts to using other vectors, and the success rate for these vectors is $E(Success|K = \emptyset)$, the average success rate without specific knowledge.
- The total expected success rate when choosing to start with a_j can be modeled as the sum of the success probability of a_j itself and the probability that a_j fails multiplied by the average success rate of the remaining attack vectors. Formally, this is written as:

$$E_{a_j}(Success|K \neq \emptyset) = P(a_j) + (1 - P(a_j)) \times E(Success|K = \emptyset) \tag{4.9}$$

□

This reflects the direct impact of choosing a_j based on its known higher success rate. The mention of reverting to random selection would only apply if considering

multiple attempts beyond the initial, informed choice of a_j, which falls outside the primary scope of enhancing expected success through specific knowledge.

Imagine you're playing a game of darts. Without any knowledge, you'd just throw darts randomly, hoping to hit the bullseye (which has a success rate of $\frac{1}{n}$, where n is the total number of sections on the dartboard).

Now, let's say someone gives you a tip that aiming at a specific section increases your chances of hitting the bullseye. Armed with this knowledge, you'd focus on that section, significantly increasing your success rate. This new success rate for that section is $P(a_j)$.

However, if for some reason you decide to throw darts randomly at other sections (not the one you have knowledge about), your success rate for those sections would be the original $\frac{1}{n}$.

Combining the two strategies, your overall success rate becomes the sum of the success rate from the tip (knowledge) and the success rate from random throws.

The Expected Value of Success with Knowledge highlights the power of information. When an attacker has specific knowledge about a system, they can significantly boost their chances of success. This is why, in cybersecurity, keeping vulnerabilities secret or undisclosed can be dangerous. Once knowledge of a vulnerability becomes public, it can be exploited by attackers who now have an increased expected success rate.

In many scenarios, it is unlikely for an attacker to exhaustively search the entire attack search space. Attackers often stop as soon as they find a successful vector. For this, we need to consider the cumulative probabilities of finding at least one successful vector as we go through n trials.

Proposition 4.4 (Expected Trials to Success Without Prior Knowledge)**.** *If an attacker selects vectors randomly and without replacement, stopping upon finding a successful vector, the expected number of trials before finding a successful vector is approximately* $\frac{n}{2}$.

Proof. Assume an attacker operates within a search space of n unique attack vectors, with each vector having an equal and independent chance of being successful. The process of selection is random and without replacement, meaning each vector is only

tried once, and the sequence of selections does not influence the probability of success for the remaining vectors.

- Average Effort Calculation: Since the attacker stops after finding a successful vector, we can model this scenario as a uniform distribution of success across all attempts, with the assumption that any vector has an equal chance of being the successful one. This situation can be likened to the process of searching for a specific item in a list of n items where the order is unknown.

- Probability of Success on a Given Trial: On any given trial, the probability of success is evenly distributed across all remaining vectors. Therefore, the chance of finding a successful vector on the first trial is $\frac{1}{n}$, on the second trial (after one failure) is $\frac{1}{n-1}$, and so on, until a success is achieved.

- Expected Number of Trials: The expected number of trials before success can be viewed as the average position of a successful vector in a randomly ordered list of n vectors. Given the symmetry of the situation, with no vector being more likely to be successful than any other from the outset, the expected position is the middle of the list, yielding:

$$E(T) = \frac{n+1}{2} \tag{4.10}$$

  However, for large n, the difference between $\frac{n}{2}$ and $\frac{n+1}{2}$ becomes negligible, allowing us to approximate $E(T)$ as $\frac{n}{2}$.

□

Thus, in the absence of specific knowledge guiding the selection of attack vectors, an attacker randomly selecting attack vectors without replacement will, on average, find a successful vector after approximately $\frac{n}{2}$ trials. This model assumes a uniform distribution of success across the search space and reflects a realistic stopping condition where the search ends upon finding a success.

Imagine you're searching for a particular card in a deck. If you have no clue where that card is, you'd probably expect to find it around the middle of your search (after looking through about half the deck). Similarly, in a cybersecurity context, if an

attacker is blindly looking for a way to exploit a system, they might, on average, find it after trying about half of the possible attack vectors.

The Average Number of Trials with a Stop Condition offers a more practical viewpoint on an attacker's behavior. Instead of exhaustively trying every possibility, they might cease their efforts once they achieve their goal. For cybersecurity professionals, understanding this behavior can help in risk assessment and in deploying defenses more strategically. If defenses can be structured such that commonly tried vectors (those the attacker might try earlier in their efforts) are well-protected, it increases the chances that the attacker might give up before finding a vulnerability.

We delved into the concept of the Attack Search Space, which encapsulates all possible avenues an attacker might explore to compromise a system. We then touched upon Attack Vectors and how, when combined with prior knowledge, these can significantly narrow down this search space. The idea of an attacker leveraging prior knowledge, either to efficiently exploit known vulnerabilities or to predict the probable success of an attack, underscores the evolving nature of cyber threats. Furthermore, the examination of expected efforts, both with and without knowledge, gives a nuanced perspective on the realistic behavior of attackers, emphasizing that they often operate based on efficiency and probability rather than exhaustive efforts.

With a firm grasp on these foundational concepts, it's crucial to measure and quantify the security posture of systems. The next section will introduce two new security metrics. These metrics will serve as tangible indicators, providing insights into a system's vulnerability and the efficacy of its defenses. By quantifying security, we can better strategize, prioritize, and allocate resources to fortify our digital fortresses.

4.2 Two Security Metrics

Navigating the intricate landscape of cybersecurity necessitates not just robust defense strategies but also precise measures to gauge their efficacy. To rise to this challenge, we must introduce metrics that encapsulate the nuances of our defense mechanisms, offering tangible insights and benchmarks. In this section, we will elucidate two pivotal security metrics that stand at the forefront of evaluating a system's resilience against potential threats.

4.2.1 Knowledge Obfuscation Security Metric (KOSM)

The Knowledge obfuscation security metric (KOSM) is designed to assess how well a system impedes attackers from effectively reapplying previously acquired knowledge. KOSM quantifies the effectiveness of a system's ability to obscure or distort previously acquired knowledge by potential attackers. By making previously gained knowledge unreliable or irrelevant, KOSM aims to deter attackers from reusing known strategies or insights. A higher KOSM indicates that the system is better equipped at rendering previously acquired attacker knowledge obsolete, thus increasing the cost and effort required for an attack. This metric shines in two primary facets:

- Data Distortion: This facet focuses on intentionally modifying data to render it meaningless or misleading without changing its original format or structure. It's about preserving the appearance of authenticity while removing or altering the actual substance. For example, consider encrypted data. While it might look genuine, without the correct decryption key, the content is incomprehensible. Another example is data masking, where sensitive parts of data (like Social Security Numbers) are replaced with placeholder characters, preserving the format but obscuring the true value.

- System Behavior Randomization: This facet emphasizes changing the behavior and responses of a system over time or per interaction to ensure that attackers can't reliably predict how the system will react. By adding unpredictability to the system's operations, attackers can't effectively leverage their previous experiences or insights. For instance, in a web application, the server might randomly switch between different error messages for the same error condition. An attacker trying to exploit a vulnerability by observing error messages would find it challenging to determine the exact cause of an error due to the randomized responses. Another instance is changing API endpoints or URL structures, making previously known paths invalid.

These two facets, when combined, offer a comprehensive approach to obfuscating knowledge, making it challenging for attackers to apply prior insights or experiences to current or future attack attempts.

With this conceptual backdrop in place, we can now detail the KOSM with a precise mathematical characterization.

Definition 4.5 (**KOSM**). *Let's consider:*

- *K: The set of knowledge an attacker initially possesses about the system.*
- *K': The perceived knowledge by the attacker after obfuscation strategies are applied.*
- *O: The obfuscation function that transforms K into K'.*

For the sake of quantification, let's associate a value with knowledge:

- *Value V(K): The utility or effectiveness of the initial knowledge set K in terms of aiding an attack.*
- *Value V(K'): The utility or effectiveness of the obfuscated knowledge set K'.*

The effectiveness of KOSM can be represented by the reduction in knowledge utility:

$$\Delta V = V(K) - V(K') \tag{4.11}$$

Where:

- *A higher ΔV indicates a more effective knowledge obfuscation, implying that the system's security is enhanced by the obfuscation strategies.*
- *A ΔV close to zero would imply that the obfuscation strategies are not significantly impacting the utility of the attacker's knowledge, suggesting a need for stronger obfuscation measures.*

The KOSM metric is then defined as:

$$KOSM = \frac{\Delta V}{V(K)} \tag{4.12}$$

Where:

- *KOSM = 1 implies maximum obfuscation, rendering the attacker's prior knowledge completely useless.*
- *KOSM = 0 indicates no obfuscation, meaning the attacker's knowledge remains fully effective.*

In essence, the KOSM metric quantifies the security of a system by measuring the reduction in the utility of attacker knowledge due to obfuscation strategies. A system with a high KOSM value effectively neutralizes the advantage an attacker would have from their prior knowledge.

Example 4.7 (KOSM)**.** *Imagine an e-commerce web application that has suffered past breaches. Attackers, in their past exploits, have gained knowledge about the application's database structure, particularly the user table which contains sensitive customer information. The attackers know the table name (users) and the field names (username, password, email, address).*

Initial State: *In the initial state, the attackers have knowledge K about the database structure, which has a certain utility value, V(K).*

Implementation of Knowledge Obfuscation:

- *Database Table and Field Name Randomization: Every month, the application randomizes the names of database tables and fields. The table users might be renamed to clients one month, customers the next month, and so on. Similarly, the field username might be renamed to user_id, user_name, etc.*
- *Input Field Obfuscation: Input fields in forms are randomized in their naming. So, a field that was previously named username might now be user_id or user_name_entry. Attackers who have crafted scripts to exploit the username field might find their scripts failing, as the field names have changed.*
- *Introduction of Dummy Fields: The application introduces dummy fields like user_ref, user_code which don't hold any real data but are designed to mislead attackers.*

After implementing these obfuscation measures, the attacker's knowledge is transformed. Their initial knowledge K becomes K', which is now either partially incorrect or incomplete. The utility of this new knowledge set, V(K'), is reduced.

Measuring KOSM:

Suppose initially, with knowledge K, the attacker had a 90% chance of successfully querying the user table. This gives V(K)=0.9. After obfuscation, due to incorrect table and field names and the confusion introduced by dummy fields, suppose the attacker's success rate drops to 30%. This implies V(K')=0.3. The reduction in knowledge utility is:

$$\Delta V = V(K) - V(K') = 0.9 - 0.3 = 0.6$$

Thus, the KOSM metric is:

$$KOSM = \frac{\Delta V}{V(K)} = \frac{0.6}{0.9} \approx 0.67$$

A KOSM value of 0.67 indicates that the obfuscation strategies have effectively reduced the utility of the attacker's knowledge by 67%. The higher this percentage, the more effective the obfuscation measures are in enhancing the system's security.

4.2.2 Defense Evolution Security Metric (DESM)

The Dynamic Defense Evolution Security Metric (DESM) quantifies a system's ability to dynamically adapt and evolve its defenses in response to potential threats. A higher DESM value indicates the system's proficiency in altering its defenses to stay ahead of attackers. This continuous evolution ensures that the system's defenses remain effective against the ever-evolving landscape of attacks. Primary facets of DESM include:

- Real-time Adaptability: The system's ability to adjust its defenses instantly based on detected threats or vulnerabilities. For example, upon detecting an abnormal surge in traffic, a system dynamically adjusts its firewall rules or deploys additional resources to mitigate a potential DDoS attack.

- Predictive Evolution: The system's capacity to anticipate potential future threats and adjust its defenses accordingly. For instance, leveraging AI and machine learning to analyze historical data and predict future attack patterns, then proactively adjusting defenses before these attacks materialize.

To further elucidate the concept of DESM, let's delve into its formal definition.

Definition 4.6 (**DESM**). *Let's consider:*

- *A: The initial set of attack vectors an attacker might employ.*
- *A': The set of attack vectors after the system has evolved its defenses.*
- *D: The dynamic defense function that transforms A into A'.*

For quantification, let's associate a success rate with attack vectors:

- *Success Rate S(A): The probability of a successful attack using vectors in A.*
- *Success Rate S(A'): The probability of a successful attack using vectors in A' after defensive evolution.*

The effectiveness of DESM is represented by the reduction in attack success rate:

$$\Delta S = S(A) - S(A') \tag{4.13}$$

Where:

- *A higher ΔS indicates a more effective defense evolution, implying enhanced system security.*
- *A ΔS close to zero suggests that the dynamic defenses have not significantly impacted the effectiveness of the potential attack vectors.*

The DESM metric is then defined as:

$$DESM = \frac{\Delta S}{S(A)} \tag{4.14}$$

Where:

- *DESM = 1 implies that the defenses have evolved to render all prior attack vectors completely ineffective.*

- *DESM = 0 indicates no change in the effectiveness of the attack vectors, suggesting no significant defense evolution.*

Example 4.8 (DESM)**.** *Consider a cloud-based web service initially vulnerable to certain known attack vectors, such as specific DDoS attack patterns or SQL injection techniques.*

Initial State:

With the attack vector set A, attackers have a certain success rate, S(A).

Implementation of DESM:

- *Real-time Traffic Analysis and Response: The service uses real-time traffic monitoring. When it detects patterns that resemble a DDoS attack, it dynamically reroutes traffic, deploys additional resources, or introduces rate limiting for suspicious IP addresses.*

- *Predictive Database Security: The service employs machine learning to analyze historical database queries. Over time, it learns to identify potentially harmful query patterns and proactively adjusts its database firewall rules to block or challenge these queries even before an actual attack.*

After these measures, the effective attack vector set becomes A'. The new success rate, taking into account the evolved defenses, is S(A').

Measuring DESM:

Let's say initially, with the attack vector set A, attackers had a 70% success rate. This gives S(A)=0.7.

After the dynamic defense measures, the success rate drops to 20%, implying S(A')=0.2.

The reduction in success rate is:

$$\Delta S = S(A) - S(A') = 0.7 - 0.2 = 0.5$$

Thus, the DESM metric is:

$$DESM = \frac{\Delta S}{S(A)} = \frac{0.5}{0.7} \approx 0.71$$

A DESM value of 0.71 indicates that the dynamic defense evolution strategies have effectively reduced the success rate of potential attacks by 71%. The higher this percentage, the more effective the system's dynamic defenses are in mitigating potential threats.

KOSM and DESM are pivotal metrics in the realm of cybersecurity, embodying adaptive and proactive defensive strategies. KOSM focuses on the deliberate obfuscation of genuine data and system behaviors, rendering any previously acquired knowledge by attackers as unreliable or obsolete. By manipulating the perceived landscape, KOSM ensures that attackers cannot comfortably rely on prior insights, forcing them into a state of continuous discovery and adjustment. This not only impedes their progress but also escalates their operational costs and efforts.

DESM, on the other hand, emphasizes the system's ability to evolve and adapt its defenses dynamically. Rather than solely relying on static defensive postures, DESM encapsulates the system's agility in responding to emerging threats, its ability to predict potential vulnerabilities, and its capacity to continually fortify itself against a shifting threat landscape. By staying one step ahead of potential attackers and ensuring that defenses are not only reactive but also anticipatory, DESM showcases a system's resilience against the ever-evolving world of cyber threats. Together, KOSM and DESM represent a holistic approach to security, marrying obfuscation with evolution to ensure robust protection in an unpredictable digital age.

Having delved deeply into the foundational concepts of KOSM and DESM, we stand at the precipice of a more intricate exploration. Theoretical underpinnings, while abstract, provide the robust backbone upon which practical applications lean. As we transition into the upcoming section, we'll immerse ourselves in the rigorous world of theorems and proofs. These mathematical constructs not only validate the principles we've discussed but also pave the way for a comprehensive understanding of their implications. Armed with the knowledge from our previous discussions, let's navigate the mathematical intricacies that solidify the concepts of knowledge obfuscation and dynamic defense evolution.

Proposition 4.5 (Reduction in Successful Attacks with Increased KOSM)**.** *For a system with applied KOSM, the expected number of successful attacks decreases as KOSM increases.*

Following our theorem's proposition, we now present a proof, elucidating the conditions under which one strategy outshines the other.

Proof. Let's denote:

- $E_{initial}$: Expected number of successful attacks based on initial knowledge, K.
- $E_{obfuscated}$: Expected number of successful attacks based on obfuscated knowledge, K'.

From our earlier definition of KOSM:
$KOSM = \frac{\Delta V}{V(K)}$
Where:
$\Delta V = V(K) - V(K')$
Given that V(K) represents the utility or effectiveness of the knowledge set K for an attack, a higher KOSM value implies a greater reduction in knowledge utility.

Thus, as KOSM increases, the difference between $E_{initial}$ and $E_{obfuscated}$ becomes more pronounced:
$E_{initial} - E_{obfuscated} \propto KOSM$
Hence, $E_{obfuscated} < E_{initial}$ as KOSM increases, proving the theorem.

□

Proposition 4.6 (Increase in Time to Successful Attack with Increased DESM)**.** *For a system with applied DESM, the time to a successful attack increases as DESM increases.*

Following our theorem's proposition, we now present a proof, elucidating the conditions under which one strategy outshines the other.

Proof. Let's denote:

- $T_{initial}$: Time taken for a successful attack based on initial attack vectors, A.
- $T_{evolved}$: Time taken for a successful attack based on evolved attack vectors, A'.

From our earlier definition of DESM:
$DESM = \frac{\Delta S}{S(A)}$

Where:

$\Delta S = S(A) - S(A')$

Given that S(A) represents the success rate of an attack using vectors in A, a higher DESM value implies a greater reduction in attack success rate.

Now, if the success rate of an attack decreases, it implies that an attacker would need to invest more time to achieve a successful breach.

Thus, as DESM increases, the difference between $T_{evolved}$ and $T_{initial}$ becomes more pronounced:

$T_{evolved} - T_{initial} \propto DESM$

Hence, $T_{evolved} > T_{initial}$ as DESM increases, proving the theorem.

□

These propositions highlight the practical benefits of implementing KOSM and DESM strategies. By obfuscating knowledge and evolving defenses, systems can significantly reduce the likelihood of successful attacks and increase the time and resources attackers must invest to breach them.

While KOSM and DESM both serve to enhance a system's security posture, they address different aspects of defensive strategy:

- KOSM focuses on disrupting the attacker's ability to reuse prior knowledge. It measures how well a system makes previously acquired attacker knowledge irrelevant through obfuscation techniques such as data distortion and behavior randomization. KOSM's primary goal is to confuse and mislead attackers, reducing their efficiency by making past insights unreliable.

- DESM, on the other hand, measures a system's capacity to adapt its defenses dynamically in response to emerging threats. DESM reflects the system's agility and predictive capability in altering its defensive strategies, thus staying ahead of attackers. Unlike KOSM, which disrupts knowledge reuse, DESM emphasizes continual adaptation and evolution of defenses to proactively counteract evolving attack methods.

In essence, KOSM focuses on rendering past knowledge obsolete, while DESM ensures that defenses do not remain static but evolve in line with changing threat

landscapes. Together, they form a complementary approach to robust security, addressing both the historical and future challenges of cyber defense.

In this chapter, we delved deep into the paradigm of understanding cyber-attacks through the lens of knowledge reuse. We proposed that the recurring dynamics of computer security incidents emanate from the patterns in which attackers recycle their knowledge of system compromises. Through our model, we aimed to shed light on the crucial question: Why do defenders often fall short against attackers? By juxtaposing attack dynamics to knowledge creation in AI, we emphasized the attacker's journey as a search operation within a predefined space. This abstract search space captures the myriad ways an attacker could potentially exploit a system. Our study illuminated that attackers don't operate in vacuums; their actions often stem from prior experiences and previously acquired knowledge.

In our exploration of modern defensive metrics, KOSM and DESM emerge as pivotal tools in the ever-evolving field of cybersecurity. The intricate dance between attackers and defenders is one of continual adaptation, and the metrics we've delved into offer a structured approach to staying ahead. KOSM emphasizes the power of obfuscation, ensuring that the attacker's gathered intelligence is continually rendered ineffective or misleading. On the other hand, DESM champions the dynamic evolution of defenses, ensuring that systems are not static targets but ever-shifting enigmas, difficult to decipher and penetrate.

As we conclude this chapter, it's evident that the cyber landscape demands more than just reactive measures. The future of cybersecurity lies in proactive, adaptive, and continuously evolving strategies. The formalisms, theorems, and practical implications discussed herein underscore the importance of these metrics in real-world scenarios. They serve as a testament to the need for innovative thinking and a forward-looking approach in the face of ever-increasing cyber threats. As technology progresses and attackers devise new strategies, KOSM and DESM stand as beacons, guiding defenders towards a more secure and resilient digital future.

In the forthcoming chapter, we will delve deeper into the concept of knowledge reuse, specifically examining its manifestation in the form of shared software components. This exploration aims to provide a structured and formalized perspective on how knowledge, particularly in the realm of software development, is perpetuated

and shared across various systems and platforms. By understanding the intricacies of shared components, we can gain valuable insights into the vulnerabilities that may arise and the potential implications for system security. This chapter will elucidate these complexities, offering readers a comprehensive overview of how shared software components play a pivotal role in the broader context of knowledge reuse.

While this chapter focused on understanding the complexities and challenges associated with modeling attack difficulty, specifically leveraging the power of knowledge reuse and the nuances of various defenses, it becomes imperative to explore more comprehensive methods to understand and measure security. As our digital landscape grows, the diversity of systems and their configurations also expands, leading to an intricate web of potential vulnerabilities and security measures. This sets the stage for our next exploration—employing the concept of product family algebra. This mathematical tool will aid in providing a more granular understanding of security measures across a diverse range of system configurations.

Chapter 5

A Case Study Using Product Family Algebra

In Chapter 4, we talked about the attack search space, which means all the different ways someone might try to attack a computer system. We also saw how using the same knowledge over and over can change the number of these attack ways. We found out that if attackers choose their methods randomly, it's harder for them to be successful because there are so many options.

Now, in Chapter 5, we're going to look at this idea from a different angle. We're not just thinking about the number of attack methods, but also about how many different ways a computer system can be set up. Each setup can have its own mix of features and settings.

Think of it like a lock and keys. Just like a lock can have many key combinations, a computer system can be set up in lots of different ways. Each setup is like a different lock, with its own weak points and strong points. This makes it really hard to guess where the system might be weak, because there are so many different setups.

We will also talk about how computer systems are always changing. They get updates, fixes, and new settings all the time. This means that the way a system is set up can keep changing, making it even harder to guess how to attack it. It's like the lock is always changing, so finding the right key gets even more difficult.

Chapter 5, is all about understanding that computer systems can be set up in many different ways, and each way has its own security challenges. This chapter builds on what we learned in Chapter 4, but goes deeper into how the different setups of a system can make it hard to attack. It's important for people who work in cybersecurity to keep up with these changes and challenges to protect the systems better.

5.1 Intuition Behind Using Product Family Algebra

In the realm of cybersecurity, understanding the knowledge dependencies within a system is vital for assessing its vulnerability to attacks. Just as different types of knowledge form the building blocks of understanding a subject, various components within a system constitute the fundamental elements of its structure.

Product family algebra provides a powerful framework for analyzing dependencies and relationships among components in a system. By categorizing components into product families, which represent cohesive groups of interconnected elements, we can gain valuable insights into the dependencies and potential attack vectors present within the system.

The core intuition behind using product family algebra lies in its ability to capture the intricate web of knowledge dependencies and unveil the critical components an attacker must compromise to exploit vulnerabilities effectively. Just as understanding the core concepts and foundational knowledge of a subject is essential for grasping its complexities, identifying the central components and their dependencies is crucial for comprehending the potential avenues an attacker may exploit.

In many systems, dependencies form a chain-like structure, where one component relies on the availability or functionality of another component. This chain of dependencies can create a ripple effect, wherein compromising a single component may enable an attacker to exploit a series of interconnected vulnerabilities.

Product family algebra provides a systematic approach to unraveling these chain dependencies and quantifying the number of components an attacker needs to execute an attack. By tracing the dependencies and analyzing the connections between different product families, we can identify the critical components an attacker must compromise to traverse the dependency chain and reach their ultimate target.

We see components as different types of knowledge required for understanding the system. Just as knowledge builds upon foundational concepts, dependencies between components create a hierarchical structure, where certain components act as prerequisites for others. We can navigate the intricate web of dependencies, determine the core components an attacker needs to compromise, and gain a deeper understanding of the attack surface.

Utilizing product family algebra to answer the question of how many components

an attacker needs to execute an attack offers several benefits. It provides a systematic and structured methodology for analyzing dependencies, identifying critical components, and prioritizing defense measures.

5.1.1 Product Family Algebra

Product family algebra (PFA) [35] is an algebraic feature modeling technique with the power to describe product families precisely. It is based on the mathematical structure of idempotent commutative semirings. In addition, it allows algebraic calculations and manipulations of product families to generate new information about those product families.

A *semiring* is a mathematical structure $\big(S, +, \cdot, 0, 1\big)$ where $\big(S, +, 0\big)$ is a commutative monoid and $\big(S, \cdot, 1\big)$ is a monoid such that operator $\cdot$ distributes over operator $+$ and element 0 is annihilates S with respect to $\cdot$. We say that a semiring is *idempotent* if operator $+$ is idempotent (i.e., $x + x = x$). We say that a semiring is *commutative* if operator $\cdot$ is commutative (i.e., $x \cdot y = y \cdot x$).

Definition 5.1 (Product Family Algebra)**.** *A product family algebra (PFA) is an idempotent and commutative semiring* $\big(S, +, \cdot, 0, 1\big)$ *where each element of the semiring is a product family.*

In the product family context, $+$ can be interpreted as a choice or option between two product families and $\cdot$ can be interpreted as a mandatory composition of two product families. The constant 0 represents the empty family and the constant 1 represents the family that has one product without features. The term $a + 1$ is the product family offering the choice between a and the identity product and indicates that the feature a is optional.

With the above interpretations, other concepts in product family modeling can be expressed mathematically. In general, each idempotent semiring $\big(S, +, \cdot, 0, 1\big)$ has a natural partial order $\leq$ on S defined by $a \leq b \iff a + b = b$. Therefore, for product families $a, b \in S$, $a \leq b$ indicates that a is a *sub-family* of b if and only if $a + b = b$.

5.1.2 Products and Features

The basic building blocks of a product family in PFA are *products* and *features*.

Definition 5.2 (Product)**.** *We say that a is a product if it is different than* 0 *and satisfies:*

$$\forall(b \mid: b \leq a \implies (b = 0 \ \vee \ b = a)) \tag{5.1}$$

$$\forall(b, c \mid: a \leq b + c \implies (a \leq b \ \vee \ a \leq c)) \tag{5.2}$$

Equation 5.1 shows that a product does not have a subfamily except the empty family and itself. Equation 5.2 indicates that if a product a is a subfamily of a family formed by c and b, it must be a subfamily of one of them. Intuitively, this indicates that a product cannot be split using the choice operator $+$.

Definition 5.3 (Feature)**.** *We say that a is a feature if it is a proper product different than* 1 *satisfying:*

$$\forall(b \mid: b \leq a \implies (b = 1 \ \vee \ b = a)) \tag{5.3}$$

$$\forall(b, c \mid: a|b \cdot c \implies (a|b \ \vee \ a|c)) \tag{5.4}$$

where the division operator $|$ is defined by $a|b \iff \exists(c \mid: b = a \cdot c)$.

Equation 5.3 states that if we have a product b that divides a, then either b is 1 or $b = a$. Equation 5.4 states that for all product families b and c, if a is mandatory to form $b \cdot c$, then it is mandatory to form b or it is mandatory to form c. Intuitively, this indicates that a feature cannot be split using the composition operator $\cdot$.

New product families can be derived from other existing product families by adding features. The *refinement* relation captures such a relationship between two product families.

Definition 5.4 (Refinement)**.** *The refinement relation on a PFA is defined as follows:*

$$a \sqsubseteq b \iff \exists(c \mid c \in S \ : \ a \leq b \cdot c)$$

Informally, a product family a refines another product family b if a has the same set of features as b and possibly more. For example, assume that we have a new_mobile that has screen, keypad, calling_feature, and GPS. We have also an old_mobile that has screen, keypad, and calling_feature. Therefore, new_mobile refines old_mobile because every product in new_mobile has all features of some products in old_mobile. When a product a refines a product b, we say that b is a *sub-product* of a.

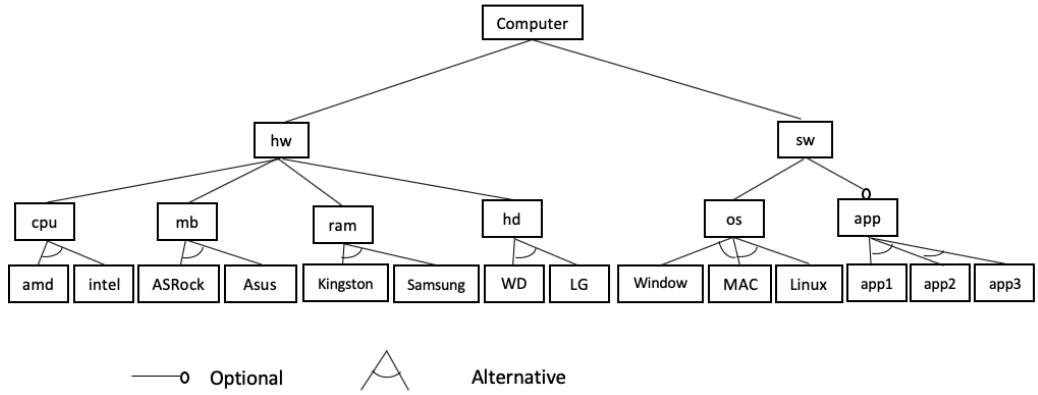

Figure 5.1: Feature model for a computer system

Next, we develop a model of computer systems and their variants for different distributions within a population of users. To achieve this, we use product family algebra. PFA helps to capture and analyze the commonalities and variabilities of a product family and allows mathematical description and manipulation of product family specifications.

5.1.3 An Example Population of Computer Systems

For simplicity, suppose that a computer system is comprised of hardware (hw) and software (sw). The hardware for the system involves only a central processing unit (CPU), motherboard (mb), random access memory (RAM), and a hard drive (hd). The software for the system involves only an operating system (OS) and an optional application (app). For any computer system, there are two kinds of CPU, mb, RAM, and hd, and three kinds of OS and three kinds of app. The product family of computer systems can be visualized as a graphical feature model as shown in Figure 5.1.

To study populations of computer systems, we assume that we have a set of computer system users. Each user operates a computer system that can be built from the product family described above. In this research, we assume that the products that can be built from this product family are distributed uniformly among the population of users. This assumption enables us to focus on defining and presenting the main conceptual model. We will use this computer system product family throughout the following sections to illustrate various aspects of our model.

5.1.4 Specifying the Computer System Product Family using PFA

We use PFA to specify the product family described in Section 5.1.3 and illustrated in Figure 5.1. We begin by declaring the basic features of computer systems. The basic features represent all of the possibly components that can be used to build a computer system. In our example, we have 14 basic features corresponding to the specific kinds of CPU, mb, RAM, hd, OS and app. Then, using the basic features and the operators of PFA, we define a labeled product family specifying the mandatory and optional features of products. For example, as described in Section 5.1.3, the software for a computer system requires only an operating system and an optional application. This is represented as $\text{sw} = \text{OS} \cdot (\text{app} + 1)$ indicated that it is mandatory to have an operating system and optional to have application. Subsequently, because we have three kinds of operating system that we can choose from, we specify $\text{OS} = \text{Windows} + \text{MacOS} + \text{Linux}$ to show that an operating system is one of Windows, MacOS or Linux. The complete PFA specification for our example computer system product family is shown in Figure 5.2. We will use this PFA specification of the computer system product family to evaluate the impact of a vulnerable component on the security of an entire population of computer systems from this product family.

5.1.5 Computing the Size of the Catalog of Products

Given the PFA specification of a product family, such as that shown in Figure 5.2, we can compute the total number of products that can be built from the specification.

Definition 5.5 (Catalog Size)**.** *Let C be a product family. Then, the number of products that can be built from C is called the* catalog size *(denoted $|C|$) and is computed recursively on the structure of product family algebra:*

$$\begin{aligned} |0| &= 0 \\ |1| &= 1 \\ |a| &= 1 \\ |a + b| &= |a| + |b| \\ |a \cdot b| &= |a| \times |b| \end{aligned}$$

```
% Declarations of basic features
bf amd
bf intel
bf ASRock
bf Asus
bf Kingston
bf Samsung
bf WD
bf LG
bf Windows
bf MacOS
bf Linux
bf app1
bf app2
bf app3
```

```
% Definitions of labeled product family
Computer = hw · sw

      hw = CPU · mb · RAM · hd
      sw = OS · (app + 1)

     CPU = amd + intel
      mb = ASRock + Asus
     RAM = Kingston + Samsung
      hd = WD + LG

      OS = Windows + MacOS + Linux
     app = app1 + app2 + app3
```

Figure 5.2: A PFA specification of the computer system product family

Example 5.1 (Computing the Catalog Size)**.** *Applying Definition 5.5 to the PFA specification of our computer system product family in Figure 5.2, we have* 192 *possible computer system products in the catalog.*

$$
\begin{aligned}
|\text{Computer}| &= |\text{hw} \cdot \text{sw}| \\
&= |\text{hw}| \times |\text{sw}| \\
&= |\text{CPU} \cdot \text{mb} \cdot \text{RAM} \cdot \text{hd}| \times |\text{OS} \cdot (\text{app} + 1)| \\
&= [(1+1) \times (1+1) \times (1+1) \times (1+1)] \times \\
&\quad\ [(1+1+1) \times (1+1+1) + 1] \\
&= [2 \times 2 \times 2 \times 2] \times [3 \times 4] \\
&= 16 \times 12 \\
&= 192
\end{aligned}
$$

For the sake of consistency we will refer to the set of products that can be built from the specification of a product family as the *catalog*.

In Chapter 4, we discussed how attackers who reuse previous methods can narrow down their options. They focus on what's worked before or on known weak spots in systems. This makes their job faster and easier because they're not looking at every single possible way to attack. It's like having a smaller list of keys to try on a lock. If an attacker knows about certain weak spots, they can focus just on those, reducing their options from the huge list of all possible attacks to a smaller list.

Now, in Chapter 5, we're going to see what happens when we change or add new things to a computer system. Imagine a computer system is like a set of products in a store. If you add new features or different versions of these products, you end up with a lot more products in your store. This is like making the list of possible computer setups much longer because you have more combinations of features to choose from.

The idea here is simple: when you add new stuff to the mix, you have more ways to put things together. It's like adding more pieces to a puzzle; the more pieces you have, the more ways you can fit them together to make different pictures. In computer terms, adding these variations means you can make lots of computer systems, each with its own set of features.

This is really important for people who work in cybersecurity. It means that as we keep adding new features or changing things in our computer systems, we're also creating new ways these systems can be set up. Each setup might have its own weak points or strong points.

Chapter 5 shows us that just like adding new products to a store makes the store's inventory bigger, adding new features to a computer system makes the number of possible system setups bigger. This means there are more points for attackers to try. It's a constant game of keeping up with these changes to make sure we stay one step ahead of potential threats.

It is important to understand how the addition of new alternative products or features (i.e., variations) in a product family affects the catalog size. The following proposition shows that adding non-zero variations to a product family increases the *catalog size*.

Proposition 5.1 (Variations Increase the Catalog Size)**.** *Adding non-zero variations*

to a product family increases the size of the catalog.

Proof. Let C be the original product family and let C' be the product family after adding a non-zero variation. A variation is the addition of an alternative feature that allows two products to differ in the choice of that feature. Then,

$$
\begin{array}{ll}
& |C| < |C'| \\
\Longleftarrow & \langle \text{ Hypothesis: } C' \text{ is the product family } C \text{ with an additional product or feature } v \neq 0 \ \rangle \\
& |C| < |C + v| \\
\Longleftrightarrow & \langle \text{ Definition 5.5 } \rangle \\
& |C| < |C| + |v| \\
\Longleftrightarrow & \langle \text{ Arithmetic } \quad \& \quad |v| > 0 \ \rangle \\
& \mathsf{true}
\end{array}
$$

□

Intuitively, Proposition 5.1 states that if we provide more choices to build a computer system, we can build more products that can be distributed among our population of users.

In Chapter 4, we assessed the difficulty of carrying out an attack by looking at how likely it is to succeed, both with and without the reuse of known methods or vulnerabilities. This approach helped us understand how using previous knowledge can make an attack more likely to work. It was like estimating the chances of picking the right key for a lock when you already know some keys that worked in the past.

Now, in Chapter 5, we're going to look at this idea from a different angle. We're moving away from just thinking about the likelihood of an attack succeeding and instead focusing on the concept of catalog size. Remember from the last section, the catalog size is like the total number of different setups or versions of a computer system we can have, especially when we keep adding new features or variations.

By measuring the difficulty of an attack based on the catalog size, we're essentially saying, "How hard is it to find a weakness when there are so many different system setups?" Imagine trying to pick the right key out of a huge pile of keys, where each

key is slightly different. The more keys there are, the harder it is to pick the right one. In the same way, the bigger the catalog size, the more challenging it becomes for an attacker to find a vulnerability that works across all these different setups.

This new measure is important because it reflects the real-world complexity of modern computer systems. These systems are no longer simple and static; they're complex and constantly evolving. Each new feature or change adds another layer to the puzzle. For attackers, this means they can't just rely on old methods; they have to constantly adapt and look for new weaknesses, which becomes harder as the number of possible system configurations increases.

Chapter 5 introduces a new way to think about the difficulty of conducting an attack. It's not just about how likely an attack is to succeed, but also about how complex and varied the target system is. The larger the catalog size, the more daunting the task becomes for attackers. For cybersecurity experts, this means that continuously updating and diversifying system features can be a powerful strategy to enhance security. The goal is to make the catalog of possible system configurations so large and diverse that finding a successful attack method becomes like finding a needle in a haystack.

In the previous section, we developed a model of a product family of computer systems that are distributed among a population of users. Using this model, we now study the *population fragility* using these computer systems. In this section, we define a measure of the *population fragility* with respect to a known exploitable system component. Then, we study how increasing variability in the product family can improve the *population fragility*.

5.1.6 Defining a Measure of the population fragility

Because products within a product family contain commonalities, there is a potential that multiple products contain the same exploitable component or sub-product. To study this phenomenon, we aim to define a measure to show how much of a population is susceptible to an attack when we know that there exists an exploitable component in the computer system product family. To do so, we need to identify which of the products in a product family contain an exploitable sub-product; that is, the set of products which contain a vulnerability that can be exploited by an adversary to

conduct an attack on the system. We call this set of products the *set of exploitable products*.

To determine set of exploitable products, we assume that there is an exploitable sub-product that we know about beforehand. Using this information, we find the set of products in the catalog that contain these exploitable sub-products. By determining the number of exploitable products with respect to the total number of products that can be built from the product family (i.e., the catalog size as defined in Section 5.1.5), we can determine the proportion the population[1] that is susceptible to an attack on the known exploitable sub-product. We call this measure the *population fragility*. Formally, the measure of the *population fragility* is defined as follows:

Definition 5.6 (Fragility)**.** *Let C be a product family and let x be an exploitable sub-product. Then, the fragility of C with respect to x is given by:*

$$\text{Fragility}(C, x) = \frac{|X|}{|C|}$$

where $X = \{c \mid c \in C \ \wedge \ c \sqsubseteq x\}$ is the set of exploitable products in the product family C.

The relationship between trust and fragility is a crucial aspect of system security. Trust in specific system components, such as widely used software or standardized configurations, often leads to these components being reused across multiple systems within a population. This widespread adoption creates a monoculture effect, where vulnerabilities in trusted components can lead to large-scale exploitation. Trust, when placed in homogeneous setups, inherently increases fragility because it encourages the reuse of known configurations that attackers can predict and target. Consequently, fragility rises as the attack surface becomes more uniform and predictable, making it easier for adversaries to exploit shared weaknesses. Effective security, therefore, requires a balanced approach that questions excessive trust in common solutions and promotes variability and diversification to reduce overall fragility.

For the sake of our example, suppose that it has been revealed that there is an exploitable vulnerability affecting computer systems containing the combination of

[1]Note that because we assume that the products in the catalog are uniformly distributed among all users within a population, we can view the catalog size and the size of the population as being equal.

Windows, intel, and app1. Because we do not know exactly which of the features Windows, intel, and app1 has the exploitable vulnerability—it may be one of them or any combination of them—we say that all computer systems that contain the exploitable sub-product (Windows $\cdot$ intel $\cdot$ app1) are vulnerable. Applying Definition 5.6 shows that, out of 192 possible computer system products, there are 8 products that contain the known exploitable sub-product. This means that the *population fragility* is 0.0417 as detailed in Example 5.2 below.

Example 5.2 (population fragility with respect to the exploitable sub-product Windows $\cdot intel \cdot app1$). *Applying Definition 5.6, the set of exploitable products is given by:*

$$\begin{aligned} X = \{&(\text{intel} \cdot \text{ASRock} \cdot \text{Kingston} \cdot \text{WD} \cdot \text{Windows} \cdot \text{app1}), \\ &(\text{intel} \cdot \text{Asus} \cdot \text{Samsung} \cdot \text{LG} \cdot \text{Windows} \cdot \text{app1}), \\ &\ldots, \\ &(\text{intel} \cdot \text{Asus} \cdot \text{Samsung} \cdot \text{WD} \cdot \text{Windows} \cdot \text{app1})\} \end{aligned}$$

Therefore, $|X| = 8$. *Using the results from Example 5.1, we can compute the fragility of* C *with respect to* x*:*

$$\text{Fragility}(C, x) = \frac{|X|}{|C|} = \frac{8}{192} = 0.0417$$

Now that we are able to compute the *population fragility*, we turn our attention to determining how we can improve the *population fragility* by adding variations to the product family of computer systems distributed among the population.

In Chapter 4, we introduced two key security metrics: the KOSM and the DESM. These metrics help us understand how well a system can adapt to potential threats. KOSM focuses on how a system's security adapts based on past knowledge, while DESM measures how efficiently a system can dynamically defend against new threats. These metrics offer valuable insights into a system's overall security resilience.

Now, in Chapter 5, we shift our focus to examine the impact of adding variations to products within a system, specifically contrasting the effects of these variations on exploitable versus non-exploitable products. This comparison is crucial in understanding how changes in a system's components can influence its overall security posture.

First, consider adding variations to exploitable products. These are products or components within a system that are known to have vulnerabilities or have been exploited in the past. Introducing variations here can be a double-edged sword. On one hand, it could potentially confuse attackers, as they now face a broader array of targets, each slightly different from the others. On the other hand, each new variation might introduce new vulnerabilities, potentially increasing the system's overall risk. Here, KOSM and DESM would help assess whether the added complexity actually aids in defense or merely expands the attack surface.

In contrast, adding variations to non-exploitable products presents a different scenario. These products are considered secure or haven't been a target in the past. Introducing variations in this category could bolster the system's security, as it adds complexity without significantly increasing vulnerability. The variations could act as a form of defense in depth, creating additional layers that an attacker must navigate. In this case, the DESM would likely show an improvement in the system's ability to dynamically defend against attacks, as the variations enhance the system's complexity without compromising its integrity.

Section 5.1.7 and Section 5.1.8 dive into the nuanced implications of adding variations to different types of products within a system. By comparing the effects on exploitable and non-exploitable products, we gain a clearer understanding of how these changes can either strengthen or weaken a system's security. The key takeaway is that while adding variations can be a powerful strategy for enhancing security, it needs to be applied judiciously, considering the nature of the products being varied and the overall impact on the system's security metrics like KOSM and DESM.

5.1.7 Adding Variations to Exploitable Products

As shown in Proposition 5.1, adding variations to a product family specification can increase the size of the catalog of products that can be built. This means that we can have more products with more variability, meaning that it is less likely that products share combinations of features. When considering how we can improve the *population fragility*, there are several places where we can add variations. One of these places is in the labeled product families that contain the exploitable sub-product. To illustrate our intuition, we carry out a broad example of adding variations in this manner. The

details are described in Example 5.3 below.

Example 5.3 (Adding variation to labeled product families that contain the exploitable sub-product)**.** *We continue to assume that we have the exploitable sub-product* $x = (\text{Windows} \cdot \text{intel} \cdot \text{app1})$*.Consider the addition of one more alternative to each of* OS*,* CPU*, and* app *labeled product families in the PFA specification shown in Figure 5.2. More specifically, suppose we add* Unix *as an alternative operating system,* Threadripper *as an alternative* CPU*, and* app4 *as an alternative* app*. Note that these additions yield new choices to avoid the features present in the exploitable sub-product. The revised PFA specification for our example computer system product family with the added variations is shown in Figure 5.3.*

As a result, the set of exploitable products (i.e., X) is the same as in Example 5.2. Therefore, $|X|$ remains 8*, while the catalog size (i.e. $|C|$), computed using Definition 5.5, increases from* 192 *to* 480*. Thus, by applying Definition 5.6, the population fragility is reduced to* 0.0167 *as a result of these added variations in the product family.*

The following proposition generalizes our intuition that adding variations to exploitable products reduces the *population fragility.*

Proposition 5.2 (Adding Variation to Exploitable Products)**.** *Adding non-zero variations in an exploitable product decreases the population fragility.*

Proof. Assume an exploitable sub-product x. Let $\text{Fragility}(C, x)$ be the original *population fragility* and let $\text{Fragility}(C', x)$ be the *population fragility* after adding a non-zero variation in an exploitable product.

$$
\begin{aligned}
&\text{Fragility}(C, x) > \text{Fragility}(C', x) \\
\iff &\langle \text{ Definition 5.6 } \rangle \\
&\frac{|X|}{|C|} > \frac{|X'|}{|C'|} \\
\impliedby &\langle \text{ Hypothesis: } X \subseteq X' \Longrightarrow |X'| = |X' \backslash X| + |X| \rangle \\
&\frac{|X|}{|C|} > \frac{|X' \backslash X| + |X|}{|C'|} \\
\iff &\langle \text{ Fraction Addition } \rangle \\
&\frac{|X|}{|C|} > \frac{|X' \backslash X|}{|C'|} + \frac{|X|}{|C'|}
\end{aligned}
$$

```
% Declarations of basic features
bf amd
bf intel
bf Threadripper
bf ASRock
bf Asus
bf Kingston
bf Samsung
bf WD
bf LG
bf Windows
bf MacOS
bf Linux
bf Unix
bf app1
bf app2
bf app3
bf app4
```

```
% Definitions of labeled product family
Computer = hw · sw

      hw = CPU · mb · RAM · hd
      sw = OS · (app + 1)

     CPU = amd + intel +Threadripper
      mb = ASRock + Asus
     RAM = Kingston + Samsung
      hd = WD + LG

      OS = Windows + MacOS + Linux + Unix
     app = app1 + app2 + app3 + app4
```

Figure 5.3: Revised PFA specification with new variations (emphasize in boldface) in labeled product families that contain the exploitable sub-product $x = (\text{Windows} \cdot \text{intel} \cdot \text{app1})$

$$
\begin{aligned}
&\Longleftarrow \langle \text{ Hypothesis: Add variation in an exploitable sub-product } \implies \neg\, \exists (c \mid \\
&\quad c \in C'\backslash C \,:\, c \sqsubseteq x\,) \implies |\{c \mid c \in C \wedge c \sqsubseteq x\}| = |\{c \mid c \in C' \wedge \\
&\quad c \sqsubseteq x\}| \implies |X| = |X'| \implies |X'\backslash X| = 0 \,\rangle \\
&\frac{|X|}{|C|} > \frac{|X|}{|C'|} \\
&\Longleftrightarrow \langle \text{ Proposition 5.1: } |C| < |C'| \,\rangle \\
&\quad \mathsf{true}
\end{aligned}
$$

□

Proposition 5.2 shows that if we can decrease the likelihood of an exploitable

sub-product being shared among a large proportion of the population, then we can improve the *population fragility*. The results of Proposition 5.2 show that this can be achieved by providing more alternatives to avoid known combinations of vulnerable system components that so that a smaller proportion of the population shares these vulnerabilities.

In the ongoing discourse of product family cybersecurity, we've recognized the significance of fragility—a measure indicating the vulnerability of a population of systems to specific exploits. This conversation is now furthered by examining how introducing variations to a product family can influence this fragility. Adding variations has a direct effect on the composition of the product family, altering the probability that multiple products share a common vulnerability.

The concept of introducing variations is tightly coupled with the prior discussions on the attack search space, catalog size, and the population fragility. By expanding the variety of features within a product family, we inherently dilute the concentration of any single point of failure, thereby reducing the overall fragility. This is a strategic move that mirrors the defensive depth approach in cybersecurity, where diversity and complexity are allies in stymieing the attacker's path.

The fragility model closely relates to concepts of diversity, software, and knowledge reuse. By modeling system configurations using product family algebra, the approach highlights how diversity in software and hardware components disrupts an attacker's ability to consistently apply known exploits. The reuse of knowledge, such as previously successful attack methods, becomes less effective in a diverse environment where no two systems are exactly alike. Introducing variations in software configurations, hardware components, or operational settings increases the complexity of the attack landscape. This strategic diversification makes it harder for attackers to leverage reused knowledge, thereby expanding the attack search space and lowering the overall fragility of the system population. In this context, promoting diversity is not just about adding more options but about strategically varying system setups to minimize shared vulnerabilities and enhance security resilience.

The addition of new, non-zero variations into a product family is akin to introducing new genes into a biological pool. Just as biodiversity can lead to a more resilient ecosystem, a diversified product family can result in a more robust system population

against cyber threats. The impact of this strategy is twofold: it not only decreases the likelihood that any single vulnerability will be widespread but also increases the complexity an attacker must navigate to find and exploit vulnerabilities.

In the context of the Knowledge Obfuscation Security Metric (KOSM) and the Defense Evolution Security Metric (DESM), the introduction of variations serves as a proactive measure to increase system security. KOSM benefits from these variations by making the attack search space more complex and less predictable, which aligns with its goal of making it harder for attackers to understand and navigate the system. DESM, which is concerned with the evolution and adaptability of defensive measures, leverages these variations to continually shift the security landscape, making it more challenging for attackers to reuse their knowledge and tools effectively.

The proposition and its proof underscore an essential principle in cybersecurity: complexity and change are adversaries of exploitation. By ensuring that no single vulnerability can affect a large portion of the system population, we increase the resilience of individual systems and the population as a whole. This approach encourages the adoption of a dynamic and varied defensive strategy, which is at the heart of both KOSM and DESM, underscoring their relevance in modern cybersecurity practices.

5.1.8 Adding Variations to Non-exploitable Products

In the previous section, we added variations in the labeled product families that contain the exploitable sub-product. Here we explore the affect that adding variations in the non-exploitable sub-products has on the *population fragility*. As in the previous section, we being with a broad example of adding variations in this manner to illustrate our intuition. The details are described in Example 5.4 below.

Example 5.4 (Adding variation to labeled product families that do not contain the exploitable sub-product)**.** *Again we assume that we have the exploitable sub-product* $x = (\text{Windows} \cdot \text{intel} \cdot \text{app1})$*. Consider the addition of one more alternative to each of the labelled product families* CPU*,* mb*,* RAM*,* hd*,* OS*, and* app *in the PFA specification shown in Figure 5.2. The revised PFA specification is similar to that shown in Figure 5.3. Note that that only do these additions overlap with the exploitable sub-product* $x = (\text{Windows} \cdot \text{intel} \cdot \text{app1})$*, but also with the non-exploitable sub-products in the product family.*

As a result of the additions, the number of exploitable products (i.e., $|X|$) increases from 8 *to* 27 *and the catalog size (i.e., $|C|$) increases from* 192 *to* 1620. *Applying Definition 5.6, we find that the population fragility is* 0.0167 *again.*

In Example 5.4, the *population fragility* remains unchanged because the ratio between the number of exploitable products and the catalog size remains same as that in Example 5.3. Adding more variations to the non-exploitable products (e.g., mb, RAM, and hd), increases the number of products that contain the exploitable sub-product while also increasing the catalog size; we can build new products, but a subset of those new products inevitably contain the exploitable sub-product and thus become exploitable products themselves. Therefore, increasing variations in the non-exploitable sub-products does not help to reduce the *population fragility*. The following proposition generalizes these observations:

Proposition 5.3 (Adding Variation to Non-exploitable Products)**.** *Adding non-zero variations in a non-exploitable sub-product does not change the population fragility.*

Proof. Assume an exploitable sub-product x. Let $\text{Fragility}(C, x)$ be the original *population fragility* and let $\text{Fragility}(C', x)$ be the *population fragility* after adding a non-zero variation in a non-exploitable sub-product.

$$
\begin{aligned}
&\text{Fragility}(C, x) = \text{Fragility}(C', x) \\
\iff &\langle \text{ Definition 5.6 } \rangle \\
&\frac{|X|}{|C|} = \frac{|X'|}{|C'|} \\
\impliedby &\langle \text{ Multiply Both Sides by 2 } \rangle \\
&2\frac{|X|}{|C|} = 2\frac{|X'|}{|C'|} \\
\iff &\langle \text{ Multiply Both Sides by } 1 = \tfrac{|C'|}{|C'|} = \tfrac{|C|}{|C|} \rangle \\
&2\frac{|X||C'|}{|C||C'|} = 2\frac{|X'||C|}{|C'||C|} \\
\iff &\langle \text{ Expand Multiplication: } 2a = a + a \rangle \\
&\frac{|X||C'| + |X||C'|}{|C||C'|} = \frac{|X'||C| + |X'||C|}{|C'||C|} \\
\iff &\langle \text{ Fraction Addition } \rangle \\
&\frac{|X||C'|}{|C||C'|} + \frac{|X||C'|}{|C||C'|} = \frac{|X'||C|}{|C'||C|} + \frac{|X'||C|}{|C'||C|}
\end{aligned}
$$

$$\iff \langle \text{ Arithmetic } \rangle$$
$$\frac{|X||C'|}{|C||C'|} - \frac{|X'||C|}{|C'||C|} = \frac{|X'||C|}{|C'||C|} - \frac{|X||C'|}{|C||C'|}$$
$$\iff \langle \text{ Subtract Both Sides by } \tfrac{|C'||C|}{|C'||C|} \rangle$$
$$\frac{|X||C'|}{|C||C'|} - \frac{|C'||C|}{|C'||C|} - \frac{|X'||C|}{|C'||C|} = \frac{|X'||C|}{|C'||C|} - \frac{|C||C'|}{|C||C'|} - \frac{|X||C'|}{|C||C'|}$$
$$\iff \langle \text{ Arithmetic } \& \text{ Distributivity } \rangle$$
$$\frac{|X||C'|}{|C||C'|} - \frac{\big(|C'| - |X'|\big)|C|}{|C'||C|} = \frac{|X'||C|}{|C'||C|} - \frac{\big(|C| - |X|\big)|C'|}{|C||C'|}$$
$$\iff \langle \text{ Cancellation } \rangle$$
$$\frac{|X|}{|C|} - \frac{|C'| - |X'|}{|C'|} = \frac{|X'|}{|C'|} - \frac{|C| - |X|}{|C|}$$
$$\iff \langle \text{ Arithmetic } \rangle$$
$$\frac{|X|}{|C|} + \frac{|C| - |X|}{|C|} = \frac{|X'|}{|C'|} + \frac{|C'| - |X'|}{|C'|}$$
$$\Longleftarrow \langle \text{ Hypothesis: } X \subseteq C \implies |C \backslash X| = |C| - |X| \ \wedge$$
$$X' \subseteq C' \implies |C' \backslash X'| = |C'| - |X'| \ \rangle$$
$$\frac{|X|}{|C|} + \frac{|C \backslash X|}{|C|} = \frac{|X'|}{|C'|} + \frac{|C' \backslash X'|}{|C'|}$$
$$\iff \langle \text{ Fraction Addition } \rangle$$
$$\frac{|X| + |C \backslash X|}{|C|} = \frac{|X'| + |C' \backslash X'|}{|C'|}$$
$$\Longleftarrow \langle \text{ Hypothesis: } X \subseteq C \implies |C| = |C \backslash X| + |X| \ \wedge$$
$$X' \subseteq C' \implies |C'| = |C' \backslash X'| + |X'| \ \rangle$$
$$\frac{|C|}{|C|} = \frac{|C'|}{|C'|}$$
$$\iff \langle \text{ Cancellation } \& \ |C| \neq 0 \ \& \ |C'| \neq 0 \ \rangle$$
$$1 = 1$$
$$\iff \langle \text{ Reflexivity of } = \rangle$$
$$\mathsf{true}$$

□

Proposition 5.3 emphasizes the point that not all variations are effective at reducing the *population fragility*. This is important because a tendency may be to simply add as many more variations that can be afforded into the product family. We need to be careful to not simply build more products that contain exploitable sub-products.

The decision of where to introduce these variations needs to be much more strategic as indicated by the results of Proposition 5.2.

Continuing from the insights gained in Chapter 4, Chapter 5 delves deeper into the dynamics of cybersecurity through the lens of product family algebra. The chapter opens by highlighting the complexity inherent in computer systems, comparing them to a lock with many key combinations. Each setup of a computer system, like each lock, has its vulnerabilities and strengths. The ever-changing nature of these systems, with regular updates and modifications, further complicates the task for potential attackers, akin to a lock that constantly changes its combination.

The heart of Chapter 5 lies in the application of product family algebra to cybersecurity. This approach allows for a detailed analysis of the dependencies and relationships among system components. By categorizing these components into product families, cybersecurity professionals can identify critical components and potential attack vectors, enhancing their understanding of the system's vulnerabilities. This method draws a parallel between the foundational knowledge of a subject and the core components of a system, emphasizing the importance of understanding the fundamental elements to anticipate and counteract potential attacks.

In practical terms, the chapter discusses how the addition of new features or variations to a computer system increases the catalog size, that is, the number of possible system configurations. This expansion in variety makes it more challenging for attackers to find a successful method of attack. For cybersecurity experts, this means a constant effort is required to stay abreast of these changes and adapt defense strategies accordingly. The chapter concludes with a critical analysis of how variations within a product family affect its overall security. While adding variations to exploitable products can decrease population fragility, the same cannot be said for non-exploitable products. This distinction underscores the need for strategic thinking in cybersecurity measures, emphasizing that not all changes enhance security equally.

Ultimately, Chapter 5 builds on the concepts introduced in Chapter 4 and expands them by applying product family algebra to the realm of cybersecurity. This approach provides a nuanced understanding of how system configurations, their variations, and interdependencies play a crucial role in defining a system's security landscape. The chapter serves as a crucial guide for cybersecurity professionals, offering insights into

how the strategic addition of features and variations can fortify systems against a constantly evolving array of cyber threats.

In the upcoming chapter, we will explore "The Role of Information Security Analysts in the Era of KOSM and DESM." This chapter promises to delve into the evolving responsibilities and strategies of information security analysts in the contemporary cybersecurity landscape, particularly in the context of the KOSM and the DESM. As these metrics become increasingly integral to cybersecurity practices, the role of analysts is shifting towards a more dynamic, adaptive approach. We will examine how these professionals are adapting their skills and methodologies to navigate the complexities introduced by KOSM and DESM, ensuring robust and resilient defense mechanisms in the face of sophisticated cyber threats. The chapter aims to provide valuable insights into the strategic thinking, analytical skills, and innovative approaches required for effective cybersecurity management in this new era.

Chapter 6

Knowledge Reuse and Security Analysis

In the field of cybersecurity, the necessity for improved defensive strategies is pressing. Chapters 4 and 5 offered a perspective on how the reuse of known attack methods shapes the attack search space and how software reuse impacts security.

This chapter aims to bridge the theoretical concepts introduced previously with the world of practical security analysis. Here we will explore how the models and metrics, KOSM, DESM, and fragility, can be employed by security analysts to bolster defenses against cyber threats.

We will begin by examining how security analysts can adapt to changing threat landscapes using product family algebra principles. This will include understanding how system variations and configurations impact security and the application of sophisticated metrics like KOSM and DESM in crafting robust security strategies. The chapter will also provide insights into the systematic analysis of system configurations and dependencies, emphasizing the continuous learning and adaptation necessary in the fast-paced world of cybersecurity.

In an era where cyber threats are becoming more sophisticated, and system architectures more complex, the need for a deeper, more analytical approach to security is evident. This chapter is not just a theoretical exploration but a practical toolkit, aiming to equip security analysts with the knowledge and skills to stay ahead in the ever-evolving cybersecurity landscape.

As we transition into the main content of this chapter, let us keep in mind that the goal is not just to understand these concepts in isolation but to integrate them into a cohesive, strategic approach to information security analysis. With this perspective, let's embark on a journey that melds theoretical concepts with practical applications, setting a new standard in cybersecurity analysis.

6.1 Current Practice

Although security analysis is complex, the process can be abstracted into three stages: understand the threat model, study defenses, and search for threats in the threat model that are not covered by the defenses. Defensive security analysts engage in this process in order to find areas of potential vulnerability to be addressed, while offensive analysts look for vulnerabilities to be exploited.

In most organizations, however, security analysts also have another role: ensuring compliance. Virtually all large organizations have some regulatory requirements to ensure security, and their business partners and customers often place additional security requirements on them. While sometimes these requirements dictate periodic security reviews, for the most part they take the form of lists of requirements: authentication requirements, data handling and storage procedures, disaster recovery procedures, anti-malware tools, and regular software update procedures, among others. Ensuring compliance can require a lot of effort; further, potential security improvements identified by an analyst may not get much buy-in from management unless those improvements also fall under compliance requirements. Thus, the security posture of many organizations is dictated by the compliance regimes they fall under.

"Best practices" is a general term for the processes and tools that are considered by experts in an area to be the general way thing should be done. Security best practices greatly inform compliance regulations, and as a result there is significant commonality in the compliance requirements. This commonality, however, can actually be a weakness, as we will explain.

6.2 Best Practices and Knowledge Reuse

Best practices can be seen as a form of knowledge reuse by defenders. Such reuse means that attackers can also reuse their knowledge of best practices. We can see this potential impact of attacker knowledge reuse by considering the effect of best practices on our three metrics. As we shall see, current best practices do not optimize any of our proposed metrics.

6.2.1 Population fragility

Population fragility is proportional to how often components and systems are reused—the more reuse, the higher the fragility, as the impact of a vulnerability in any component is proportional to how often that component is used. The question, then, is to what degree do security best practices encourage reuse.

There do exist some requirements for some types of software diversity, e.g., n-version programming for essential components as is done in aerospace—but these are done more with reliability in mind, not security. In other words, the diversity is in service of mitigating errors rather than attacks.

Best practices in security, however, often require the use of approved components and systems. Cryptography is highly standardized, with standard cryptographic algorithms and protocols and certified cryptographic libraries, rather than custom algorithms and libraries. Well-known operating systems backed by strong companies with standard update practices are preferred for secure commercial systems. Firewalls, anti-malware solutions, SIEMs, single sign-on solutions—these and more tend to come from a few reputable vendors.

The tendency to use what is popular is a social risk mitigation strategy, captured in aphorisms such as "Nobody got fired for buying IBM." Popularity, however, can lead to systemic risk, as show by the Solarwinds hack [4] among others. Population fragility gives us a measure of how much risk is incurred by using what everyone else is.

6.2.2 Knowledge Obfuscation

While software reuse is a kind of knowledge reuse, knowledge reuse is a much broader concept and encompasses system design, configurations, administrative processes, security policies, and more. The more attackers know about a target, the better able they are able find and exploit security weaknesses. We refer to knowledge obfuscation (as measured by KOSM) as the degree to which defenders take steps to obscure the knowledge attackers need to conduct their attack, i.e., by expanding the search space that the attacker must search.

Security best practices tend to reduce KOSM, as those best practices tend to be known to both attackers and defenders. For example, compliance regulations often

require many kinds of events to be logged to a SIEM so that logs can be retained and analyzed. Attackers thus know their targets often have SIEMs running, and depending upon the targeted industry or organization, attackers can know which of a small number of SIEMs are being used based on their relative market share. Further, these SIEMs will tend to be configured in certain standard ways, because those are the way the vendors and other experts recommend they be configured, meaning that network topology, the kind of data stored, and even the access control methods can all be determined by an attacker without surveiling a target simply because they know *that is how things are done*. If required knowledge is not obscure (i.e., is well known as a best practice), KOSM goes down.

To be sure, there are costs with obfuscating knowledge. Such costs are captured by other metrics such as those around administrative costs due to increased training or system complexity. KOSM, however, captures the security benefit of such obfuscation, allowing appropriate trade-offs to be made.

6.2.3 Defense Evolution

Security best practices require that defenders respond to attacker innovation in certain standard ways. In particular, defenders should keep their anti-malware signatures up to date and patch their systems in a timely fashion. Failure to follow such practices is uniformly derided as leaving systems insecure. Because best practices require defenders keep their systems updated, in some ways following best practices leads to increased DESM (defense evolution).

However, DESM is about much more than simply keeping systems up to date. The idea behind DESM is the defender changes where the attacker must search because the system has changed in some way. Patching a vulnerability means that a previously developed exploit will no longer work—but this only invalidates a small amount of the attacker's knowledge. To really change the space, defenders need to make larger moves. So long as defenders are purely reactive, attackers will be able to leverage the knowledge they have to continue to develop successfull attack strategies.

6.3 Mitigating Knowledge Reuse in Practice

Given that existing best practices do not optimize for the proposed measures—KOSM, DESM, and fragility—how can these practices be adapted to reduce the value of attacker knowledge? A key strategy is to increase software diversity, which should occur at the population level, ensuring that different systems behave differently. This could involve building systems using sets of interchangeable components with similar functionality, allowing for an exponential increase in configuration possibilities. This approach, supported by the population fragility metric, would also enhance KOSM and DESM if the configurations were periodically altered, preventing attackers from adapting to specific system setups. However, this approach represents a significant shift from conventional software engineering practices and would require substantial effort to ensure novel configurations maintain acceptable functionality and performance.

There are also less disruptive steps that defenders can implement to increase security metrics. Adaptive defenses, such as those based on anomaly detection or machine learning, dynamically respond to changing attack patterns, enhancing DESM by evolving in response to new threats. However, these defenses improve KOSM only if they generate site-specific profiles, creating a localized adaptation strategy. This profile-level diversity ensures that a successful attack on one system does not guarantee success on another, even with similar hardware and software, translating to improved security with manageable overhead.

Security analysts should conduct detailed analyses of system configurations and dependencies, applying KOSM and DESM to identify vulnerabilities and tailor defenses accordingly. Making non-standard choices, such as altering typical configurations or implementing obfuscation techniques like port knocking, can also invalidate previously obtained attacker knowledge, enhancing KOSM and DESM without significantly altering existing practices. For instance, even widely used security tools like SIEMs can be configured in unique ways, such as running multiple independent SIEMs with a unified read-only interface, complicating attacker efforts by introducing unexpected monitoring layers.

While these efforts may challenge the principle of "avoid security through obscurity," our research suggests that some level of obscurity is integral to effective

computer defense. In practical contexts, adaptive defenses and knowledge obfuscation—though often viewed skeptically—can be strategically used to keep attackers at a disadvantage. It is crucial, however, to recognize the limitations: increasing complexity and non-standard configurations can introduce new vulnerabilities, such as adversarial attacks targeting adaptation logic in machine learning systems. Security analysts must balance these strategies against potential scalability issues and ensure that the benefits of increased metric scores outweigh the operational impacts.

Thus, improving KOSM and DESM is achievable through adaptive, localized defenses and deliberate configuration choices that make systems less predictable to attackers. However, blindly following existing best practices will not enhance these metrics, potentially leaving systems more vulnerable than they need to be.

6.4 Limitations of Proposed Metrics

While the strategies outlined can significantly enhance security metrics like KOSM and DESM, they also come with inherent limitations that must be carefully managed in practice:

- Complexity and Vulnerability: While increased complexity in security measures can enhance metrics like KOSM and DESM, it may also introduce new vulnerabilities. Complex adaptive systems, especially those relying on machine learning, can be susceptible to adversarial attacks targeting the adaptation logic itself. For instance, attackers might manipulate inputs to train an adaptive defense incorrectly, causing it to behave predictably or ineffectively.

- Scalability Issues: Adaptive defenses, particularly those that generate unique profiles per site, can introduce significant overhead. The additional computational and maintenance burden may not scale well across large or highly interconnected environments, where consistent performance and availability are critical.

- Knowledge Sharing and Misuse: Adaptive defenses are most effective when unique to each environment; however, this approach can be compromised if attackers gain knowledge about adaptive mechanisms. The risk of attackers

learning and sharing insights about commonly used adaptive strategies remains a challenge, potentially reducing the effectiveness of KOSM if defenses are not sufficiently localized.

- Implementation Challenges: Practical application of the proposed metrics requires careful consideration of system-specific constraints, such as resource availability, existing infrastructure, and organizational readiness for complex adaptive systems. Security analysts must evaluate whether the benefits of increased metric scores outweigh the potential costs and operational impacts.

In conclusion, while the proposed metrics and adaptive strategies provide a promising pathway for enhancing cybersecurity, it is essential to carefully weigh these approaches against their potential drawbacks. The balance between increasing security and avoiding new vulnerabilities is delicate, requiring ongoing assessment and tailored implementation to ensure that the gains in resilience do not inadvertently compromise system integrity.

Chapter 7

Conclusion

This chapter represents the culmination of a comprehensive exploration into the evolving landscape of cybersecurity. It synthesizes insights from the literature, the motivation behind advanced security strategies, novel methodologies for modeling attack difficulty, and the practical application of these concepts through a case study using Product Family Algebra. Here, we reflect on key findings, contextualize their significance within the broader field of cybersecurity, and consider implications for future research and practice. This chapter serves as both a reflection on the journey undertaken and a forward-looking perspective on the path ahead.

The literature review laid the groundwork for understanding the evolution of cybersecurity, tracing its trajectory from simple password protection to today's complex, multi-layered defense systems. It highlighted the escalating arms race between cyber attackers and defenders, setting the context for the subsequent exploration of innovative defense strategies. This foundational understanding was crucial in setting the stage for the advanced methodologies and models introduced later in the dissertation.

Chapter 4 marked a significant shift in the cybersecurity discourse, moving from a static understanding of threats to a dynamic, behavior-based perspective. It delved into the concept of knowledge reuse among attackers, revealing how they adapt and evolve their strategies based on past experiences. The introduction of KOSM and DESM as metrics to quantify defense effectiveness against such adaptive threats was a pivotal moment in the dissertation, providing not just theoretical insights but practical tools for enhancing system security and resilience against increasingly sophisticated attacks.

Chapter 5 presented a deep dive into product family algebra as a novel approach to dissecting and fortifying system architectures. It showcased how understanding the intricate web of system dependencies can reveal potential vulnerabilities and strategies for mitigation. This chapter effectively bridged the gap between theoretical concepts

and practical applications, demonstrating how abstract mathematical principles can be applied to the real-world challenges of cybersecurity.

Chapter 6 synthesized the theoretical insights and methodologies from the previous chapters, illustrating their direct implications and benefits for information security analysts. It highlighted the real-world applicability of the research, validating the proposed models against practical constraints, and demonstrating their potential to significantly advance cybersecurity strategies.

This dissertation makes multiple contributions to the field of cybersecurity. It provides a nuanced, multi-dimensional understanding of cyber threats and introduces innovative tools and strategies for combating them. The introduction of KOSM and DESM as quantifiable metrics offers a new lens through which to assess and enhance system security, while the application of product family algebra provides a structured method to dissect and mitigate complex system vulnerabilities.

However, the nature of cybersecurity is inherently dynamic; as new technologies emerge and threat actors evolve, so too must our strategies for defense. The models and methodologies proposed in this dissertation offer a robust framework, but they must be continually adapted and evolved to remain effective. Future research should focus on expanding these models, exploring new methodologies, and refining our understanding of both the threats we face and the tools we use to combat them.

In conclusion, this dissertation is not just an academic contribution; it's a roadmap for the future of cybersecurity. It underscores the need for a proactive, adaptive, and continuously evolving approach to security, one that anticipates and responds to the ever-changing tactics of cyber adversaries. The insights and strategies presented herein are a significant step forward, providing both a deeper understanding of the cybersecurity challenges and innovative tools to address them. As the digital landscape continues to evolve, so too will the field of cybersecurity. With the foundation laid by this research, the cybersecurity community is better equipped to navigate this ever-changing terrain, ensuring a more secure and resilient digital future for all.

Future work on the concepts presented in this dissertation can be extended both theoretically and practically. Theoretically, further formalization of the models introduced, particularly PFA for security metrics, could provide deeper insights into exploit propagation and vulnerability dynamics. Integrating AI techniques, such as

reinforcement learning, could enhance the search-based framework, enabling adaptive defense mechanisms that evolve alongside emerging threats. Additionally, expanding on the KOSM and exploring cross-disciplinary applications of the proposed metrics can offer broader utility beyond cybersecurity, enriching fields like economics or behavioral sciences. Developing dynamic threat models that incorporate real-time data could also improve proactive defense strategies by predicting potential exploits.

On the practical side, developing a prototype tool that implements the proposed metrics would validate their effectiveness in real-world settings, offering a tangible application of theoretical work. Empirical studies across diverse industries would further demonstrate the metrics' adaptability and impact. Collaborating with industry partners for real-world validation and integrating these models into existing security frameworks like NIST or ISO would bridge the gap between theory and practice. Finally, AI-driven threat mitigation systems could automate security adjustments based on evolving threats, enhancing the adaptability of defense mechanisms in real-time. These extensions collectively aim to refine, validate, and apply the proposed models, contributing to more robust cybersecurity strategies.

Bibliography

[1] S. Ahmad and B. Ehsan, "The cloud computing security secure user authentication technique (multi level authentication)," *IJSER*, vol. 4, no. 12, pp. 2166–2171, 2013.

[2] M. Al-Shabi, "A survey on symmetric and asymmetric cryptography algorithms in information security," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 3, pp. 576–589, 2019.

[3] M. Alabbad, "A feature modelling language based on product family algebra," Master's thesis, McMaster University, Hamilton, ON, Canada, September 2013.

[4] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, "Solar winds hack: In-depth analysis and countermeasures," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2021, pp. 1–7.

[5] R. Anderson, *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2020.

[6] C. Anley, J. Heasman, F. Lindner, and G. Richarte, *The shellcoder's handbook: discovering and exploiting security holes*. John Wiley & Sons, 2011.

[7] R. Azimi and F. Hosseini, "Generative programming: A model driven approach," University of Toronto, Toronto, Canada, Expert Topic Report ECE1770, 2003.

[8] R. K. Baggett and B. K. Simpkins, *Homeland security and critical infrastructure protection*. ABC-CLIO, 2018.

[9] R. Bejtlich, *The practice of network security monitoring: understanding incident detection and response*. No Starch Press, 2013.

[10] B. Biringer, E. Vugrin, and D. Warren, *Critical infrastructure system security and resiliency*. CRC press, 2013.

[11] B. Blakley and D. M. Kienzle, "Some weaknesses of the tcb model," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*. IEEE, 1997, pp. 3–5.

[12] D. J. Bodeau, C. D. McCollum, and D. B. Fox, "Cyber threat modeling: Survey, assessment, and representative framework," MITRE CORP MCLEAN VA MCLEAN, Tech. Rep., 2018.

[13] E. K. Budiardjo, E. M. Zamzami *et al.*, "Feature modeling and variability modeling syntactic notation comparison and mapping," *Journal of Computer and Communications*, vol. 2, no. 02, p. 101, 2014.

[14] S. V. Casteele, "Threat modeling for web application using stride model," *MIS Thesis, Information Security group, Royal Holloway, University of London*, 2004.

[15] Y. Chen, B. Boehm, and L. Sheppard, "Value driven security threat modeling based on attack path analysis," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE, 2007, pp. 280a–280a.

[16] C. Chio and D. Freeman, *Machine learning and security: Protecting systems with data and algorithms*. " O'Reilly Media, Inc.", 2018.

[17] J. Clarke-Salt, *SQL injection attacks and defense*. Elsevier, 2009.

[18] K. Czarnecki, "Generative programming-principles and techniques of software engineering based on automated configuration and fragment-based component models," Ph.D. dissertation, Verlag nicht ermittelbar, 1999.

[19] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski, "Cool features and tough decisions: a comparison of variability modeling approaches," in *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*. ACM, 2012, pp. 173–182.

[20] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *International Conference on Software Product Lines*. Springer, 2004, pp. 266–283.

[21] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *International Workshop on Software Factories*. ACM San Diego, California, USA, 2005, pp. 16–20.

[22] D. Dasgupta, Z. Akhtar, and S. Sen, "Machine learning in cybersecurity: a comprehensive survey," *The Journal of Defense Modeling and Simulation*, vol. 19, no. 1, pp. 57–106, 2022.

[23] U. W. Eisenecker and K. Czarnecki, "Generative programmierung: wie man komponenten baut und nutzt," *iX*, vol. 2, pp. 126–132, 1999.

[24] J. Erickson, *Hacking: the art of exploitation*. No starch press, 2008.

[25] M. Eriksson, J. Börstler, and K. Borg, "The pluss approach–domain modeling with features, use cases and use case realizations," in *International Conference on Software Product Lines*. Springer, 2005, pp. 33–44.

[26] L. Floridi, *The fourth revolution: How the infosphere is reshaping human reality*. OUP Oxford, 2014.

[27] S. Forrest, A. Somayaji, and D. H. Ackley, "Building diverse computer systems," in *Proceedings. The Sixth Workshop on Hot Topics in Operating Systems (Cat. No. 97TB100133)*. IEEE, 1997, pp. 67–72.

[28] F. Freiling, I. Eusgeld, and R. Reussner, "Dependability metrics," *Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany*, 2008.

[29] D. Geer, R. Bace, P. Gutmann, P. Metzger, C. Pfleeger, J. Quarterman, and B. Scheier, "Cyberinsecurity: The cost of monopoly–how the dominance of microsoft's products poses a risk to security," *Computer & Communications Industry Association Report*, 2003. [Online]. Available: http://www.ccianet.org/papers/cyberinsecurity.pdf

[30] A. Gómez and I. Ramos, "Cardinality-based feature modeling and model-driven engineering: Fitting them together." *VaMoS*, vol. 37, pp. 61–68, 2010.

[31] T. Grandison and M. Sloman, "A survey of trust in internet applications," *IEEE Communications Surveys & Tutorials*, vol. 3, no. 4, pp. 2–16, 2000.

[32] M. L. Griss, "Software reuse architecture, process, and organization for business success," in *Proceedings of the Eighth Israeli Conference on Computer Systems and Software Engineering*. IEEE, 1997, pp. 86–89.

[33] M. L. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)*. IEEE, 1998, pp. 76–85.

[34] R. Heady, G. Luger, A. Maccabe, and M. Servilla, "The architecture of a network level intrusion detection system," Los Alamos National Lab.(LANL), Los Alamos, NM (United States); New Mexico ..., Tech. Rep., 1990.

[35] P. Höfner, R. Khedri, and B. Möller, "Feature algebra," in *Proceedings of the 14th International Symposium on Formal Methods (FM 2006)*, ser. Lecture Notes in Computer Science, J. Misra, T. Nipkow, and E. Sekerinski, Eds., vol. 4085. Springer, 2006, pp. 300–315.

[36] G. Irazoqui, M. S. IncI, T. Eisenbarth, and B. Sunar, "Know thy neighbor: Crypto library detection in cloud." *Proc. Priv. Enhancing Technol.*, vol. 2015, no. 1, pp. 25–40, 2015.

[37] I. Jacobson, "Business process reengineering with object technology," *Object Magazine*, vol. 4, no. 2, pp. 16–ff, 1994.

[38] I. Jacobson, M. Griss, and P. Jonsson, "Reuse-driven software engineering business (RSEB)," 1997.

[39] I. Jacobson, *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993.

[40] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[41] A. Jaquith, *Security metrics: replacing fear, uncertainty, and doubt*. Pearson Education, 2007.

[42] J. Jesan, "Threat modeling web-applications using stride average model," in *Computer Security Conference*, 2008.

[43] T. A. Johnson, *Cybersecurity: Protecting critical infrastructures from cyber attack and cyber warfare*. CRC Press, 2015.

[44] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.

[45] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-; oriented reuse method with domain-; specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 1, p. 143, 1998.

[46] J. S. Keller and M. Monks. (2015) Morgan stanley says data stolen in insider breach. [Online]. Available: https://www.bloomberg.com/news/articles/2015-01-05/morgan-stanley-says-data-for-350-000-wealth-clients-stolen

[47] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "Sok: Automated software diversity," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 276–291.

[48] D. C. Latham, "Department of defense trusted computer system evaluation criteria," *Department of Defense*, 1986.

[49] B. Littlewood and L. Strigini, "Validation of ultra-high dependability for software-based systems," in *Predictably Dependable Computing Systems*. Springer, 1995, pp. 473–493.

[50] Á. Longueira-Romero, R. Iglesias, D. Gonzalez, and I. Garitano, "How to quantify the security level of embedded systems? a taxonomy of security metrics," in *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, vol. 1. IEEE, 2020, pp. 153–158.

[51] M. R. Lyu, J.-H. Chen, and A. Avizienis, "Software diversity metrics and measurements," in *1992 Proceedings. The Sixteenth Annual International Computer Software and Applications Conference*. IEEE Computer Society, 1992, pp. 69–70.

[52] A. Manna, A. Sengupta, and C. Mazumdar, "A survey of trust models for enterprise information systems," *Procedia Computer Science*, vol. 85, pp. 527–534, 2016.

[53] H. Marmanis, *Algorithms of the intelligent web*, 2009.

[54] L. Metcalf and W. Casey, *Cybersecurity and applied mathematics*. Syngress, 2016.

[55] H. Mili, F. Mili, and A. Mili, "Reusing software: Issues and research directions," *IEEE transactions on Software Engineering*, vol. 21, no. 6, pp. 528–562, 1995.

[56] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*. Wiley Online Library, 2004, vol. 2.

[57] A. A. Neto, M. Kalinowski, A. Garcia, D. Winkler, and S. Biffl, "A preliminary comparison of using variability modeling approaches to represent experiment families," in *Proceedings of the Evaluation and Assessment on Software Engineering*. ACM, 2019, pp. 333–338.

[58] D. O'Brien, "Misconfigured firewall leads to la county health data breach," *Infosecurity Magazine*, April 2011. [Online]. Available: https://www.infosecurity-magazine.com/news/misconfigured-firewall-leads-to-la-county-health-data-breach/

[59] L. O'Murchu and F. P. Gutierrez, "The evolution of the fileless chick-fraud malware poweliks," Symantec, Tech. Rep., 2015.

[60] J. K. Ousterhout, *A philosophy of software design*. Yaknyam Press Palo Alto, CA, USA, 2018, vol. 98.

[61] P. L. Overbeek, "Common criteria for it security evaluation-update report," in *Information Security—the Next Decade*. Springer, 1995, pp. 41–49.

[62] J. N. Pelton, I. B. Singh, and I. B. Singh, *Digital defense: A cybersecurity primer*. Springer, 2015.

[63] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–35, 2016.

[64] P. Pohjalainen, "Feature oriented domain analysis expressions," in *InNordic Workshop on Model Driven Software Engineering (NW-MoDE'08), Reykjavik, Iceland*, 2008.

[65] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending feature diagrams with uml multiplicities," in *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, vol. 23, 2002, pp. 1–7.

[66] I. Ristic, *Bulletproof SSL and TLS: Understanding and deploying SSL/TLS and PKI to secure servers and web applications*. Feisty Duck, 2014.

[67] R. C. Seacord, *Secure Coding in C and C++*. Addison-Wesley, 2013.

[68] A. Shostack, *Threat modeling: Designing for security.* John Wiley & Sons, 2014.

[69] H. J. Sienkiewicz, *The art of cyber conflict.* Dog Ear Publishing, 2017.

[70] A. S. Sodiya, S. A. Onashoga, and B. Oladunjoye, "Threat modeling using fuzzy logic paradigm," *Informing Science: International Journal of an Emerging Transdiscipline*, vol. 4, no. 1, pp. 53–61, 2007.

[71] W. Stallings, *Computer security principles and practice*, 2015.

[72] M. Stevens, A. Lenstra, and B. De Weger, "Chosen-prefix collisions for md5 and colliding x. 509 certificates for different identities," in *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26.* Springer, 2007, pp. 1–22.

[73] D. Stuttard and M. Pinto, *The web application hacker's handbook: Finding and exploiting security flaws.* John Wiley & Sons, 2011.

[74] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in *2013 IEEE Symposium on Security and Privacy.* IEEE, 2013, pp. 48–62.

[75] Q. Tong, Y. Guo, H. Hu, W. Liu, G. Cheng, and L.-s. Li, "A diversity metric based study on the correlation between diversity and security," *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 10, pp. 1993–2003, 2019.

[76] B. Valeriano and R. C. Maness, *Cyber war versus cyber realities: Cyber conflict in the international system.* Oxford University Press, USA, 2015.

[77] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Proceedings Working IEEE/IFIP Conference on Software Architecture.* IEEE, 2001, pp. 45–54.

[78] L. Wang, S. Jajodia, and A. Singhal, *Network security metrics.* Springer, 2017.

[79] T. Winters, T. Manshreck, and H. Wright, *Software engineering at google: Lessons learned from programming over time.* O'Reilly Media, 2020.

[80] W. Yang, C. Huang, B. Wang, T. Wang, and Z. Zhang, "A general trust model based on trust algebra," in *2009 International Conference on Multimedia Information Networking and Security*, vol. 1. IEEE, 2009, pp. 125–129.

[81] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[82] M. Zalewski, *The tangled Web: A guide to securing modern web applications.* No Starch Press, 2011.

[83] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese, "Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 1071–1086, 2016.

[84] Q. Zhang, "Aspect-oriented product family modeling," Ph.D. dissertation, McMaster University, Hamilton, ON, Canada, June 2013.