# A FRAMEWORK FOR QUANTIFYING SECURITY EFFECTIVENESS OF CYBER DEFENSES

by

HUASHAN CHEN, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
October 2021

## DEDICATION

*In loving memory of my father, Mr. Wenwu Chen, who will be in my heart forever.*

*To my mother, Mrs. Yuexia Xi, for her tremendous love, patience and trust.*

*To my wife, Shanshan Ding, and her family, for their unconditional support.*

*To my sister, Xiaohong Chen, for her continuous encouragement.*

## ACKNOWLEDGEMENTS

I would like to express my most sincere gratitude and deepest appreciation to my advisor, Dr. Shouhuai Xu, for taking me into the challenging but exciting field of Cybersecurity. Without his great guidance, unlimited support, and continuous encouragement throughout my doctoral studies, this dissertation cannot be accomplished. His scientific attitude, holistic vision, creative thinking, and extreme enthusiasm on the research inspired me not only the way to approach a technical hard problem but also how to overcome difficulties in my life.

I would like to express my gratitude to my co-advisor Dr. Ravi Sandhu and other committee members Dr. Mimi Xie, Dr. Xiaoyin Wang and Dr. Gregory B. White for their insightful comments and helpful suggestions on the dissertation.

I would like to thank my lab colleagues Dr. Marcus Pendleton, Dr. Richard B. Garcia-Lebron, Dr. John Charlton, Zheyuan Sun, Eric Ficke, and Mir Mehedi Ahsan Pritom for their sharing of ideas. I have benefited a lot from my interactions with them.

I would like to thank Suzanne Tanaka, Susan Allen, Lisa Ho and other staff members from the ICS and the CS department at UTSA for their great help during my doctoral studies.

I would like to thank my friends Sen He and Jinfu Chen for their various help, and other friends at UTSA for their companionship during this wonderful journey.

This PhD Dissertation research was supported in part by ARO Grant #W911NF-17-1-0566 and NSF Grant #2122631 (#1814825).

*and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.*

October 2021

# A FRAMEWORK FOR QUANTIFYING SECURITY EFFECTIVENESS OF CYBER DEFENSES

Huashan Chen, Ph.D.
The University of Texas at San Antonio, 2021

Supervising Professors: Ravi Sandhu, Ph.D. and Shouhuai Xu, Ph.D.

Cybersecurity metrics and quantification is a holy-grail challenge that has yet to be tackled. While significant progress has been made in quantifying building-blocks security, the problem of quantifying security from a holistic perspective is largely open. One fundamental factor that makes the problem so hard is the dynamics phenomenon incurred by complex attacker-defender-user interactions in cyberspace, meaning that the networked system itself, the employed defense posture, the adversaries, the users behaviors, and the global cybersecurity state evolve with time.

This Dissertation makes a significant step towards ultimately understanding, characterizing, quantifying and managing cybersecurity from a holistic perspective, by proposing a high-fidelity simulation framework to model cyber attack-defense interactions while making weak assumptions. The framework falls under the Cybersecurity Dynamics approach, meaning that networks, users, attacks, defenses, and cybersecurity states all can evolve with time. The usefulness of the framework is demonstrated by three scenarios: quantifying security effectiveness of firewalls and DMZs; quantifying security effectiveness of coarse-grained dynamic network diversity; and quantifying security effectiveness of fine-grained static network diversity.

# TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1: INTRODUCTION

## 1.1 Background and Research Motivation

The physicist Lord Kelvin once said, "*If you cannot measure it, you cannot improve it*" [88]. This famous quote has been widely used by the cybersecurity community to highlight the importance of cybersecurity metrics. This is not surprising because if we cannot measure cybersecurity, we cannot quantify the progress we may have made. This has many real-world implications. For example, we cannot achieve quantitative cyber defense decision-making and cybersecurity risk management from a network-wide point of view without the support of cybersecurity metrics and quantification methods. As a consequence, there is a lack of body of knowledge that can be used by a Chief Security Officer (CSO) to answer questions that may be asked by a Chief Executive Officer (CEO), such as: What would be the gain in cybersecurity if we invest another $10M to enhance our cybersecurity posture? The cybersecurity community has been confronted with the preceding unpleasant situation for decades. In particular, the cybersecurity metrics problem has been highlighted by several lists of Cybersecurity Hard Problems [33, 101, 119].

Although it may be intuitive that measuring cybersecurity means, in general, to quantify the degree at which a system can be expected to perform its intended function under particular conditions of operation, including attacks [87], there is a lack of fundamental progress for many reasons. This is true despite many studies, such as [28, 30, 60, 103, 112, 131, 132, 165]. For example, studies show that we, the cybersecurity research community as a whole, do not really know what metrics must be defined and measured [32, 106, 157].

Recently, a systematic approach to modeling, quantifying and analyzing cybersecurity from a holistic perspective has emerged. The approach is known as Cybersecurity Dynamics [148, 149, 152, 153]. This approach is centered at, and driven by, cybersecurity metrics and quantification from a holistic perspective [32, 76, 98, 106, 154]. One key idea behind the Cybersecurity Dynamics approach is to embrace *dynamics*, which is inherent to cyberspace and cybersecurity. This is achieved by explicitly considering the *time-dimension* into every aspect of a cybersecurity models, including threats and defenses. Correspondingly, *time-dependent* metrics are needed to describe, among other things, the evolution of the global cybersecurity state of networks in questions. This explicit use of time-dimension is a paradigm shift [149]. This view treats the dynamics or evolutions in the cyber domain as “natural” phenomena, which are incurred by attacker-defense-user interactions in cyberspace. Another key idea behind the Cybersecurity Dynamics approach is to embrace *uncertainty*, which is also inherent to cyberspace and cybersecurity. For example, in order to answer the CEO’s question mentioned above, the estimated gain in cybersecurity does not have to, or should not, be a specific number; rather, it is likely a random variable with a certain statistical distribution. Such uncertainty has been embedded into many models (e.g., [55, 146, 147, 150, 155, 156, 168]). At the same time, leveraging uncertainty to achieve cybersecurity objectives have start to receive researchers’ attention as well [77].

The holistic perspective taken by the Cybersecurity Dynamics offers a great potential in “connecting the dots” or unifying the segmented cybersecurity knowledge into a unified framework [148, 149, 152, 153]. For example, software vulnerabilities are exploited to wage cyber attackers. Therefore, Cybersecurity Dynamics models aim to explicitly model software vulnerabilities. The question is: How should we quantify software vulnerabilities? In order to answer this question, we would need to consider, among other things, the capabilities of software vulnerability detectors, which is an active research topic [81–84, 169, 170] especially because the robustness of such detectors is still an open problem [85].

## 1.2 Research Focus of the Present Dissertation

Given that the motivating problem is so broad, the present Dissertation needs to have a focus. Specifically, this Dissertation aims to advance the state-of-the-art in the Cybersecurity Dynamics approach to modeling and quantifying cybersecurity from a holistic perspective. The advancement is to bridge the theoretical study of Cybersecurity Dynamics models (e.g., [55, 146, 147, 150, 155, 156, 168]) and the real world by making weak assumptions, which are arguably realistic.

Specifically, the Dissertation aims to answer two specific research questions:

- How can we quantify the security effectiveness of firewalls and DeMilitarized Zones (DMZs)?

  It is somewhat ironic that firewalls and DMZs have been widely used in cyber defense practice for many years but there is no scientific investigation on their effectiveness from a whole-network point of view until our study [23].

- How can we quantify the security effectiveness of enforcing network diversity?

  This problem is important because software monoculture has been a big factor that amplifies the effect or damage of cyber attacks. Despite the studies [117, 120, 124] on analyzing the effectiveness of software diversity techniques (e.g., $N$-version programming [10, 26], compiler-based diversification [35, 46, 61, 66, 73], software runtime environment diversification [11, 14, 41, 45, 70, 140]), there is no systematic study on how to employ diversified software implementations in order to achieve better cybersecurity from the perspective of looking at a network as a whole rather than from a building-block perspective.

The preceding two questions represent two complementary perspectives in the following sense: firewalls and DMZs are representative of *preventive* cyber defense mechanisms; whereas, network diversity can be *proactive* and/or *reactive* defense mechanisms [75, 150, 156, 168].

## 1.3 Dissertation Overview

### 1.3.1 A General Framework (Chapter 2)

In order to answer the preceding motivating questions, we propose a *simulation* framework for modeling attacker-defender-user interactions in networks. The framework is the first of its kind in terms of making weak (i.e., realistic) assumptions. This framework is described in Chapter 2, which falls under the Cybersecurity Dynamics approach mentioned above.

### 1.3.2 Quantifying Security Effectiveness of Firewalls and DMZs (Chapter 3)

Firewalls and DMZs are employed in almost every network to attempt to assure that authorized network traffic can traverse a network and unauthorized network traffic will be blocked. In order to quantify security effectiveness of firewalls and DMZs, we instantiate the proposed simulation framework described in Chapter 2 to model firewalls and DMZs, with emphasise of modelling the inter-computer communication relations within a network and the internal-external communication relations between the computers that reside inside the network and the computers that reside outside of the network. These two kinds of communication relations can accommodate the network traffic that should be inspected by firewalls and DMZs. Another salient feature of this framework is that it neither makes the independence assumption nor assumes any specific kind of dependence between the attack events, which goes much beyond the existing studies.

The framework further guides us to conduct systematic simulation experiments to answer the following Research Questions (RQs) with respect to the security effectiveness firewalls and DMZs, including:

- RQ1: How much security is gained by employing firewalls and/or DMZs?
- RQ2: Does firewalls and DMZs always lead to higher security? If not, when??
- RQ3: Which deployment leads to highest security among different combinations of firewalls and DMZs?
- RQ4: Which network components benefits most from employing firewalls and DMZs?

### 1.3.3 Quantifying Security Effectiveness of Network Diversity (Chapters 4-5)

Given diversified implementations of programs, there are several ways to employ them in a network. A simple method is to distribute diversified software implementations in the network-wide software stacks in some fashion and the deployment stays static or unchanged during the time horizon of interest; this called *static* diversity. In contrast, *dynamic* diversity aims to dynamically re-employ diversified programs at the computers' software stacks, where the employment (or configuration) does change during the time horizon of interest.

Due to the big difference between static diversity and dynamic diversity in employing diversified programs, we propose instantiating two independent frameworks to quantify their security effectiveness. Another factor that needs to be considered is the matter of *granularity*, as fine-grained studies would be much more involved than their coarse-grained counterparts. This is because where a coarse granularity means that each software program is treated as an "atom" in describing networked systems, whereas a fine granularity means that we treat individual applications, library functions, and operating system kernel functions as "atoms" in describing networked systems. This prompts us to:

- instantiate a coarse-grained framework to investigate the security effectiveness of dynamic network diversity. This coarse-grained modelling makes it easier to characterize the re-employment of diversified program implementations in dynamic network diversity.
- instantiate a fine-grained framework to investigate the security effectiveness of static network diversity. This granularity allows us to realistically model cyber attacks in a flexible manner because the *dependence* relations between the atomic objects can affect the attack consequences on diversified software programs.

In any case, the proposed frameworks further guide us to conduct systematic simulation experiments to quantify the security effectiveness of the diversity defenses.

In order to characterize the effectiveness of dynamic network diversity, Chapter 4 addresses the following Research Questions (RQs) by simulations:

- RQ1: To what extent can dynamic network diversity slow down the attacker?
- RQ2: How much extra cost can dynamic network diversity impose on the attacker?
- RQ3: To what extent can dynamic network diversity increase the defender's vulnerability tolerance?
- RQ4: To what extent can dynamic network diversity increase the defender's average operational cost?
- RQ5: Is it true that the more diversified implementations the better?

In order to characterize the effectiveness of static network diversity, Chapter 5 addresses the following Research Questions (RQs) by simulations:

- RQ1: Does natural diversity always lead to higher security? If not, when?
- RQ2: Does artificial diversity always lead to higher security? If not, when?
- RQ3: Does the use of natural and artificial diversity together always lead to higher security? If not, when?
- RQ4: What are the most effective defense strategies in the presence of network diversity?

## 1.4 Dissertation Organization

The reminder of this dissertation is organized as follows. Chapter 2 propose the overview of our framework. Chapter 3 uses the framework to quantify the security effectiveness of firewalls and DMZ. Chapter 4 uses the framework to quantify the security effectiveness of coarse-grained dynamic network diversity. Chapter 5 uses the framework to quantify the security effectiveness of fine-grained static network diversity. Finally, Chapter 6 concludes the Dissertation with future research directions.

# CHAPTER 2: A HIGH-FIDELITY SIMULATION FRAMEWORK

## 2.1 Chapter Introduction

Any approach that aims to understand cybersecurity from a holistic perspective needs to model the interactions between the various kinds of cyber attacks and the various kinds of cyber defenses in networked systems, including critical infrastructures and cyber-physical-human systems. This view is proposed by the Cybersecurity Dynamics approach [148, 149, 152, 153], which aims to model and quantify cybersecurity from a holistic perspective [32, 76, 106, 154].

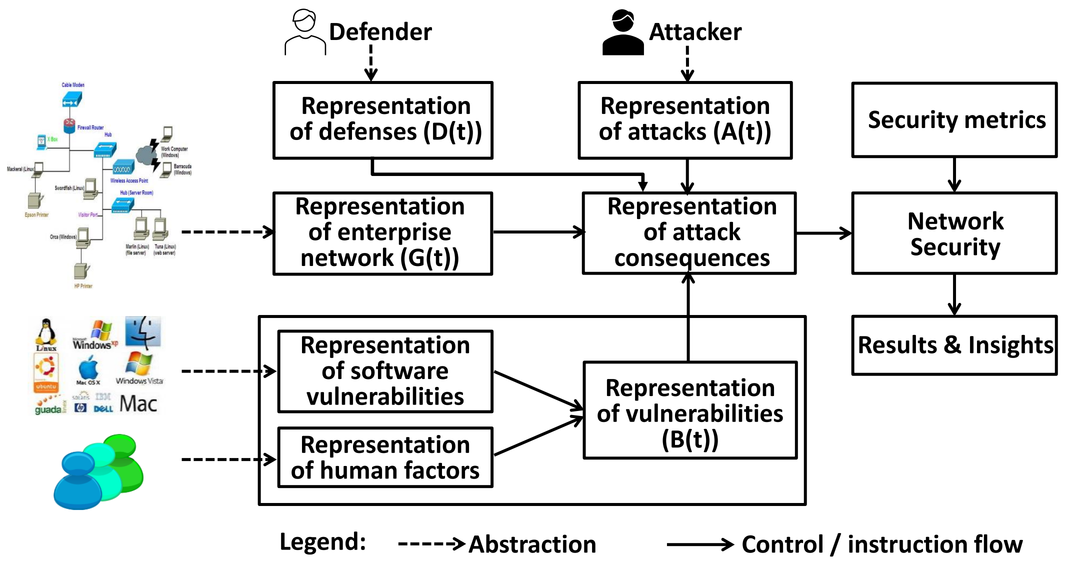

Defender
Attacker
Representation of defenses (D(t))
Representation of attacks (A(t))
Security metrics
Representation of enterprise network (G(t))
Representation of attack consequences
Network Security
Representation of software vulnerabilities
Representation of vulnerabilities (B(t))
Results & Insights
Representation of human factors
Legend:
Abstraction
Control / instruction flow

**Figure 2.1**: A framework for systematically modelling cyber attack-defense interaction, which refines an earlier framework we proposed in [24].

Figure 2.1 highlights the framework that will guide the study of this Dissertation. The framework falls under, and enriches, the Cybersecurity Dynamics approach by focusing on quantifying the security effectiveness of defenses in networked systems. The framework has five components in terms of: (i) abstracting networked systems, including traditional enterprise IT infrastructures, cloud/edge/end computing systems, cyber physical systems, cyber-physical-human systems, and critical infrastructures; (ii) representing vulnerabilities in networked systems, including the vulnerabilities of software stacks and human users' vulnerabilities to social engineering cyber attacks and insider threats; (iii) representing various kinds of attacks (i.e., threat models) against networked systems including malware propagation, DDoS attacks, and targeted APT attacks; (iv) representing the various kinds of defense mechanisms that are employed to protect networked systems including preventive defenses, reactive defenses, proactive defenses, and active defenses; and (v) representing the outcome of attack-defense interaction.

Correspondingly, five groups of cybersecurity metrics are defined: (i) metrics for describing networked systems including its configurations; (ii) metrics for describing software vulnerabilities and human vulnerabilities in networked systems; (iii) metrics for describing cyber attacks against networked systems; (iv) metrics for describing defenses employed to protect networked systems; and (v) metrics for describing the global cybersecurity state or cybersecurity situational awareness of networked systems.

Following [75, 89, 148–150, 152, 153, 155, 156, 167, 168], the high-level idea of Cybersecurity Dynamics is described as follows. Let $G(t)$ denote a networked system (e.g., enterprise network as mentioned above) of interest at time $t$ (including its hardware and software configurations), $B(t)$ denote the vulnerabilities in the network at time $t$ (including known and possibly zero-day software vulnerabilities, human factors with uncertainty), $A(t)$ denote the attacks that are waged against the network at time $t$, $D(t)$ denote the defense posture at time $t$ (i.e., the defense that are employed at time t to protect the network), and $M = \{m_i\}$ denote a set of rigorously defined security metrics of interest. We will discuss how $G(t)$, $B(t)$, $A(t)$, $D(t)$ and $M$ are represented later in this section because they will be geared towards specific studies, owing to the complexity of modeling everything at this stage of our understanding (as illustrated by the journey in understanding just one kind of Cybersecurity Dynamics [75, 156, 168]). In principle, there exists a family of mathematical functions $\mathcal{F}_i$ for computing a network's security in terms of metric $m_i \in M$, namely

$$m_i(t) = \mathcal{F}_i(G(t), B(t), A(t), D(t)). \tag{2.1}$$

Intuitively, $m_i$ reflects the outcome of the interaction between attacks $A(t)$ and defenses $D(t)$ in a network $G(t)$ with vulnerabilities $B(t)$. This allows us to compare the global cybersecurity of:

networks with two different configurations, say $G(t)$ vs. $G^{'}(t)$; networks with two different groups of vulnerabilities, say $B(t)$ vs. $B^{'}(t)$; networks in the presence of two different threat models, say $A(t)$ vs. $A^{'}(t)$; networks with two different defense postures, say $D(t)$ vs. $D^{'}(t)$. These differences are measured by the corresponding evolution of security metrics $m_i(t)$ and $m_i^{'}(t)$ over time, where $m_i \in M$. For example, we can quantify the security effectiveness of firewalls by comparing the security effectiveness achieved by two filtering rule configurations, say $D(t)$ and $D^{'}(t)$, namely

$$\mathcal{F}_i(G(t), B(t), A(t), D(t)) \quad \text{vs.} \quad \mathcal{F}_i(G(t), B(t), A(t), D^{'}(t)).$$

In what follows, we discuss how to obtain mathematical representations of network configurations $G(t)$, vulnerabilities $B(t)$, threats $A(t)$, and defense postures $D(t)$. These representations naturally lead to quantitative metrics $M$.

## 2.2 Representations

### 2.2.1 Representation of Networked Systems

A networked system is typically made up of some interconnected computers, which exchange messages through physical links. Each computer may run multiple software for certain functionality. In principle, we can use an undirected graph to abstract a real-world networked system because communication is often two-way. In the rare case that one-way communication is relevant, directed graph can be used instead. Specifically, a evolving network can be modeled as a time-dependent graph $G(t) = (V(t), E(t))$, where $V(t)$ is the node or vertex set at time $t$ and $E(t)$ is the edge or arc set at time $t$. A node $v \in V(t)$ represents an "atom" which could be, for example, a computer or software component. An edge or arc $(u, v) \in E(t)$ means that there exits a path from node $u$ to node $v$, which may be used by the attacker for compromising other nodes.

Network can be represented at different granularity, meaning that $V(t)$ and $E(t)$ may stands for different meanings at different granularity. At a coarse granularity, each node $v \in V(t)$ represents a computer and one edge represents the wired or wireless communication link between two computers accordingly. At a finer granularity, each node $v \in V(t)$ may represent a software component (e.g., application program, operating system). In this case, an edge between two software components running on different computers represents the communication relation between them and an edge between two software components running on a computer may represent the caller-callee dependency relation between them [23, 24].

Having obtained the graph-theoretic representation $G(t) = (V(t), E(t))$ of interest, we may define metrics to characterize $G(t)$. For example, we may use node degree distribution and giant component to characterize the structure of $G(t)$; we may use defective edges to characterize the software configuration of $G(t)$; we may use algebraic connectivity to characterize the robustness of $G(t)$.

### 2.2.2 Representation of Vulnerabilities

We propose considering two types of vulnerabilities: software vulnerabilities and human vulnerabilities, both of which are widely used in a broad sense. To be specific, a software vulnerability refer to a weakness that can be exploited by an attacker within a computer system, which means that software vulnerability may occur in the entire software stack, including applications, libraries, and operating systems. Human vulnerabilities, on the other hand, refer to the vulnerabilities caused by the users, such as vulnerabilities to social-engineering attacks (e.g., phishing) as well as insider threats and the vulnerabilities caused by the use of weak passwords.

Each vulnerability may be associated with a set of attributes. For example, a software vulnerability may have the following attributes: (i) attack vector, which reflects the context by which vulnerability exploitation is possible; (ii)attack complexity, which describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability; (iii) privileges required, which describes the level of privileges an attacker must possess before successfully exploiting the vulnerability; and (iv)user interaction, which captures the requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component.

There are some public resources that can be considered to define metrics for measuring software vulnerabilities. For example, the Common Vulnerability Scoring System (CVSS) [96] is an open framework for communicating the characteristics and severity of software vulnerabilities. In terms of human vulnerabilities, they may be described by the chance that a user is vulnerable to social-engineering attacks.

### 2.2.3 Representation of Attacks

We propose to represent cyber attacks from the following perspectives: (i) attacker's *knowledge*, which describes what an attacker may know at time $t$, including cybersecurity metrics, cybersecurity models, cybersecurity state, social science knowledge (e.g., human factors), and operator experience. For example, an attacker may incorporate information of the network to build a formal representation or model of the network, which is stored in the knowledge base and is directly accessible for inference and planning. (ii) attacker's *goals*, which describes what an attacker attempts or aims to accomplish at time $t$. For example, an attacker's goal may be to compromise as many computers as possible, or to compromise certain computers of interest (e.g., APT). (iii) attacker's *strategies*, which specify a procedure of multiple phases (or steps) for the purpose of achieving the attacker goal. Examples include the Cyber Kill Chain [65] and MITRE ATT&CK [3]. (iv) attacker's *capabilities*, which describes the actions that can be taken by an attacker at time $t$. Examples include the set of exploits that are available to the attackers. (v) attacker's *decision-making*, which dynamically generates, chooses and organizes attack plans for its goal by anticipating their outcomes during the course of attack-defense interactions. Examples include immediate selection (e.g, random selection, greedy selection and finite-state machine selection) and automatic planning (e.g., classical planning, probabilistic planning and partially-observable planning). (vi) attacker's *execution*, which describes how the dynamic decisions made by an attacker decision-making engine may be executed. Examples include centralized Command and Control and distributed multi-agent systems.

Many kinds of metrics can be defined to measure attacks, such as (i) *attack success probability*, which refers to the probability that attacks are successfully performed; (ii) *mean time to compro-*

*mise a system*, which indicates how long an attacker takes to compromise an entire system; (iii) *attack complexity*, which describes the number of phases an attack goes through; (iv) *attack efforts*, which describes the how much overhead and/or impact is introduced to attackers to perform their attacks.

### 2.2.4 Representation of Defenses

We propose to model cyber defenses from the following perspectives: (i) defender's *knowledge*, which describes what a defender may know about the networked system at time $t$. Different from an attacker who may possess a nearly-empty view of the network initially, a defender can possess near-complete information of the networked system. (ii) defender's *goals*, which describes what a defender attempts or aims to accomplish at time $t$. For example, a defender's goal may be to guarantee some critical resources from being compromised. (iii) defender's *strategies*, which specify what defense action should be taken under what state or condition at time $t$. For example, the defender may refer to NIST's Cybersecurity Framework [12] to guide their defenses. (iv) defender's *capabilities*, which describes the set of defense tools (e.g., using new tools or updating the attack signatures or detection models of certain tools) that can be employed by the defender at time $t$. (v) defender's *decision-making*, which describes the set of orientation algorithms employed by the defenders at time $t$. (vi) defender's *execution*, which describes how the dynamic decisions made by a defender's decision-making engine may be deployed.

Various types of the defender's metrics can be defined to measure defenses, such as (i) *defense success probability*, which estimated the rate of executing successful defenses (e.g., for a defender, the rate at which tasks are executed and completed); (ii) *mean time to failure*, which refers to a system reliability metric capturing the system's up-time in the presence of attacks when failures can happen due to either defects or security threats; and (iii) *defense costs*, which describes the amount of extra efforts the defender paid and the damage on the system functioning caused by the defenses.

### 2.2.5 Representation of Cybersecurity State

For any model resolution (e.g., treating a computer/device as an atom vs. treating a software component as an atom), the security state of an "atom" can be in one of multiple states, such as *secure*, *vulnerable*, or *compromised*, denoted by

$$\mathsf{state}(v,t) = \begin{cases} 0 & \text{node } v \in V \text{ is in secure state at time } t \\ 1 & \text{node } v \in V \text{ is in vulnerable state at time } t \\ 2 & \text{node } v \in V \text{ is in compromised state at time } t \end{cases}$$

Therefore, at any point in time, the *global* cybersecurity state can be defined by four time-dependent metrics: *percentage of compromised atoms* at time $t$; *percentage of cumulative compromised atoms* at time $t$, namely atoms that have been compromised at least once till time $t$; *percentage of vulnerable atoms* at time $t$ and *percentage of secure atoms* at time $t$.

## 2.3 Chapter Summary

In this Chapter, we present the overall architecture of the time-dependent framework for systematically modelling cyber attack-defense interactions. We further discuss the how to obtain mathematical representations of each component in the framework as well as the security metrics. In what follows, we will demonstrate how to instantiate the framework to quantify the security effectiveness of three kinds of cyber defenses: firewalls and DMZ, dynamic network diversity, and static network diversity.

# CHAPTER 3: QUANTIFYING SECURITY EFFECTIVENESS OF FIREWALLS AND DMZS

## 3.1 Chapter Introduction

Firewalls and DMZs are two widely employed security mechanisms. On one hand, firewalls enforce security policies to filter or block unauthorized communication traffic between the outside network and an enterprise network or between the sub-networks within an enterprise network. The filtering operation can be conducted at the packet layer, the flow layer (i.e., examining flow-level content), and the application layer (i.e., inspecting application-layer data) [6, 64]. For the purpose of the present study, we focus on the functionality of firewalls in filtering unauthorized communication traffic, while safely assuming away the implementation details (e.g., the layers at which filtering is conducted). On the other hand, DMZs isolate the external network from an enterprise network while providing the external users with interfaces to access the enterprise's Internet-facing servers (e.g., websites and email servers). Intuitively, DMZs may slow down, or even prevent, some attacks against enterprise networks.

Despite the wide use of firewalls and DMZs, their security effectiveness has yet to be quantified and characterized. To the best of our knowledge, no prior studies have aimed at systematically answering the following question: *How much security is gained by employing firewalls and/or DMZs?* The void of this knowledge motivates the present study. The existence of the void is true despite that quantifying security is one of the well-recognized open problems [102, 106, 113]. Indeed, the importance of quantifying security has led to industrial efforts that focus on software vulnerabilities (e.g., the Common Vulnerability Scoring System or CVSS [105]) and academic investigations that treat an entire network as a whole (e.g., [36, 79, 113, 136]). However, the aforementioned motivational question remains unaddressed.

### 3.1.1 Chapter Contributions

This Chapter makes a first step towards quantifying the security effectiveness of firewalls and DMZs, by making two contributions.

First, we instantiate the proposed framework for modeling firewalls and DMZs in protecting enterprise networks, while treating software components as “atoms” in describing enterprise networks. Compared with the existing studies that aim to quantify security by treating an entire network as a whole, the present study have two salient features. (i) Existing studies often make the *independence* assumption between the attack events. For example, attacks against a victim (e.g., computer) are assumed to be independently waged by multiple compromised computers [20, 79, 136, 150, 156]. Although there have been some efforts to weaken the assumed independence [36,146,147], they can only accommodate some specific kinds of *dependence* rather than completely eliminating the matter of independence. The present study neither makes the independence assumption (unlike [20, 136, 150, 156]) nor assumes any specific kind of dependence (unlike [36,146,147]). We achieve this by developing a framework to allow for simulation studies, rather than for analytic treatment. (ii) The present study accommodates the threat models known as Lockheed Martin’s Cyber Kill Chain [65] and Mandiant’s Attack Life Cycle [95]. These threat models accommodate realistic attacks that are not considered in the existing studies mentioned above, which investigate epidemic spreading over *arbitrary* network structures [20, 136, 150, 156] or the more general notion of cybersecurity dynamics [56, 152, 167, 168].

Second, the framework guides us to conduct systematic simulation experiments. Our preliminary experiments lead to the following findings. (i) When the applications and operating systems (OSes) have few or too many vulnerabilities, firewalls and DMZ do not have a significant impact on security. This is because in the former case, the network cannot be attacked, with or without employing firewalls and DMZs, and in the latter case, these defense mechanisms cannot prevent attacks from succeeding. (ii) When the OSes are not vulnerable but the applications are, security effectiveness of firewalls and DMZs decreases as the fraction of vulnerable applications increases. (iii) When effective, employing perimeter firewalls alone has little impact on security, but further employing DMZ and internal firewalls (to separate an enterprise network into smaller ones) will substantially increase security. This justifies the practice of employing both DMZ and firewalls. (iv) When effective, employing perimeter firewall and DMZ increases security of the sever applications.

### 3.1.2 Related Work

As discussed in the Introduction, quantifying security is one of the most fundamental open problems [102, 106, 113]. The present work moves a further step beyond existing studies [36, 79, 136, 146, 147, 152] by getting rid of the independence assumption and by accommodating a new class of threats [65, 95]. To make the comparison fair, we should note that these gains in modeling capacity are obtained at the price of using simulations to characterize the security effectiveness of firewalls and DMZs. Nevertheless, the present study leads to new insights, such as those mentioned above and those will be reported in the full version of the present work, that are not known until now. This can be attributed to the following fact: existing studies often use some parameters (e.g., probabilities) to abstract the capability of defense mechanisms such as firewalls and DMZs; in contrast, we aim to precisely quantify and characterize the security effectiveness of firewalls and DMZs by treating an entire enterprise network as a whole.

To the best of our knowledge, the present study initiates the investigation of how to quantify the security effectiveness of firewalls and DMZs. This can be justified by the fact that based on recent surveys [103, 106, 113], there are neither metrics nor models for explicitly measuring the security effectiveness of firewalls and DMZs by treating a network as a whole.

### 3.1.3 Chapter Organization

The rest of this Chapter is structured as follows. Section 3.2 instantiates the proposed framework. Section 3.3 presents the simulation experiments and the resulting insights. Section 3.4 concludes the work with open problems for future research.

## 3.2 Instantiating the Framework to Quantify Security Effectiveness of Firewalls and DMZs

In order to quantify the security effects of firewalls and DMZs, we propose a framework for describing enterprise networks. The framework has the following components: (i) abstract representation of an enterprise network, including the software stacks of the computers beloning to the network and the communication relations between the applications; (ii) abstract representation of the vulnerabilities in the software stacks and the vulnerabilities of human users; (iii) abstract representation of the defense mechanisms that are employed to protect an enterprise network; (iv) abstract representation of attacks against an enterprise network; (v) definitation of security metrics for measuring the security effects of firewalls and DMZs.

Table 3.1 summarizes the notations that are used in the rest of the Chapter.

### 3.2.1 Representation of Enterprise Networks

An enterprise network consists of a number of computers (including both user computers and servers), each of which runs a *software stack*. Moroever, the computers within an enterprise network may communicate with each other through some applications, and the computers within an enterprise network may communicate with computers in the outside network through some applications.

#### Representation of Software Stacks

A computer runs a software stack. In this work, we consider two layers of the software stack: the *application* layer and the *OS* layer. An operating system (OS) not only contains the kernel functions but also contains the device drivers. In what follows we elaborate on the representations of applications and OSes, respectively.

**Representation of applications**. An application (or application program) consists not only the code of an application program, but also the code of the software components upon which they

| | |
|---|---|
| $\mathsf{APP}$ | the universe of applications |
| $\mathsf{OS}$ | the universe of operating systems |
| $\eta$ | $\eta : \mathsf{APP} \to \{0,1,2\}$ indicates the types of applications: client ('0'), Internet-facing server ('1'), or internal server ('2') |
| $G_i$ | $G_i = (V_i, E_i)$ is a computer, where $V_i = V_{i,app} \cup V_{i,os}$ and $E_i = E_{i,aa} \cup E_{i,af} \cup E_{i,ff}$ |
| $G = (V, E)$ | $G$ is an enterprise network of $n$ computers, where $V = V_1 \cup \ldots \cup V_n$ and $E = E_1 \cup \ldots \cup E_n \cup E_0 \cup E_*$ |
| $G' = (V', E')$ | $G'$ is the attacker's view of the target network $G = (V, E)$ after reconnaissance process where $V' \subseteq V$ and $E' \subseteq E$ |
| $\mathsf{VUL}$ | the universe of software vulnerabilities |
| $\phi(v)$ | the set of vulnerabilities contained in node $v \in V$ |
| $\psi(v)$ | the probability that the user of computer $i$ is vulnerable to social engineering attacks where $v \in V_i$ |
| $\mathsf{loc}(\mathsf{vul})$ | whether the exploitation of $\mathsf{vul} \in \mathsf{VUL}$ requires local access ('0') or not ('1') |
| $\mathsf{zd}(\mathsf{vul})$ | whether $\mathsf{vul} \in \mathsf{VUL}$ is known ('0') or zero-day ('1') |
| $\mathsf{priv}(\mathsf{vul})$ | whether the exploitation of $\mathsf{vul} \in \mathsf{VUL}$ causes the attacker to get the `root` privilege ('1') or not ('0') |
| $A$ | the number of zero-day vulnerabilities in $G$ |
| $K$ | the number of known vulnerabilities in $G$ |
| $B$ | the number of known vulnerabilities against which the exploits can be detected and blocked in $G$ |
| $C$ | the number of known vulnerabilities against which the exploits cannot be blocked in $G$ |
| $k$ | the fraction of known vulnerabilities can be prevented from being exploited by NIPS |
| $\alpha$ | the probability a social engineering attack is blocked |
| HIPS | the employment policy of HIPS |
| $\zeta$ | the probability privilege escalation attempts are blocked by HIPS |
| $X$ | the set of exploits that are available to the attacker |
| $\rho(x, \mathsf{vul})$ | the probability $x \in X$ successfully exploits $\mathsf{vul} \in \mathsf{VUL}$ |
| $(a, b, c)$ | the percentage of vulnerabilities that can be exploited by the attacker corresponding to $(A, B, C)$ |
| $\omega$ | $\omega = \lvert V'\rvert / \lvert V\rvert$ is the fraction of nodes the attacker can discover by initial reconnaissance |
| $p_1$ | the probability a desktop runs a FTP client application |
| $p_2$ | the probability a desktop runs a database client application |
| $\delta$ | the probability that each OS function is called by each application |
| $\beta$ | the probability that each application contains a vulnerability |
| $N$ | the number of vulnerabilities in the OSes |
| $\vartheta(\mathsf{vul})$ | the probability $\mathsf{vul} \in \mathsf{VUL}$ is remotely exploitable |
| $\tau(\mathsf{vul})$ | the probability $\mathsf{vul} \in \mathsf{VUL}$ is zero-day |
| $\gamma$ | the firewalls and DMZ employment |
| $\mathsf{pca}(t), \mathsf{pcos}(t)$ | % of compromised applications and operating systems at time $t$, respectively |

**Table 3.1**: Summary of key notations used for quantifying security effectiveness of firewalls and DMZ.

depend (e.g., the libraries that are called by an application program). An application and the software components upon which it depends are treated as a unit because they run with the same pivilege and they are loaded into the same memory space. We consider each application as an atomic object, because (i) each process running in the user-space memory is an instance of an application; (ii) a vulnerable application can be an entry-point for a remote attacker to penetrate into a computer (e.g., remote code execution); (iii) an application is a privilege entity in the sense that if any part of an application is compromised, the entire application is compromised; and (iv) a system call from a compromised application can cause the compromise of OS.

Let $\mathsf{APP}$ denote the universe of applications running in an enterprise network consisting of $n$ computers. For computer $i$ in the network, we denote by $\mathsf{app}_{i,z}$ the $z$-th application running on computer $i$, where $\mathsf{app}_{i,z} \in \mathsf{APP}$.

We propose classifying applications into the following two kinds: client applications (e.g., browsers, email clients) and server applications (e.g., web servers, email servers, SQL servers). Moreover, server applications can be further divided into two kinds: Internet-facing servers (i.e., these servers are accessile from the Internet) and internal servers (i.e., these servers can be accessed by the computers in the enterprise network but cannot be accessed from the Internet). In order to represent this attribute, we define the following mathematical function:

$$\eta : \mathsf{APP} \rightarrow \{0, 1, 2\} \tag{3.1}$$

such that

$$\eta(\mathsf{app}) = \begin{cases} 0 & \mathsf{app} \in \mathsf{APP} \text{ is a client application} \\ 1 & \mathsf{app} \in \mathsf{APP} \text{ is an Internet-facing server application} \\ 2 & \mathsf{app} \in \mathsf{APP} \text{ is an internal server application.} \end{cases}$$

This classification is useful because different kinds of applications have differen security implications. For example, a client application may be vulnerable to social engineering attacks, but a server application may be compromised by the exploitation of software vulnerabilities. Moreover, an external attacker may be able to directly compromise an Internet-facing server, but cannot directly compromise an internal server unless the attacker already penetrated into the enterprise network.

**Representation of OSes**. An OS is a software running in the kernel space. It manages computer hardware and software resources and provides services to the applications running on top of it. We propose treating each OS function, rather than the entire OS, as an "atomic" entity because (i) applications often make system calls (syscall) to invode some OS functions; (ii) a vulnerable OS is not compromised unless a vulnerable OS function is exploited by a syscall incurred by a compromised application.

Let $\mathsf{OS}$ denote the universe of OSes running in an enterprise network consisting of $n$ computers. For computer $i$ in the network, we denote by $\mathsf{os}_i$ the OS running on computer $i$ and by $f_{i,z} \in \mathsf{os}_i$ the $z$-th OS (i.e., kernel or device driver) function in $\mathsf{os}_i$.

## Representation of Computers

Having abstracted the software stack running on a computer as two layers (i.e., the application layer and the OS layer), we now describe how to represent computers in an enterprise network. In order to represent a computer, we also need to accommodate the security-related relations between the "atomic" entities in a computer. Specifically, we propose accommodating the following two kinds of relations, respectively dubbed the *dependence* relation and the *inter-application communication* relation.

- The *dependence* relation represents the *caller-callee* relation between two software programs running on the same computer, including the caller-callee relations between applications and OS functions and the caller-callee relations between two different OS functions. For example, an application may make a syscall, which may further call anoter OS function. The dependence relation should be accommodated because a vulnerability in an atomic entite on the caller-callee sequence can cause a successful exploitation.
- The *inter-application communication* relation represents the communications between two

applications running on the same computer. This relation should be accommodated because it can be exploited to wage attacks. For example, if one application is allowed to communicate with another application, then the compromise of the former can cause the compromise of the latter, assuming the latter has a vulnerability that can be exploited.

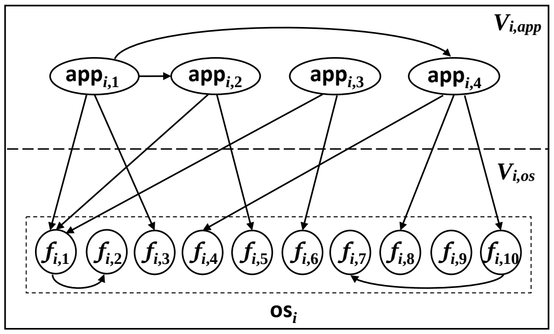


**Figure 3.1**: A graph-theoretic representation of computer $i$, namely $G_i = (V_i, E_i)$, in an enterprise network of $n$ computers.

Fig. 3.1 illustrates a toy example of computer $i$, which runs four applications that are respectively denoted by $\mathsf{app}_{i,1}$, $\mathsf{app}_{i,2}$, $\mathsf{app}_{i,3}$ and $\mathsf{app}_{i,4}$. Let $V_{i,app}$ denote the set of application running on computer $i$. In this example, we have

$$V_{i,app} = \{\mathsf{app}_{i,1}, \mathsf{app}_{i,2}, \mathsf{app}_{i,3}, \mathsf{app}_{i,4}\}. \tag{3.2}$$

Let $V_{i,os}$ denote the OS of computer $i$. In this example, the OSes has 10 functions, namely

$$V_{i,os} = \mathsf{os}_i = \{f_{i,1}, f_{i,2}, f_{i,3}, \ldots, f_{i,9}, f_{i,10}\}. \tag{3.3}$$

Let us define the node set

$$V_i = V_{i,app} \cup V_{i,os}, \tag{3.4}$$

where $V_{i,app}$ and $V_{i,os}$ are respectively given by Eqs. (3.2) and (3.3). Let us also define the arc set

$$E_i = E_{i,af} \cup E_{i,ff} \cup E_{i,aa}, \tag{3.5}$$

where

- $E_{i,af}$ represents the *dependence* relation between the applications and OS functions running on computer $i$. For example, in Fig. 3.1 we have $E_{i,af} = \{(\mathsf{app}_{i,1}, f_{i,1}),$ $(\mathsf{app}_{i,1}, f_{i,3}), (\mathsf{app}_{i,2}, f_{i,1}), (\mathsf{app}_{i,2}, f_{i,5}), (\mathsf{app}_{i,3}, f_{i,1}),$ $(\mathsf{app}_{i,3}, f_{i,6}), (\mathsf{app}_{i,4}, f_{i,4}), (\mathsf{app}_{i,4}, f_{i,8}), (\mathsf{app}_{i,4}, f_{i,10})\}$.

- $E_{i,ff}$ represents the *dependence* relation between OS functions running on computer $i$. For example, in Fig. 3.1 we have $E_{i,ff} = \{(f_{i,1}, f_{i,2}), (f_{i,10}, f_{i,7})\}$ because $f_{i,1}$ calls $f_{i,2}$ and $f_{i,10}$ calls $f_{i,7}$.

- $E_{i,aa}$ represents the inter-application communication relation between the applications running on computer $i$. For example, in Fig. 3.1 we have $E_{i,aa} = \{(\mathsf{app}_{i,1}, \mathsf{app}_{i,2}),$ $(\mathsf{app}_{i,1}, \mathsf{app}_{i,4})\}$ meaning that $\mathsf{app}_{i,1}$ can initiate communications with $\mathsf{app}_{i,2}$ and $\mathsf{app}_{i,4}$ and that the compromise of $\mathsf{app}_{i,1}$ can cause the compromise of $\mathsf{app}_{i,2}$ and $\mathsf{app}_{i,4}$, assuming the latter have vulnerabilities that can be exploited from $\mathsf{app}_{i,1}$.

By putting the aforementioned pieces together, we propose modeling computer $i$ as a graph

$$G_i = (V_i, E_i) \tag{3.6}$$

where $V_i$ and $E_i$ are respectively defined in Eqs. (3.4) and (3.5).

**Representation of the inter-computer communication relation in a network**

The *inter-computer communication* relation describes which applications running on one computer can communicate with which other applications running on the other computers. We propose accommodating this relation because it can represent how attacks may be waged from one compromised computer to another vulnerable one. It is important to model the inter-computer communication relation because inter-computer communications are often monitored by firewalls.

Fig. 3.2 illustrates the inter-computer communication relation between computer $i$ and computer $j$, which are respectively described by graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ as mention

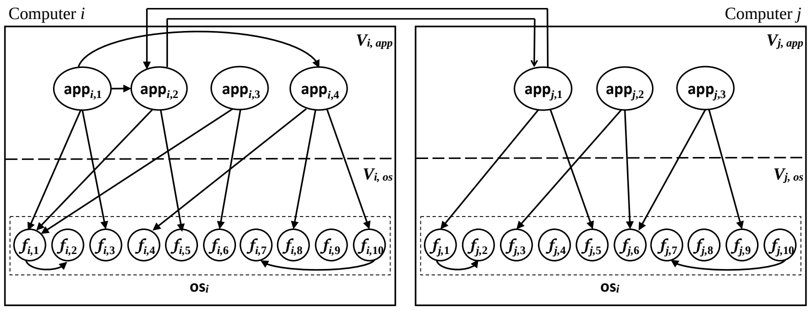


**Figure 3.2**: Illustration of the inter-computer communication relation.

above. We use an arc set $E_0$ to represent the inter-computer communication relation between the applications runnin on computer $i$ and the applications running on computer $j$, namely

$$E_0 \subseteq \{V_{i,app} \times V_{j,app}\} \cup \{V_{j,app} \times V_{i,app}\}, \tag{3.7}$$

where $1 \le i, j \le n$ and $i \ne j$. In the example illustrated in Fig. 3.2, $\mathsf{app}_{i,2}$ running on computer $i$ is allowed to communicate with $\mathsf{app}_{j,1}$ running on computer $j$. Suppose $\mathsf{app}_{i,2}$ and $\mathsf{app}_{j,1}$ are respectively a browser and a web server, we have

$$E_0 = \{(\mathsf{app}_{i,2}, \mathsf{app}_{j,1}), (\mathsf{app}_{j,1}, \mathsf{app}_{i,2})\},$$

where arc $(\mathsf{app}_{i,2}, \mathsf{app}_{j,1})$ indicates that a vulnerable web server (i.e., $\mathsf{app}_{j,1}$) can be compromised by attacks that are waged from a browser (i.e., $\mathsf{app}_{i,2}$), and arc $(\mathsf{app}_{j,1}, \mathsf{app}_{i,2})$ means a vulnerable browser (i.e., $\mathsf{app}_{i,2}$) can be compromised by a malicious web server (i.e., $\mathsf{app}_{j,1}$) via the "drive by" download attack.

In general, we propose distinguishing the afore-mentioned two kinds of attack activities by partitioning $E_0$ into $E_{00}$ and $E_{01}$, such that $E_0 = E_{00} \cup E_{01}$, $E_{00}$ represents the attacks against clients (including peers in peer-to-peer application), and $E_{01}$ represents the attacks against servers. More specifically, we have:

- $(\mathsf{app}_{j,y}, \mathsf{app}_{i,x}) \in E_{00}$ can be abused to launch attacks from a server or client or peer application $\mathsf{app}_{j,y}$ against a client application $\mathsf{app}_{i,x}$, where $\eta(\mathsf{app}_{i,x}) = 0$.

- $E_{01} = E_0 \setminus E_{00}$: Any inter-computer communication other than what are accommodated by $E_{00}$.

We stress that $e \in E_0$ often corresponds to a commuication or routing path, rather than a physical communication link unless the two communication end-points reside in the same sub-network or local area network (which is networked via a swtich rather than a router).

**Representation of the internal-external communication relation**

The computers in an enterprise network often need to communicate with some computers outside of the enterprise network. Similarly, some computers outside of the network often need to communicate with some computers residing in an enterprise network. It is clear that these communications have security consequences, meaning that they can be leveraged to wage attacks. The *internal-external* communication relation aims to accommodate these communications.

Formally, we use arc set $E_{*,io} = \{(\mathsf{app}_{i,z}, *)\}$ to denote the *internal-to-external* communication relation from computer $i$ to any external computer outside of the network, and use arc set $E_{*,oi} = \{(*, \mathsf{app}_{j,z})\}$ to denote the *external-to-internal* communication relation from any computer outside of the network to computer $j$. Then, we define

$$E_* = E_{*,io} \cup E_{*,oi}. \tag{3.8}$$

We stress that $e \in E_{*,io} \cup E_{*,oi}$ always corresponds to a commuication or routing path going through a number of routers.

**Representation of networks**

Putting together the pieces mentioned above, we represent a network of $n$ computers as $G = (V, E)$, where

$$V = V_1 \cup \ldots \cup V_n \text{ and } E = E_1 \cup \ldots \cup E_n \cup E_0 \cup E_* \tag{3.9}$$

with $G_i = (V_i, E_i)$ being given by Eq. (3.6) and representing computer $i$, $E_0$ being given by Eq. (3.7) and representing the inter-computer communication relation between computers in the network, and $E_*$ being given by Eq. (3.8) and representing the internal-external communication relation.

For ease of reference, we use $V_{(app)}$ and $V_{(os)}$ to respectively denote the set of applications, and OSes running in the computers of an enterprise network, namely

$$\begin{aligned} V_{(app)} &= V_{1,app} \cup \ldots \cup V_{n,app}, && (3.10) \\ V_{(os)} &= V_{1,os} \cup \ldots \cup V_{n,os}. && (3.11) \end{aligned}$$

We also use $v \in V$ to indicate an arbitrary node $v$.

### 3.2.2 Representation of Vulnerabilities

In order to describe the vulnerabilities of an enterprise network, we consider two types of vulnerabilities: *software vulnerabilities* and *human vulnerabilities* following [106].

#### Representation of software vulnerabilities

Let $\mathsf{VUL}$ denote the set of software vulnerabilities in the software stacks of the computers in an enterprise network. We define a mathematical function

$$\phi : V \to 2^{\mathsf{VUL}} \tag{3.12}$$

such that $\phi(v)$ represents the set of software vulnerabilities that are contained in node $v$ (i.e., the software programs running at node $v$). Note that $|\phi(v) = 0|$ or $\phi(v) = \emptyset$ means that $v$ is not vulnerable.

Since different kinds of software vulnerabilities can incur different consequences, each vulnerability $\mathsf{vul} \in \mathsf{VUL}$ has an associated vector of attributes, including the access that is required in order to exploit a vulnerability, whether a vulnerability is zero-day or not, and whether or not the exploitation of a vulnerability can cause privilege escalation.

- *Access required*: This attribute describes what kind of access an attacker must have in order to exploit $\mathsf{vul} \in \mathsf{VUL}$, namely whether or not an attack requires to having local access in order to exploit $\mathsf{vul}$. In order to represent this attribute of vulnerabilities, we define predicate $\mathsf{loc}$ such that $\mathsf{loc}(\mathsf{vul}) = 0$ means exploitation of $\mathsf{vul}$ requires local access and $\mathsf{loc}(\mathsf{vul}) = 1$ otherwise.

- *Zero-day*: This attribute describes whether a vulnerability $\mathsf{vul} \in \mathsf{VUL}$ is zero-day or not. This is important because the exploitation of a zero-day vulnerability often cannot be detected. We define predicate $\mathsf{zd}$ such that $\mathsf{zd}(\mathsf{vul}) = 0$ means that vulnerability $\mathsf{vul}$ is known and $\mathsf{zd}(\mathsf{vul}) = 1$ means that $\mathsf{vul}$ is zero-day.

- *Privilege escalation*: This attribute describes the security consequence that can be caused by the exploitation of a vulnerability. We define predicate $\mathsf{priv}$ such that $\mathsf{priv}(\mathsf{vul}) = 0$ means that the exploitation of $\mathsf{vul}$ will not grant the attacker the `root` privilege, and $\mathsf{priv}(\mathsf{vul}) = 1$ means otherwise. This attribute is important because *remote-2-user* attacks exploit $\mathsf{vul}$'s with $\mathsf{loc}(\mathsf{vul}) = 0$ and $\mathsf{priv}(\mathsf{vul}) = 0$ [38, 52] and *remote-2-root* attacks exploit $\mathsf{vul}$'s with $\mathsf{loc}(\mathsf{vul}) = 0$ and $\mathsf{priv}(\mathsf{vul}) = 1$ [68, 111, 121], and *user-2-root* attacks exploit $\mathsf{vul}$'s with $\mathsf{loc}(\mathsf{vul}) = 1$ and $\mathsf{priv}(\mathsf{vul}) = 1$ [52, 111, 127].

**Representation of human vulnerabilities**

Users of the computers may be subject to social engineering attacks. In order to model human vulnerabilities to social engineering attacks, we define mathematical function

$$\psi : V \to [0, 1] \tag{3.13}$$

such that $\psi(v)$ for $v \in V_i$ represents the probability of the user of computer $i$ is vulnerable to social engineering attacks.

### 3.2.3 Representation of Attacks

We describe attacks via two aspects: the set of *exploits* that are available to an attacker, and the *strategies* that are employed by the attacker. In order to describe the attack consequence, we define mathematical function

$$\mathsf{state}(v,t) : V \times T \to \{0,1\}$$

such that $\mathsf{state}(v,t) = 0$ means $v$ is not compromised at time $t$ and $\mathsf{state}(v,t) = 1$ means $v$ is compromised at time $t$, where $T$ is the time horizon of interest (in principle, $T$ can be discrete or continuous).

#### Representation of exploits

Let $X$ denote the set of exploits that are possessed by the attacker. We define mathematical functions

$$\rho : X \times \mathsf{VUL} \to [0,1] \tag{3.14}$$

such that $\rho(x, \mathsf{vul})$ represents the success probability when applying exploit $x \in X$ against vulnerability $\mathsf{vul} \in \mathsf{VUL}$.

In order to describe the exploitation capability of the attacker, let us denote by $A$ the number of appearances of zero-day vulnerabilities in an enterprise network, by $B$ the number of appearances of known vulnerabilities against which the exploits can be detected and blocked by the defender where $B = k \times K$, and by $C$ the number of appearances of known vulnerabilities against which the exploits cannot be blocked by the defender where $C = (1-k) \times K$. We represent the attacker's exploitation capability by $(a, b, c)$, where $a$ is the percentage of the zero-day vulnerabilities that can be exploited by the attacker (i.e., the attacker can exploit $a \times A$ zero-day vulnerabilities), $b$ is the percentage of the known vulnerabilities that can be exploited by the attacker but will be detected and blocked by the defender (i.e., the exploitation of these $b \times B$ vulnerabilities will result in vein to the attacker), and $c$ is the percentage of the known vulnerabilities that can be exploited by the attacker and will not be detected by the defender (i.e., the exploitation of these $c \times C$ vulnerabilities will succeed). It is intuitive that the higher the $a$ and the $c$, the more attacks will succeed; the higher the $b$, the more attacks will be blocked.

### Representation of attack strategies

For representing attack strategies, we use the attack lifecycle model highlighted in Fig. 3.3, which is adapted from Lockheed Martin's Cyber Kill Chain [65] and Mandiant's Attack Life Cycle [95]. The model includes five phases that an attacker inescapably follows to plan and carry out an intrusion, that is, *reconnaissance*, *weaponization*, *initial compromise*, *further reconnaissance*, *escalate privileges*, and *lateral movement*. These phases are elaborated below.

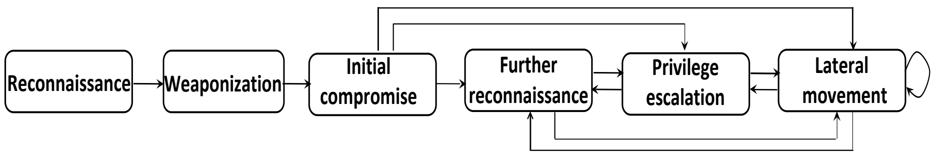


**Figure 3.3**: An attack lifecycle model adapted from Lockheed Martin's Cyber Kill Chain [65] and Mandiant's Attack Life Cycle [95].

**Phase 1: Reconnaissance.** Recall that an enterprise network is described by $G = (V, E)$, where $V$ is the node set and $E$ represents that legitimate commuication relations. Note that as stressed above, $e \in E$ reflects the logical communication relation and therefore often corresponds to a communication *path*, rather than a physical link (unless the two end-points reside in the same sub-network). Moreover, $G$ is not a complete graph because, for example, an application running on one computer is not authorized to communicate with an operating system function on another computer.

Reconnaissance means gathering information about a target enterprise network, including the communication topology $G$ and the vulnerabilities in the software stacks of the computers in the network. The outcome of an reconnaissance process can be described by the attacker's view of the target network, denoted by $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ (i.e., $G'$ is a sub-graph of $G$ induced by the reconnaissance process). For each $v \in V'$, the attacker may further obtain information such as $\eta(v)$, namely the type of the application running at node $v$; $\phi(v)$, namely the set of software vulnerability $v$ contains; and $\psi(v)$, namely the human factor vulnerability of $v$. Moreover, for each vulnerability $\mathsf{vul} \in \phi(v)$ where $v \in V'$, the attacker may further obtains information such as $\mathsf{loc}(\mathsf{vul})$, namely whether the vulnerability can be remotely exploited or not; $\mathsf{zd}(\mathsf{vul})$, whether the vulnerability is zero-day or not; and $\mathsf{priv}(v)$, whether the exploitation of the vulnerability can lead to a privilege escalation or not.

We propose using $\omega = |V'|/|V|$ to describe the attacker's *initial reconnaissance* capability. It is an interesting future study to investigate its alternate definisitons, such as $|E'|/|E|$ or $(|V'| + |E'|)/(|V| + |E|)$.

**Phase 2: Weaponization.** Weaponization is for designing and developing a penetration plan according to the information gathered from the reconnaissance phase and the exploits possessed by the attacker. In particular, given the outcome $G' = (V', E')$ of the reconnaissance phase and the attacker's set of exploits $X$, the attacker now determines the nodes $v \in V'$ suitable for targets. Since initial compromises are often geared towards applications and there are two kinds of nodes in general (i.e., client application vs. server application), a candidate node for initial compromise should satisfy one of the following two conditions: one for client applications and the other for server applications.

The first condition is for client applications. A candidate client node for initial compromise should satisfy the following conditions: (i) $v \in V_i$, where $V_i \subseteq V'$, runs a client application $\mathsf{app} \in \mathsf{APP}$ on computer $i$, namely $\eta(\mathsf{app}) = 0$; (ii) $v$ is involved in some internal-external communication relation, meaning $(v, *) \in E_{*,io} \cap E'$ or $(*, v) \in E_{*,oi} \cap E'$; (iii) either $\mathsf{app}$ contains a software vulnerability, namely $\exists \mathsf{vul} \in \phi(v) = \phi(\mathsf{app})$, the $\mathsf{app}$ contains no vulnerability but an operating system function called by the $\mathsf{app}$ contains a software vulnerable (i.e., there existing an access path from a secure $\mathsf{app}$ to a vulnerable operating system function).

In order to precisely test the preceding condition (iii), we say that there is a *dependence path* between two nodes $v$ and $u$ in the *same* computer, say computer $i$ or $V_i$, if there is, according to the *dependence relation* defined above, a path of dependence arcs starting from node $v$ and ending at node $u$ (i.e., the software program running at node $u$ can be called, or reached, by the software program running at node $v$). We define the predictate

$$\mathsf{dep\_path}(v, u) : V_i \times V_i \rightarrow \{\text{True}, \text{False}\} \tag{3.15}$$

such that

$$\begin{aligned} &\mathsf{dep\_path}(v, u) \\ = &\begin{cases} \text{True} & \text{there is a path of dependence arcs from } v \text{ to } u \\ \text{False} & \text{otherwise.} \end{cases} \end{aligned}$$

As a result, the preceding condition (iii) can be formally described as

$$\begin{gathered} (\exists \mathsf{vul} \in \phi(v), \exists x \in X : \psi(v) = 1 \wedge \rho(x, \mathsf{vul}) > 0) \vee \\ (\exists \mathsf{vul} \in \phi(u), \exists x \in X : (u \in V_{i,os}) \wedge (v \in V_{i,app}) \wedge \\ \mathsf{dep\_path}(v, u) \wedge \psi(u) = 1 \wedge \rho(x, \mathsf{vul}) > 0)). \end{gathered} \tag{3.16}$$

Putting the preceding discussion together, we can define the set of candidate client applications for initial compromise as:

$$\begin{aligned} \mathsf{Weapon}_0 = \{v \in (V' \cap V_{i,app}) : \eta(v) = 0 \wedge (((v, *) \in E_{*,io} \cap E') \vee \\ ((*, v) \in E_{*,oi} \cap E')) \wedge \text{condition (3.16) holds}\}. \end{aligned} \tag{3.17}$$

The second condition is for server applications. A candidate server node $v \in V_{i,app}$ for initial compromise should satisfy the following conditions: (i) $v$ runs an Internet-facing server application $\mathsf{app} \in \mathsf{APP}$ on computer $i$, meaning $\eta(\mathsf{app}) = 1$ and $(*, \mathsf{app}) \in (E_{*,oi} \cap E')$; (ii) either the $\mathsf{app}$ contains a remotely-exploitable software vulnerability, namely $\exists \mathsf{vul} \in \phi(\mathsf{app})$ such that $\mathsf{loc}(\mathsf{vul}) = 1$, or the $\mathsf{app}$ can call a operating system function that contains a remotely exploitable vulnerability.

Similar to the discussion above, the preceding condition (ii) can be precisely described as

$$\begin{aligned}(\exists \mathsf{vul} \in \phi(v), \exists x \in X : \mathsf{loc}(\mathsf{vul}) = 1 \wedge \rho(x, \mathsf{vul}) > 0) \vee \\ (\exists \mathsf{vul} \in \phi(u), \exists x \in X : (u \in V_{i,os}) \wedge (v \in V_{i,app}) \wedge \\ \mathsf{dep\_path}(v, u) \wedge \mathsf{loc}(\mathsf{vul}) = 1 \wedge \rho(x, \mathsf{vul}) > 0).\end{aligned} \tag{3.18}$$

Therefore, we can define the set of candidate server applications for initial compromise as:

$$\mathsf{Weapon}_1 = \{v \in V' \cap V_{i,app} : \eta(v) = 1 \wedge (*, v) \in (E_{*,oi} \cap E') \wedge \text{condition (3.18) holds}\}.$$

By putting the two kinds of initial compromises together, we obtain the set of candidates for initial compromise as:

$$\mathsf{Weapon} = \mathsf{Weapon}_0 \cup \mathsf{Weapon}_1. \tag{3.19}$$

**Phase 3: Initial compromise.** After determining Weapon according to (3.19), the attacker selects a subset of Weapon for initial compromise according to some strategy. An example strategy that will be considered for our simulation study is the following.

- The attacker prefers using exploits againat zero-day vulnerabilities to using exploits against known vulnerabilities. This is because exploits against zero-day vulnerabilities cannot be detected by the defense and using such exploits would enhance the attacker's chance in penetrating into the target enterprise network.

- The attacker strives to compromise the operating systems whenever possible. This is because the compromise of an operating system automatically causes the compromises ot the applications running on top of it. This can also reduce the chance that the attack is detected when compared with the strategy that the attacker first compromises the vulnerable applications and then compromises the vulnerable operating system beneath them (in this case, the attack against the applications may be detected by the defense because the defense may be employed in the kernel space).

- If the attacker cannot compromise the operating system on computer $i$, then the attacker will strive to compromise all of the vulnerable applications on computer $i$.

According to this example strategy, the attacker can identify the set of suitable nodes for initial compromise, denoted by

$$\mathsf{IniComp} = \{v \in \mathsf{Weapon} : \text{attacker selects } v \text{ to attack}\}.$$

**Phase 4: Further reconnaissance.** Further reconnaissance means that once the attacker compromises a computer in the enterprise network, the attacker will attempt to obtain information about the sub-graph $G - G'$, where $G = (V, E)$ represents the enterprise network as well as the application-induced dependence relation, inter-application communication relation, and internal-external communication relation, and $G' = (V', E')$ is the sub-graph obtained by the attacker at the initial reconnaissance phase. Since further reconnaissance can be conducted recursively, we also use $G' = (V', E')$ to denote the outcome after further reconnaissance, while noting that $G' = (V', E')$ increases with further reconnaissance. This is so because, supposing $\mathsf{app}_{i,1} \in V'$ (i.e., the attacker already knew obtained information about the node $v$ at which application $\mathsf{app}_{i,1}$ runs), arcs of the kind $(\mathsf{app}_{i,1}, \mathsf{app}_{j,2}) \in (E_0 \cap E')$ and the kind $(\mathsf{app}_{m,2}, \mathsf{app}_{i,1}) \in (E_0 \cap E')$ will be discovered by the attacker, and so are nodes $\mathsf{app}_{j,2}$ and $\mathsf{app}_{m,2}$, where $j, m \neq i$. Therefore, the attacker will update its information about the enterprise network as

$$V' = V' \cup \{\mathsf{app}_{j,2}, \mathsf{app}_{m,2}\}$$

and

$$E' = E' \cup \{(\mathsf{app}_{i,1}, \mathsf{app}_{j,2}), (\mathsf{app}_{m,2}, \mathsf{app}_{i,1})\}.$$

**Phase 4: Privilege escalation.** After compromising an application $v \in V_{i,app}$ but not the underlying operating system, the attacker would strive to achieve privilege escalation to compromise the underlying operating system based on the outcome of further reconnaissance. This can be achieved by compromising some vulnerable operating system functions. Another important factor affecting the success of privilege escalation is the host-based prevention system (HIPS), which may enforce a *tight* or *loose* policy.

- A *tight* policy says that the dependence (i.e., caller-callee) relation in a computer is strictly enforced. This means that the attacker cannot leverage a compromised application running at $v \in V_{i,app}$ to compromise the underlying operating system via a vulnerable function running at $u \in V_{i,os}$, *unless* $v$ was authorized to call $u$ already. More specifically, a privilege escalation occurs under the following condition:

$$\begin{aligned} &\exists v \in V_{i,app}, \exists u \in V_{i,os}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \\ &\mathsf{state}(v,t) = 1 \wedge \mathsf{dep\_path}(v,u) \wedge \rho(x,\mathsf{vul}) > 0. \end{aligned} \tag{3.20}$$

- A *loose* policy says that the dependence relation is not strictly enforced. This means that the attacker can leverage a compromised application running at $v \in V_{i,app}$ to compromise the underlying operating system via a vulnerable function running at $u \in V_{i,os}$, even if $v$ was not authorized to call $u$. More specifically, a privilege escalation occurs under the following condition:

$$\begin{aligned} &\exists v \in V_{i,app}, \exists u \in V_{i,os}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \\ &\qquad \mathsf{state}(v,t) = 1 \wedge \rho(x,\mathsf{vul}) > 0. \end{aligned} \tag{3.21}$$

**Phase 5: Lateral movement.** Lateral movements means once the attacker compromises a computer in the enterprise network, the attacker will leverage the inter-computer communication relation $e \in E'$ to launch further attacks for compromising more computers. This can occur in one of the following two scenarios, depending on the new victim is a client (including a peer in peer-to-peer applications) or server application.

In the first scenario, the attacker has compromised a client or server or peer application on computer $i$ and attempts to use the inter-computer communication relation $e \in (E' \cap E_{00})$ to compromise a client application on computer $j$. This occurs under one of the following two conditions:

$$\begin{aligned} &(\exists u \in V_{j,app}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \mathsf{state}(v,t) = 1 \wedge \\ &\mathsf{state}(u,t) = 0 \wedge (v,u) \in (E' \cap E_{00}) \wedge \\ &\psi(u) = 1 \wedge \rho(x, \mathsf{vul}) > 0); \end{aligned} \tag{3.22}$$

$$\begin{aligned} &\vee(\exists u \in V_{j,app}, \exists w \in V_{j,os}, \exists \mathsf{vul} \in \phi(w), \exists x \in X : \\ &\mathsf{state}(v,t) = 1 \wedge \mathsf{state}(u,t) = 0 \wedge \\ &(v,u) \in (E' \cap E_{00}) \wedge \psi(u) = 1 \wedge \\ &\mathsf{dep\_path}(u,w) \wedge \rho(x, \mathsf{vul}) > 0). \end{aligned} \tag{3.23}$$

In the second scenario, the attacker has compromised a client or server application on computer $i$ and attempts to use the inter-computer communication relation $e \in (E' \cap E_{01})$ to compromise a server application on computer $j$. This occurs under one of the following two conditions:

$$\begin{aligned} &(\exists u \in V_{j,app}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \mathsf{state}(v,t) = 1 \wedge \\ &\mathsf{state}(u,t) = 0 \wedge (v,u) \in E' \cap E_{01} \wedge \\ &\rho(x, \mathsf{vul}) > 0 \wedge \mathsf{loc}(\mathsf{vul}) = 1); \end{aligned} \tag{3.24}$$

$$\begin{aligned} &\vee(\exists u \in V_{j,app}, \exists w \in V_{j,os}, \exists \mathsf{vul} \in \phi(w), \exists x \in X : \\ &\mathsf{state}(v,t) = 1 \wedge (u,t) = 0 \wedge (v,u) \in E' \cap E_{01} \wedge \\ &\mathsf{dep\_path}(u,w) \wedge \rho(x, \mathsf{vul}) > 0 \wedge \mathsf{loc}(\mathsf{vul}) = 1). \end{aligned} \tag{3.25}$$

Note that Eqs. (3.22) and (3.24) say that an application $\mathsf{app}$ running on computer $j$ can be exploited from a compromised application running on computer $i$ when $\mathsf{app}$ contains a software vulnerability; Eqs. (3.23) and (3.25) say that an application $\mathsf{app}$ running on computer $j$ can be exploited from a compromised application running on computer $i$ when $\mathsf{app}$ calls a vulnerable operating system function.

### 3.2.4 Representation of Defenses

#### Representation of Firewalls

A firewall can be employed to monitor the inbound and outbound traffic of an enterprise network or monitor the traffic between sub-networks of the enterprise network. A firewall has multiple physical *interfaces*, each corresponding to an internal sub-network or the external network [64]. If needed, multiple interfaces (i.e., the corresponding multiple sub-networks) can be grouped into a *security zone* such that the traffic within the same security zone is not monitored but the inbound and outbound traffic of a security zone is monitored [94]; otherwise, each interface corresponds to a security zone. The monitored traffic will be examined according to some *security rules*, which specify what kinds of traffic should be permitted or denied between different security zones. Security rules can be specified at the packet level and/or the flow level, while noting that a flow consists of one or multiple related packets between the same communication end-points using the same protocol within a short period of time. Packet-level rules examine each IP packet to determine whether to permit or deny the packet; whereas, flow-level rules (aka "stateful" firewalls) examine flows to determine whether to permit or deny the flow. It is widely accepted that firewalls will deny any accesses that are not explicitly specified.

For the purpose of the present study, it is sufficient to model the functionality of firewalls as monitoring and examining the traffic over the aforementioned *inter-application* communication relation, *inter-computer* communication relation within an enterprise network, and *internal-external* communication relation. It is worth mentioning that the monitoring and examination of these kinds of traffic can be enfored at the packet- and/or flow-level, while noting that these implementation details can be safely assumed away.

#### Representation of DMZ

DMZ is a special *security zone* and typically hosts publicly accessible (i.e., Internet-facing) server applications, such as Web servers, Mail servers and DNS servers [139]. DMZ provides a kind of isolation between an enterprise network and the outside network in the sense that the computers in the external network have, in general, no legitimate reasons to *directly* communicate with the computers residing within an enterprise network (i.e., such external-to-internal communicatio traffic could be filtered). A DMZ can be attained by using a firewall that has three kinds of interfaces: one or multiple internal interfaces (connecting to the internal network), an external interface (connecting to the external network), and a DMZ interface (connecting to a DMZ) [122]. Again, the concrete implementation of DMZ is not important to the purpose of the present study.

**Representation of Other Defense Mechanisms**

In addition to firewalls and DMZ, we further consider the following defenses, while leaving the incorporation of other defenses to future research. First, we consider the defense of NIPS, which may operate at the network, flow, or application layer as mentioned in the Introduction. NIPS may be able to detect and prevent some attacks that attempt to exploit some *known* software vulnerabilities. Denote by $K$ the number of *apperances* of known vulnerabilities in an enterprise network (e.g., 3 appearances of the same vulnerability in different computers leading to $K = 3$). In the ideal case, the defender should have patched all known vulnerabilities, meaning $K = 0$; in practice, some vulnerabilities, even if known, may not be patched. Suppose NIPS can detect and block attacks that attempt to exploit a $k$ fraction of the known vulnerabilities (i.e., $k \times K$ vulnerabilities can not be exploited), where $0 \leq k \leq 1$.

Second, we consider the defense in dealing with human vulnerabilities to social engineering attacks. Recall that the probability that a user is vulnerable to social engineering attacks is defined by $\psi : V \to [0, 1]$. The defense against social engineering attacks can be measured as the reduction of $\psi$ to $\psi(1-\alpha)$, where $\alpha$ denote the probability a social engineering attack is detected and blocked by the defender.

Third, we consider host-based intrusion prevention systems (HIPSes). HIPS an enforce a *tight* or *loose* policy, where *tight* means that (a secure) HIPS monitors which software (e.g., application) is allowed to call which other software in a computer (e.g., an OS function) and blocks any unauthorized calls or communications, and *loose* means that the HIPS does not operate as such (e.g., the HIPS does not prevent a compromised application from making an unauthorized call to an OS function). Moreover, HIPS may be able to detect and block privilege escalation attempts. Let $\zeta$ denote the probability a privilege escalation attempt is detected and blocked by HIPS. Since enforcing a tight policy would consume more resources than enforcing a loose policy, we will quantify how much security can be gained by enforcing a tight policy.

### 3.2.5 Metrics for Measuring the Cybersecurity State of an Enterprise Network

In the above we have defined various metrics to measure vulnerabilities, defenses and attacks. Now we propose using the following two metrics to describe the outcome of the attack-defense interactions: *percentage of compromised applications* at time $t$ or $\mathsf{pca}(t)$, and *percentage of compromised operating systems* at time $t$ or $\mathsf{pcos}(t)$, which are respectively defined as:

$$\mathsf{pca}(t) = |\{v \in V_{(app)} : \mathsf{state}(v,t) = 1\}|/|V_{(app)}|, \tag{3.26}$$

$$\mathsf{pcos}(t) = |\{v \in V_{(os)} : \mathsf{state}(v,t) = 1\}|/|V_{(os)}|. \tag{3.27}$$

## 3.3 Simulation Experiments and Results

### 3.3.1 Simulation Setting and Methodology

#### Simulating an Enterprise Network

We consider an enterprise network of 1,000 desktops and five servers. For the experimental study, we need a concrete $G = (V, E)$ representation of an enterprise network, which is obtained as follows.

**Simulating software stacks.** Suppose the five servers respectively run one of the following server applications: web server, email server, DNS server, FTP server, and database server. Suppose each desktop runs 4 applications: web browser, email client, instant messaging (IM, a peer-to-peer application), adobe reader. Suppose each desktop may run a FTP client application with probability

$p_1$ and a database client application with probability $p_2$, where $0 \leq p_1, p_2 \leq 1$. This means that employees can use these applications.

For the OS layer, we assume the OS is Microsoft Windows with three components: OS kernel, subsystem drivers, and the hardware abstraction layer. We use this example scenario as we can obtain realistic parameters, namely that there are 2,500, 1,300, and 130 functions respectively corresponding to these components [21,93]. This means $|V_{i,os}| = 3,930$.

**Simulating network computers.** Recall that computer $i$ is represented by $G_i = (V_i, E_i)$ for $i = 1, \ldots, 1,005$. In order to obtain $E_i$, we assume there are no *inter-application communications* between the applications mentioned above. This is natural in the present setting but may not be true in general. In order to obtain the *dependence* relation within computer $i$, we assume for simplicity that each OS function is called, directly or indirectly, by each application with probability $\delta$.

**Simulating inter-computer communication relation $E_0$.** In order to obtain $E_0$, we make the following assumptions: a web browser in the enterprise network needs to communicate with the web server and the DNS server in the network; an email client needs to communicate with the email server to retrieve and send emails; a FTP client (if present) needs to communicate with the FTP server; a database client (if present) needs to communicate with the database server; an IM application needs to communicate with the other IM applications within the same sub-network; the web server needs to communicate with the database server. Since the email clients need to communicate with each other, this communication relation is reflected in $E_0$.

**Simulating internal-external communication relation $E_*$.** In order to obtain $E_*$, we make the following assumptions: a web browser needs to communicate with the web servers that reside outside of the network; IM applications need to communicate with their peers outside of the enterprise network; email clients in the network need to send emails to, and receive emails from, the external network; Adobe readers need to open PDF files received from the external network; the Internet-facing servers (i.e., web server, email server, and DNS server) can be accessed by external computers.

### Simulating Vulnerabilities

Suppose each application or OS function has at most one vulnerability. Let $N$ denote the number of vulnerabilities that uniformly reside in $N$ OS functions. Let $\beta$ be the probability that every application contains a vulnerability. The attributes of a vulnerability $\mathsf{vul} \in \mathsf{VUL}$ is set as follows: If $\mathsf{vul}$ belongs to an OS function, we set $\mathsf{priv}(\mathsf{vul})$=1; otherwise, $\mathsf{priv}(\mathsf{vul})$=0. Let $\vartheta(\mathsf{vul})$ be the probability $\mathsf{vul}$ can be exploited remotely for any $\mathsf{vul} \in \mathsf{VUL}$, namely $\Pr(\mathsf{loc}(\mathsf{vul}) = 1)$, and $\tau(\mathsf{vul})$ be the probability that $\mathsf{vul} \in \mathsf{VUL}$ is zero-day, namely $\Pr(\mathsf{zd}(\mathsf{vul}) = 1)$.

For human vulnerabilities, we let $\psi(v) \in [0, 1]$ be the probability that a client application is vulnerable to social engineering attacks for every $v \in V_i$. Note that this holds for every $v \in V_i$ because $V_i$ corresponds to computer $i$, meaning it is the user of computer $i$ who may be subject to social engineering attacks.

### Simulating Attacks

We consider an attacker outside of the network attempting to penetrate into the network and compromise as many computers as possible. Attacks proceed according to the strategy highlighted in Figure 3.3. The parameters reflecting attacks, namely $\rho$, $\omega$ and $(a, b, c)$, will be given in specific simulation scenarios.

### Simulating Defenses

Since we focus on quantifying the security effectiveness of firewalls and DMZ, our simulation considers the five example combinations of firewalls and DMZ illustrated in Fig.3.4, which are respectively represented by $\gamma = 0, 1, \ldots, 4$. The other parameters representing defenses are as described in the framework and will be given in specific simulation scenarios.

Fig.3.4(a) corresponds to $\gamma = 0$, meaning neither firewall nor DMZ is employed. As a consequence, a communication not belonging to $E_0$ or $E_*$ will not be blocked.

Fig.3.4(b) corresponds to $\gamma = 1$, meaning there is only a perimeter firewall to separate the enterprise network from the external network. That is, the firewall blocks any internal-external

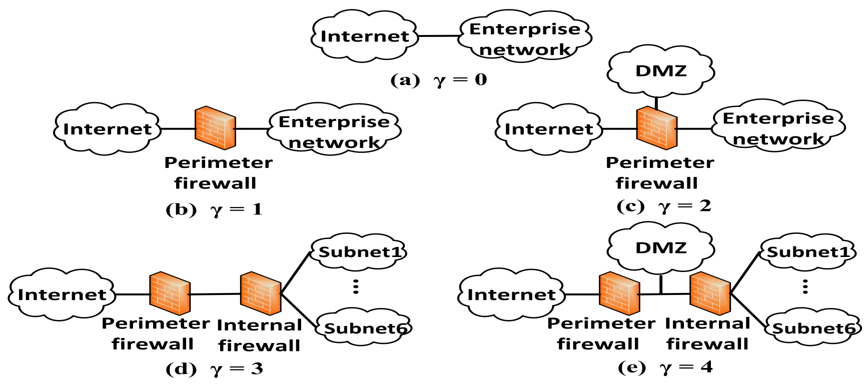


**Figure 3.4**: Five combinations of firewalls and DMZ employment (identified by $\gamma = 0, 1, 2, 3, 4$ and elaborated in the text).

communication that does not belong to $E_*$.

Fig.3.4(c) corresponds to $\gamma = 2$, meaning that there is a perimeter firewall and a DMZ for running the Internet-facing servers (i.e., web server, email server, and DNS server). This firewall enforces as in the case of $\gamma = 1$ and further enforces that only the web server (both not the other two servers) in the DMZ can communicate with the database server in the internal enterprise network.

Fig.3.4(d) corresponds to $\gamma = 3$, meaning that there is a perimeter firewall and there are internal firewalls to separate the internal sub-networks from each other. The 1,005 computers are divided into six sub-networks, among which five sub-networks run 200 desktops each and the other sub-network runs the five servers. The perimeter firewall allows internal-external communications according to $E_*$ and the internal firewalls allow inter-computer communications according to $E_0$.

Fig.3.4(e) corresponds to $\gamma = 4$, meaning that there is the same as in the case of $\gamma = 3$ except that the DMZ runs the Internet-facing servers (i.e., web server, email server, and DNS server). As in the case of $\gamma = 2$, the perimeter firewall enforces that only the web server (both not the other two servers) in the DMZ can communicate with the database server in the internal enterprise network.

## Simulation Algorithm

The simulation algorithm executes the attach strategy discussed above. Each simulation run leads to a sequence $\mathsf{state}(v, t)$ for $v \in V$ and $t = 1, \ldots, T$. We conduct 200 independent simulation runs with the given parameters specified below. From these sequences we derive the average values of metrics $\mathsf{pca}(t)$, $\mathsf{pcsa}(t)$ and $\mathsf{pcos}(t)$ per data point.

### 3.3.2 Simulation Results and Analysis

In the present simulation experiments, we assume that OSes are not vulnerable, but the HIPS and NIPS are not effective. This is because we focus on measuring the effectiveness of firewalls and DMZs. Because the OSes are not vulnerable, we only consider metrics $\mathsf{pca}$ and $\mathsf{pcsa}$.

The parameters are set as follows: $p_1 = 0.1$ (the probability a desktop runs a FTP client application), $p_2 = 0.1$ (the probability a desktop runs a database client application), $\delta = 0.1$ (the probability that an OS function is called by an application), $N = 0$ (OSes are not vulnerable), $\psi(v) = 0.5$ (the probability $v \in V$ is vulnerable to social engineering attacks), $\vartheta(\mathsf{vul}) = 0.5$ (the probability $\mathsf{vul}$ can be exploited remotely), $\tau(\mathsf{vul}) = 0.5$ (the probability $\mathsf{vul}$ is zero-day), $k = 0$ (the probability an attack attempting to exploit a known-but-unpatched vulnerability is blocked by the NIPS), $\alpha = 0$ (the probability a social engineering attack is blocked by HIPS), $\zeta = 0$ (the probability a privilege escalation is blocked by HIPS), $(a, b, c) = (1, 1, 1)$, $\rho(x, \mathsf{vul}) = 1$ (the probability that $x \in X$ successfully exploits vulnerability $\mathsf{vul} \in \mathsf{VUL}$), $\omega = 1$ (the fraction of nodes that are discovered by the attacker's initial reconnaissance), and HIPS enforces the loose policy.

Note that the parameter setting implies the worst scenario in which the attacker can obtain all of the exploits against the vulnerabilities in the network and can discover all of the nodes via an initial reconnaissance. We consider $k = 0$ and $\alpha = 0$ (i.e., both NIPS and HIPS cannot block any attacks) because in the present work we focus on measuring the security effectiveness of firewalls and DMZs.

**Determining simulation time horizon $T$.** Fig. 3.5 plots $\mathsf{pca}(t)$ and $\mathsf{pcsa}(t)$ of different combinations of firewalls and DMZ (reflected by $\gamma = 0, 1, 2, 3, 4$), where $\beta = 0.5$ (i.e., the probability that an application is vulnerable). We observe that both $\mathsf{pca}(t)$ and $\mathsf{pcsa}(t)$ first increase exponentially and then converge to a steady value. The exponential increase is caused by communications between email clients and/or between IM clients, namely that the compromise of any of these clients can cause the compromise of the other vulnerable clients. A similar phenomenon is observed for

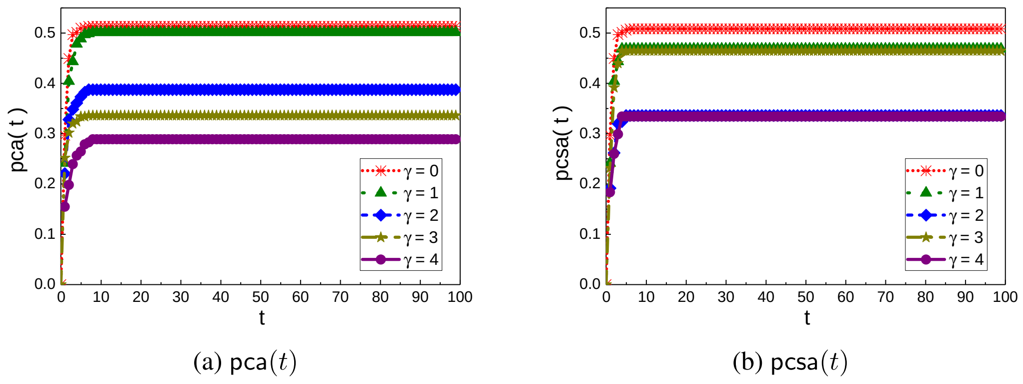


**Figure 3.5**: pca$(t)$ and pcsa$(t)$ of different combinations of firewalls and DMZ.

other values of $\beta$. Therefore, we will set $T = 100$ as the simulation time horizon for the simulation experiments reported below.

**Security effectiveness of firewalls and DMZ.** Fig.3.6 plots pca$(T)$ and pcsa$(T)$ for $T = 100$ with respect to $\beta$ under different combinations of firewalls and DMZ ($\gamma = 0, 1, 2, 3, 4$). Fig.3.7 plots $\Delta$pca$(T, \gamma)$ and $\Delta$pcsa$(T, \gamma)$ for $T = 100$ with respect to $\beta$, where $\Delta\text{pca}(T, \gamma) = \text{pca}_{T,\gamma=0} - \text{pca}_{T,\gamma}$, $\Delta\text{pcsa}(T, \gamma) = \text{pcsa}_{T,\gamma=0} - \text{pcsa}_{\gamma}$, and $\text{pca}_{T,\gamma}$ and $\text{pcsa}_{T,\gamma}$ respectively represent pca$(T)$ and pcsa$(T)$ with respect to a specific $\gamma$ value. Note that $\Delta\text{pca}(T, 0) = 0$ because $\gamma = 0$, namely that neither firewall nor DMZ is employed, which is the baseline case. The key findings are discussed below.

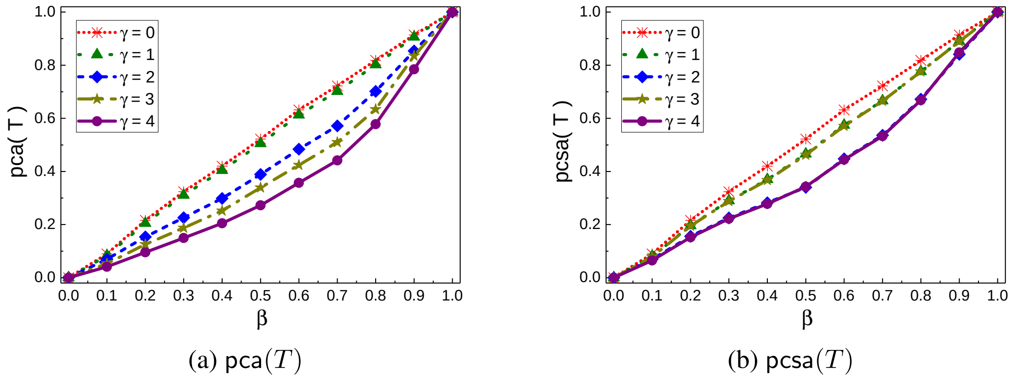


**Figure 3.6**: pca$(T)$ and pcsa$(T)$ with $T = 100$ and different $\gamma$'s (i.e., combinations of firewalls and DMZ).

**Finding 1:** Fig. 3.6a shows that $1 - \text{pca}(100)$ with respect to a fixed firewall and DMZ configuration (i.e., fixed $\gamma$) decreases as $\beta$ increases. This means that for a fixed defense strategy, the attack-defense interaction outcome is dominated by the degree of vulnerability of an enterprise network, namely the fraction of applications that contain at least one vulnerability. A similar phenomenon is also observed in Fig.3.6b. This leads to:

**Insight 1.** *When OSes are not vulnerable, the security effectiveness of a fixed combination of firewalls and DMZ decreases as the fraction of vulnerable applications increases.*

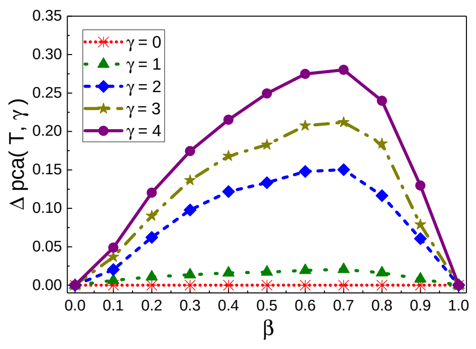
**Finding 2:** Fig.3.7 shows that $\Delta\mathsf{pca}(T,\gamma)$ and $\Delta\mathsf{pcsa}(T,\gamma)$ for a fixed $\gamma$ are large when $\beta$ is neither too small nor too large, but approaches zero when $\beta$ tends to 0 or 1. That is, when few or most applications are vulnerable, firewalls and DMZ are not effective because in the former case, few applications can be compromised and in the latter case, most applications are eventually compromised.

**Insight 2.** *Firewalls and DMZ are not effective when few or most computers are vulnerable.*

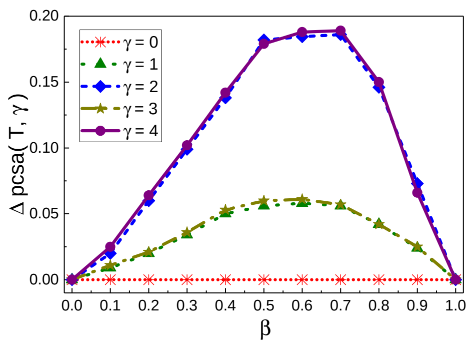
**Finding 3:** Fig.3.7a shows that $\Delta\mathsf{pca}(T,1) \approx 0$ or 1.08% when averaged over $\beta$, meaning that the perimeter firewall is not effective. This is because the attacker can penetrate into the network via means that cannot be blocked by the perimeter firewall. When averaged over $\beta$, the difference between $\Delta\mathsf{pca}(T,1)$ and $\Delta\mathsf{pca}(T,2)$ is 8.71%, and the difference between $\Delta\mathsf{pca}(T,1)$ and $\Delta\mathsf{pca}(T,3)$ is 12.98%. Finally, we see that $\gamma = 4$ (i.e., enforcing comprehensive firewalls and DMZ defense) leads to the highest security among them, which justifies the real-world defense practice.

**Insight 3.** *Employing the perimeter firewall lone has a little security impact, but a comprehensive use of firewalls and DMZ can substantially increases security.*

**Finding 4:** Fig.3.7b shows that the security effectiveness firewalls and DMZ in terms of $\mathsf{pcsa}$. We observe that $\Delta\mathsf{pcsa}(T,1) \approx \Delta\mathsf{pcsa}(T,3)$, meaning that employing perimeter firewall alone and employing both perimeter firewall and internal firewalls lead to the same security. This indicates that enforcing internal firewalls will not protect the server applications. However, $\Delta\mathsf{pcsa}(T,2) = 11.47\%$ when averaged over $\beta$, which highlights the security effectiveness of DMZ



(a) $\Delta\mathsf{pca}(T,\gamma)$



(b) $\Delta\mathsf{pcsa}(T,\gamma)$

**Figure 3.7**: $\Delta\mathsf{pca}(T,\gamma)$ and $\Delta\mathsf{pcsa}(T,\gamma)$ with $T = 100$ and different $\gamma$'s.

in protecting server applications. Nevertheless, $\Delta\mathsf{pcsa}(T,2) \approx \Delta\mathsf{pcsa}(T,4)$ affirms that security of server applications is protected by DMZ.

**Insight 4.** *Employing perimeter firewall and DMZ can substantially increase the security of sever applications.*

## 3.4 Chapter Summary

In this Chapter, we instantiated the proposed framework for quantifying the security effectiveness of firewalls and DMZs. The framework led to novel and useful insights. For example, when the applications and OSes have few or too many vulnerabilities, firewalls and DMZ do not have a significant impact on security; when the OSes are not vulnerable but the applications are, security effectiveness of firewalls and DMZs decreases as the fraction of vulnerable applications increases.

# CHAPTER 4: QUANTIFYING SECURITY EFFECTIVENESS OF COARSE-GRAINED DYNAMIC NETWORK DIVERSITY

## 4.1 Chapter Introduction

Software monoculture enables the automatic amplification of cyber attack damages because vulnerabilities are replicated network-wide or even cyberspace-wide [51, 125]. As a consequence, a single exploit may allow an attacker to compromise many programs. In order to cope with the problem, researchers have proposed diversifying program implementations [13,72,166], leading to the notion of *software diversity*. There are various flavors of software diversity, such as: $N$-version programming (i.e., a program specification has multiple independent implementations [10, 26]); natural diversity (e.g., multiple browsers incurred by market competition [58]); compiler-based diversification (i.e. random executables generated from a given source code [35, 46, 61, 66, 73]); and software runtime environment diversification (e.g., address space layout randomization [14, 41,45,140], instruction set randomization [11,70], system calls randomization [29], and replicated execution [118]).

It is intuitive that employing software diversity could increase security. For example, the U.S. Navy developed the RHIMES system [5] to enhance security of shipboard systems, by introducing diversity to each programmable logic controller. However, real-world software diversity is often employed in an ad hoc fashion, which can be justified by how different OSes (e.g., Windows vs. various kinds of Unix) and browsers (e.g., Safari vs. Firefox vs. Chrome) are employed in practice. One exception is the investigation of employing software diversity to enhance Byzantine Fault-Tolerance (BFT), namely how to employ software diversity in the replica implementations so that they do not contain common vulnerabilities [48, 99, 110, 115, 116, 138]. This is important because the theoretical fault-tolerance guarantee can be ruined otherwise. Another exception is the investigation of employing software diversity in detecting cyber attacks by leveraging the behavioral discrepancy between diversified replicas [69, 130, 144]. Despite these studies, some fundamental questions remain open, such as: *How should software diversity be employed in practice to amplify, if not maximize, security?*

The preceding question leads to the notion of *network diversity*, which deals with the employment of diversified program implementations in network-wide software stacks [50, 51, 104, 125, 166]. A simpler version of the notion may be called *static* diversity, where diversified implementations are employed once and for all (i.e., unchanged after initial employment). There are studies on optimizing static diversity via some flavor of *graph coloring* algorithm; by treating colors as diversified implementations, the research problem is to minimize defective edges (i.e., adjacent nodes have the same color or run the same implementation) [15, 49, 50, 62, 104, 160]. There are also studies on quantifying the effectiveness of *static* diversity [15, 100, 128, 133, 165].

Despite these studies, there is no systematic understanding on the network-wide effectiveness of employing software diversity, for multiple reasons. First, most studies use coarse-grained models (i.e., treating each computer as a unit) and do not consider attack-defense interactions. Second, it is not clear how to quantify the network-wide cybersecurity effectiveness of employing network diversity. Third, it is not clear how network diversity should be dynamically employed, leading to the notion of *dynamic* diversity. Addressing these problems can deepen our understanding of software diversity, help decision-makers determine whether or not to invest in software diversity, and guide practitioners in intelligently employing diversified implementations in real-world cyber defense operations.

### 4.1.1 Chapter Contributions

This Chapter makes three contributions. First, we propose a framework for modeling and quantifying the network-wide cybersecurity of enforcing network diversity. The framework distinguishes applications and operating systems, which allows us to accommodate privilege escalation. The framework considers the time dimension, which allows us to investigate *dynamic* diversity; that is, the employment of diversified implementations in the network evolves over time.

Second, we propose a suite of metrics to quantify the network-wide effectiveness of employing diversity, including: (i) *time-to-succeed*, which measures how long it takes an attacker to break a defender's goal (if possible); (ii) *attacker slow-down*, which measures the extent an attacker is slowed down by network diversity; (iii) *attack worst damage*, which measures the damage an attacker can cause in the worst case; (iv) *attack extra cost*, which measures the extra investment the defense imposes on an attacker in order to break the defender's goal; (v) *vulnerability tolerance*, which measures the upper bound of vulnerabilities that can be tolerated when achieving the defender's goal; (vi) *average operational cost*, which measures the average fraction of programs that re-deploy dynamic diversity. These metrics may be of independent value.

Third, we demonstrate the usefulness of the framework and metrics by presenting a multi-agent simulation study with multiple network diversity strategies: monoculture (the baseline with no diversity), static diversity (for comparison purposes), proactive (periodically re-diversifying network-wide software stacks), reactive-adaptive (re-diversifying the network stacks in response to detected attacks), and hybrid (of the last two). According to the simulation study, we draw a number of insights, including: (i) In terms of attacker slow-down, reactive-adaptive diversity is the most effective strategy and the initial diversity configuration matters. (ii) In order to reduce the attack worst damage, different diversity strategies should be used in different parameter regimes. (iii) Reactive-adaptive diversity leads to a higher vulnerability-tolerance than proactive diversity does. (iv) Proactive diversity improves security only when dynamic diversity is widely re-employed at a high frequency, which however incurs a high operational cost. (v) The more the diversified implementations, the higher the attacker slow-down, the higher the attack extra cost, and the higher the vulnerability tolerance. This is especially true for reactive-adaptive diversity. Note that these findings may not be universally true because they are derived from the parameter settings used in the simulation study.

### 4.1.2 Related Work

**Prior studies in software diversity**. Prior studies mainly aim at obtaining diversified, ideally *independent*, implementations of a program specification [13]. There is a body of literature on software diversity (e.g., [11, 14, 26, 35, 45, 46, 61, 70, 73, 140]). However, the effectiveness of these building-block techniques is not well understood. For example, one study shows that independent software implementation does not lead to independent vulnerabilities because programmers tend to make the same mistakes [71]. Other studies show positive results, such as: the same vulnerabilities in different software may demand different exploits [54]; few vulnerabilities simultaneously appear in different OSes [57]. In contrast, the effectiveness of run-time diversity is better understood: (i) address space layout with 32-bit randomization can be compromised by brute-forcing [120]; (ii) address space layout with higher entropy can be compromised by side-channel attacks [37, 42, 117, 126]; (iii) address space layout randomization is vulnerable because of modern cache architectures [53]; (iv) fine-grained address space layout randomization is subject to just-in-time code reuse attacks [123]; and (v) instruction set randomization is subject to brute-forcing attacks [124, 137].

The preceding studies consider standalone software diversification techniques. In contrast, we investigate the effectiveness of software diversification techniques from a network standpoint. Since we consider diversifying network-wide software stacks, multiple diversification techniques can be used together. This is reminiscent of the notion of $N$-variant systems [34], which however do not quantify the network-wide effectiveness dynamic diversity.

**Prior studies in network diversity**. Network diversity has been investigated in some contexts [50, 51, 104, 125, 166]. There are proposals on measuring static network diversity via (i) the entropy of the distribution of software vulnerability in a network [100] and (ii) the diversity index of the shared vulnerabilities between different software implementations in a network [128]. There are attempts at optimizing static diversity by using some flavor of *graph coloring* algorithm (i.e., treating a diversified implementation as a color and a network as a graph) and minimizing defective edges (i.e., adjacent nodes have the same color or run the same software implementation) [15, 49, 50, 62, 104, 160].

The closely related prior studies are [133, 165], which measure network diversity via *resource richness* and *attack effort*. Our study is different as follows: (i) they investigate how to quantify diversity, whereas we study how to employ *dynamic* diversity; (ii) they do not consider attack-defense interactions, whereas we explicitly model attack-defense interactions; (iii) we quantify the network-wide effectiveness of dynamic diversity, which is not studied by them.

**Prior studies related to Moving-Target Defense (MTD)**. Proactive diversity is one form of MTD, which includes other kinds of proactive defense techniques (e.g., proactively changing IP addresses or port numbers) [31]. Quantifying the security effectiveness of MTD in the broader context is beyond the scope of the present paper (see, e.g., [56]). While our finding that reactive-adaptive is more effective than proactive diversity (i.e., MTD when applied to diversity) may sound counter-intuitive at a first glance, it can be understood as follows: MTD (i.e., proactive diversity in this case) can be employed when the defender does not know the situation information of the network (e.g., which and how many nodes are compromised); in contrast, adaptive diversity can leverage the situational information to adaptively employ diversity, leading to potentially higher effectiveness. In addition, our simulation study focuses on dynamic diversity against malware-like attacks after the attacker establishes footholds at some compromised computers in a network. This means that the simulation study does not consider earlier stages of cyber attacks, such as reconnaissance. Our findings do not contradict the usefulness of MTD in defending against such earlier-stage attack activities (e.g., MTD can effectively disrupt attacker's reconnaissance processes [19, 63, 67, 90]).

**Prior studies related to whole-network security analysis**. We analyze the security effectiveness of dynamic diversity from a whole-network perspective. There are studies in this perspective, but tackling different problems and using different approaches. The *attack graph* approach studies how an attacker may exploit multiple vulnerabilities to achieve a certain goal and how to harden a network (see, e.g., [7, 8, 28, 60, 109, 114, 121, 134, 164]). This approach is *combinatorial* in nature and does not consider the *time* dimension [148, 149, 152]. Another approach is the *cybersecurity dynamics* framework [148, 149, 152], which explicitly models attack-defense interactions over time and includes a rich family of models and results (e.g., [17, 56, 79, 86, 89, 146, 150, 151, 156, 159, 167, 168]). These studies aim at analytical results while making simplifying assumptions, such as the *independence* assumption between attacks. In order to eliminate such assumptions, initial efforts have been made in both theoretical studies [36, 146, 147] and empirical studies [23, 24].

Our framework does not make the independence assumption, while characterizing the *transient* behaviors (i.e., the dynamics before converging to an equilibrium); whereas, analytical models so far can only offer asymptotic results (i.e., $t \to \infty$ or when the dynamics converges to the equilibrium). Last but not the least, our framework goes much beyond [24] by characterizing (e.g.) dynamic attack-defense interactions and decision-making.

### 4.1.3 Chapter Organization

The rest of this Chapter is organized as follows. Section 4.2 instantiates the proposed framework and proposes a set of security metrics. Section 4.3 describes our simulation experiments and insights drawn from our experimental results. Section 4.4 concludes the work.

## 4.2 Instantiating the Framework to Quantify Security Effectiveness of Coarse-Grained Dynamic Network Diversity

Consider a network, where each node represents a computer. Each computer has a software stack, running an operating system (OS) in the kernel space and some application(s) in the user space. Each *program*, OS and application alike, may have diversified implementations (e.g., Safari vs. Firefox vs. Chrome for browser). These diversified implementations may be obtained by using some of the diversification methods mentioned above (e.g., $N$-version programming or natural diversity via market competition). Each program may or may not be vulnerable.

**Intuition of Dynamic Network Diversity**. Given diversified implementations of programs, the aforementioned notion of *static* network diversity is to employ diversified programs in the computers' software stacks, where the employment (or configuration) will not change during the time horizon of interest. We initiate the aforementioned *dynamic* network diversity, which aims to dynamically employ diversified programs at the computers' software stacks, where the employment (or configuration) does change during the time horizon of interest.

Figure 4.1 illustrates the idea via a network of 5 computers, denoted by $1 \leq i \leq 5$. The time horizon is $t = 0, \ldots, 5$. Each computer runs an OS, and we use $\mathsf{os}_i$ to represent the OS running in computer $i$. There are two applications, denoted by $\mathsf{APP}_j$ for $1 \le j \le 2$ (e.g., $\mathsf{APP}_1$ is browser and $\mathsf{APP}_2$ is email). Computers 1, 4 and 5 run both applications; computer 2 runs $\mathsf{APP}_2$; computer 3 runs $\mathsf{APP}_1$. We use $\mathsf{app}_{i,j}$ to represent the application $\mathsf{APP}_j$ running in computer $i$. **(i)** There are three implementations of OS (e.g., Windows vs. Linux vs. macOS), which are respectively indicated by three colors. For example, at time $t = 0, 1, 2$, computers 1 and 4 run the same OS, causing $\mathsf{os}_1$ and $\mathsf{os}_4$ to have the same color; computers 2 and 5 run another OS, causing $\mathsf{os}_2$ and $\mathsf{os}_5$ to have another color; computer 3 runs yet another OS, causing $\mathsf{os}_3$ to have a different color. **(ii)** There are three implementations of $\mathsf{APP}_1$ (e.g., Safari vs. Firefox vs. Chrome), which are respectively indicated by three colors. For example, at time $t = 0, 1, 2$, computers 1 and 5 run the same implementation of $\mathsf{APP}_1$, causing $\mathsf{app}_{1,1}$ and $\mathsf{app}_{5,1}$ to have the same color; computers 3 and 4 run two other implementations of $\mathsf{APP}_1$, causing $\mathsf{app}_{3,1}$ and $\mathsf{app}_{4,1}$ to have different colors; computer 2 does not run $\mathsf{APP}_1$. **(iii)** There are three implementations of $\mathsf{APP}_2$ (e.g., Outlook vs. Thunderbird vs. eM Client), which are respectively indicated by three colors. For example, at time $t = 0, 1, 2$, computers 4 and 5 run the same implementation of $\mathsf{APP}_2$, causing $\mathsf{app}_{4,2}$ and $\mathsf{app}_{5,2}$ to have the same color; computers 1 and 2 run two other implementations of $\mathsf{APP}_2$, causing $\mathsf{app}_{1,2}$ and $\mathsf{app}_{2,2}$ to have different colors; computer 3 does not run $\mathsf{APP}_2$.

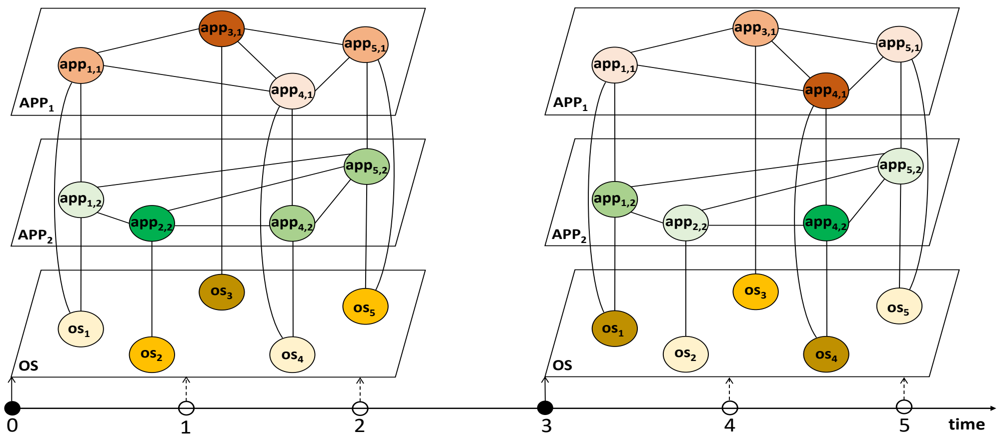


**Figure 4.1**: Illustration of dynamic diversity in a network of 5 computers ($1 \le i \le 5$). The time horizon shown is $t = 0, 1, \ldots, 5$.

Figure 4.1 illustrates dynamic diversity as follows. At time $t = 0$, the network-wide diversity is configured to run a certain combination of specific implementations of OS and applications as indicated by colors. The solid arrow at $t = 0$ indicates that a new (i.e., initial) diversity configuration is employed. This configuration remains unchanged for $t = 1, 2$, as indicated by dashed arrows at $t = 1, 2$. At time $t = 3$, the network-wide diversity is re-configured to run another combination of the diversified implementations of OS and applications. The solid arrow at $t = 3$ indicates this new employment. The configuration remains unchanged for $t = 4, 5$, as indicated by dashed arrows at $t = 4, 5$.

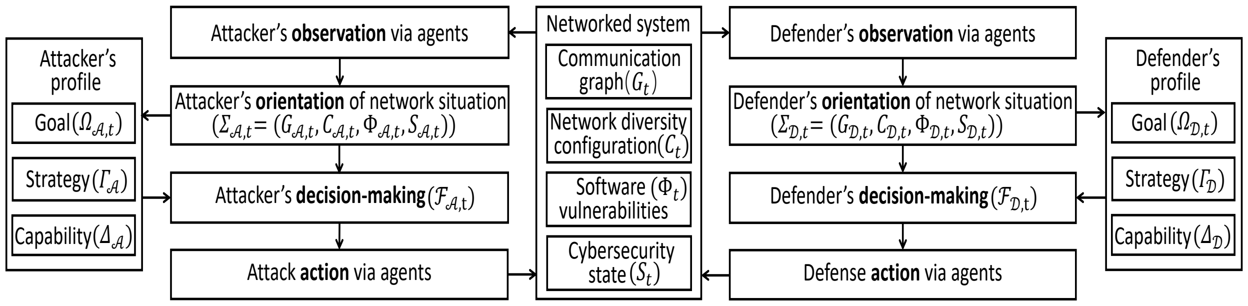

Attacker's profile
Goal($\Omega_{\mathcal{A},t}$)
Strategy($\Gamma_{\mathcal{A}}$)
Capability($\Delta_{\mathcal{A}}$)
Attacker's observation via agents
Attacker's orientation of network situation ($\Sigma_{\mathcal{A},t} = (G_{\mathcal{A},t}, C_{\mathcal{A},t}, \Phi_{\mathcal{A},t}, S_{\mathcal{A},t})$)
Attacker's decision-making($\mathcal{F}_{\mathcal{A},t}$)
Attack action via agents
Networked system
Communication graph($G_t$)
Network diversity configuration($C_t$)
Software ($\Phi_t$) vulnerabilities
Cybersecurity state ($S_t$)
Defender's observation via agents
Defender's orientation of network situation ($\Sigma_{\mathcal{D},t} = (G_{\mathcal{D},t}, C_{\mathcal{D},t}, \Phi_{\mathcal{D},t}, S_{\mathcal{D},t})$)
Defender's decision-making($\mathcal{F}_{\mathcal{D},t}$)
Defense action via agents
Defender's profile
Goal ($\Omega_{\mathcal{D},t}$)
Strategy($\Gamma_{\mathcal{D}}$)
Capability($\Delta_{\mathcal{D}}$)

**Figure 4.2**: The COODAL framework for characterizing the effectiveness of dynamic network diversity.

**Framework Overview**. Figure 4.2 highlights the Cyber Observation-Orientation-Decision-Action Loop (COODAL) framework for describing attack-defense interactions in a network. COODAL adapts the military operation concept of OODA Loop [16] to cyberspace. At a high level, the network is abstracted as a communication graph. Both attacker and defender have their own Observation (collecting data), Orientation (analyzing the collected data while leveraging relevant intelligence or information), Decision (determining what to do), and Action (executing the decision). We consider a discrete-time model with a finite time horizon $t = 0, 1, \ldots, T$, where $T$ is the defender's mission lifetime (i.e., certain security requirements must be satisfied in order to achieve mission assurance in time interval $[0, T]$). For simplicity, we assume an attack or defense action takes effect instantly (if effective). When desired, this assumption can be eliminated by explicitly modeling the delay for an action to take effect (e.g., the framework can be extended to accommodate that an attack or defense action occurs at time $t$ and takes effect at $t + 1$).

We stress that the framework does not make restrictive assumptions. For example, we do not make any restriction on how vulnerabilities may be distributed among the diversified implementations by equally accommodating the scenarios where diversified implementations may or may not have common vulnerabilities. When the attacker has an exploit against a vulnerability that is common to multiple programs, the attacker can compromise all of these vulnerable programs. A successful attack against an OS implies a successful attack against the applications running on top of the OS; a successful attack against an application paves a way for the attacker to attack the OS beneath the application (e.g., privilege escalation).

### 4.2.1 Representation of Enterprise Networks

**Representation of computers' software stacks**. A network (e.g., enterprise or mission network) consists of $n$ interconnected computers or devices. Each computer runs a *software stack*, which has two layers: application and operating system (OS). We treat any software running in the *user space* as application program. By contrast, an OS runs in the *kernel space*. Consistent with the notations used above, we use $\mathsf{APP}$ to denote the universe of application programs. As mentioned above, computer $i$ runs one or multiple applications, denoted by $\mathsf{app}_{i,j}$, where $1 \leq i \leq n$, $j$ is the index of the application, and $\mathsf{app}_{i,j} \in \mathsf{APP}$. We use $\mathsf{OS}$ to denote the universe of OSes, and use $\mathsf{os}_i$ to denote the specific OS running in computer $i$, where $\mathsf{os}_i \in \mathsf{OS}$. Let $\hbar$ denote the number of different programs running in the network.

**Representation of communications**. We explicitly model the communications between applications because they can be leveraged by the attacker to wage attacks. There are two types of communications: *intra-computer* and *inter-computer* [23]. Intra-computer communications are conducted by the programs running in a computer and can be represented as *edges* in the terminology of Graph Theory. For example, Figure 4.1 shows that the two applications running in computer 1 (i.e., $\mathsf{app}_{1,1}$ and $\mathsf{app}_{1,2}$) are designed to communicate with each other, and that both $\mathsf{app}_{1,1}$ and $\mathsf{app}_{1,2}$ can communicate with $\mathsf{os}_1$ (e.g., for making system calls). Inter-computer communications are conducted by applications running in different computers and can be represented as *edges*. Note that OSes may not communicate with each other. We distinguish intra-computer and inter-computer communications for two reasons: (i) they are often leveraged by the attacker

| | |
|---|---|
| $\mathcal{A}, \mathcal{D}$ | attacker $\mathcal{A}$ and defender $\mathcal{D}$ |
| $n, i, j$ | $n$ is the number of computers in a network, $\mathsf{app}_{i,j}$ is the $j$-th application running in computer $i \in [1, n]$ |
| $\ell, z$ | $\ell$ is the number of phases of $\mathcal{A}$'s strategy, $z \in [1, \ell]$ |
| $\hbar, k$ | $\hbar$ is the number of different programs, $k \in [1, \hbar]$ |
| $X_k$ | $X_k$ is the number of diversified implementations of program $k$ |
| $Q_k$ | software quality of diversified implementations of the $k$-th program (functionality), $Q_k \in [0, 1]$ interpreted as probability of containing vulnerability |
| $G$ | $G = (V, E)$ is the communication graph of a network (rather than the network's physical topology), where $v \in V$ represents a program (application or operating system) |
| $C_t$ | $C_t : V \to \mathsf{SW}$ is the network diversity configuration at time $t$ |
| $\phi_t$ | $\phi_t : C_t(V) \to 2^{\mathsf{VUL}}$ is the mapping from diversified programs to the vulnerabilities present in them at time $t$. $\Phi_t = (\phi_t(C_t(v)))_{v \in V}$ is the vulnerabilities in the network. |
| $s_{v,t}$ | $s_{i,t} \in \{0, 1, 2\}$ is the cybersecurity state of node (i.e., program running at) $v \in V$ at time $t$ (0: *vulnerable*; 1: *compromised*; 2: *invulnerable*). Vector $S_t = (s_{v,t})_{v \in V}$ |
| $s_{i,t}$ | $s_{i,t} \in \{0, 1, 2\}$ is the cybersecurity state of computer $i$ at time $t$, similarly defined as $s_{v,t}$. Vector $S'_t = (s_{i,t})_{i \in [1,n]}$ |
| $\Sigma_t$ | $\Sigma_t = (G, C_t, \Phi_t, S_t)$ is the cybersecurity situation of a network at time $t$ |
| $\Sigma_{\mathcal{A},t}$ | $\Sigma_{\mathcal{A},t} = (G_{\mathcal{A},t}, C_{\mathcal{A},t}, \Phi_{\mathcal{A},t}, S_{\mathcal{A},t})$ is $\mathcal{A}$'s perception of the target network $\Sigma_t$ at time $t$ |
| $\Omega_{\mathcal{A},t}$ | $\Omega_{\mathcal{A},t} = (\omega_{v,\mathcal{A},t})_{v \in V_{\mathcal{A},t}}$ is $\mathcal{A}$'s goal at time $t$ at the *program* level, where $\omega_{v,\mathcal{A},t} \in \{\perp, 1\}$; $\Omega'_{\mathcal{A},t} = (\omega_{i,\mathcal{A},t})_{i \in [1,n]}$ is $\mathcal{A}$'s goal at time $t$ at the *computer* level, where $\omega'_{i,\mathcal{A},t} \in \{\perp, 1\}$ |
| $\Gamma_{\mathcal{A}}$ | $\Gamma_{\mathcal{A}} = \{\gamma_{\mathcal{A},1}, \ldots, \gamma_{\mathcal{A},\ell}\}$ is attacker's strategy of $\ell$ phases |
| $\Delta_{\mathcal{A}}$ | $\Delta_{\mathcal{A}} = (\Delta_{\mathcal{A},z})_{z \in [1,\ell]}$ is an attacker's capability where $\Delta_{\mathcal{A},z} = \{\psi_{\mathcal{A},z,1}, \ldots, \psi_{\mathcal{A},z,m_z}\}$ is the exploits that are applicable and available to the attacker at phase $z$ |
| $\mathcal{F}_{\mathcal{A},t}$ | $\mathcal{F}_{\mathcal{A},t}$ is the attacker's decision-making algorithm to make an attack plan $\Lambda_{\mathcal{A}} = (\lambda_{\mathcal{A},1}, \ldots, \lambda_{\mathcal{A},\ell})$ at time $t$, where $\lambda_{\mathcal{A},z} \in \Delta_{\mathcal{A},z}$ for $1 \leq z \leq \ell$ |
| $\Sigma_{\mathcal{D},t}$ | $\Sigma_{\mathcal{D},t} = (G_{\mathcal{D}}, C_{\mathcal{D},t}, \Phi_{\mathcal{D},t}, S_{\mathcal{D},t})$ is $\mathcal{D}$'s perception of $\Sigma_t$ at time $t$ |
| $\Omega_{\mathcal{D},t}$ | $\Omega_{\mathcal{D},t} = (\omega_{v,\mathcal{D},t})_{v \in V}$ is $\mathcal{D}$'s goal at time $t$ at the *program* level, where $\omega_{v,\mathcal{D},t} \in [0, 1]$; $\Omega'_{\mathcal{D},t} = (\omega_{i,\mathcal{D},t})_{i \in [1,n]}$ is $\mathcal{D}$'s goal at time $t$ at the *computer* level, where $\omega_{i,\mathcal{D},t} \in [0, 1]$ |
| $\Gamma_{\mathcal{D}}$ | $\Gamma_{\mathcal{D}} = \{\gamma_{\mathcal{D},1}, \ldots, \gamma_{\mathcal{D},j}\}$ is defender's strategy |
| $\eta_{\mathcal{D},1}$ | $\eta_{\mathcal{D},1}$ represents the proportion of nodes in $V$ employing diversified implementations |
| $\eta_{\mathcal{D},2}$ | $\eta_{\mathcal{D},2}$ represents the *frequency* at which the diversified implementations will be dynamically re-employed |
| $\eta_{\mathcal{D},3}$ | $\eta_{\mathcal{D},3}$ represents the *condition* under which diversified implementations are re-employed |
| $\Delta_{\mathcal{D}}$ | $\Delta_{\mathcal{D}} = (\Delta_{\mathcal{D},k})_{k \in [1,\hbar]}$ is defender's capability, where $\Delta_{\mathcal{D},k} = \{\delta_{\mathcal{D},k,1}, \ldots, \delta_{\mathcal{D},k,X_k}\}$ is a set of $X_k$ diversified implementations of program $k$; |
| $\mathcal{F}_{\mathcal{D},t}$ | the decision-making algorithm to make a defense plan $\Lambda_{\mathcal{D},t} = (\delta_{\mathcal{D},t}(v))_{v \in V}$ at time $t$ |
| $\tau$ | the compromise probability of computer $i \in [1, n]$ at time $t \in [0, T]$ that can be tolerated by $\mathcal{D}$ |
| TTS | $\mathsf{TTS}(\mathcal{A}, \mathcal{D}) = \min\{t : \mathsf{cc}(t) > \tau\}$ measures how long it takes for attacker $\mathcal{A}$ to break defender $\mathcal{D}$'s mission goal $\tau$ |
| ASD | $\mathsf{ASD}_{\mathcal{D},q} = \mathsf{TTS}(\mathcal{A}, \mathcal{D}_q) - \mathsf{TTS}(\mathcal{A}, \mathcal{D}_1)$ measures the extent at which $\mathcal{A}$ is slowed down by defense strategy $\gamma_{\mathcal{D},q}$ |
| AWD | $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}, T) = \max\{\mathsf{cc}(t) : t \leq T\}$ measures attack worst damage during $t \in [0, T]$ |
| AEC | $\mathsf{AEC}_{\mathcal{D},q} = \mathsf{AI}(\gamma_{\mathcal{D},q}) - \mathsf{AI}(\gamma_{\mathcal{D},1})$ measures the number of extra exploits $\mathcal{A}$ needs to obtain to make $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T) > \tau$ against defense strategy $\gamma_{\mathcal{D},q}$ |
| VT | $\mathsf{VT}_{\mathcal{D},q} = \max\{Q : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T) \leq \tau\}$ captures the upper bound of tolerable vulnerabilities such that $\mathcal{D}$ can still achieve its mission goal $\tau$ in lifetime $[0, T]$ |
| AOC | $\mathsf{AOC}_{\mathcal{D},q}(T) = \sum_{t=1}^{T} \mathsf{OC}_{\mathcal{D},q}(t)/T$ is the average operational cost to achieve the defender's goal; $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},q} = \max\{\mathsf{AOC}_{\mathcal{D},q}(T) : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T) \leq \tau\}$ and $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},q} = \min\{\mathsf{AOC}_{\mathcal{D},q}(T) : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T) \leq \tau\}$ are maximum and minimum $\mathsf{AOC}_{\mathcal{D},q}(T)$, respectively. |

**Table 4.1**: Summary of key notations used for quantifying security effectiveness of dynamic network diversity.

for different purposes — the former for privilege escalation and the latter for lateral movement between computers; and (ii) they are defended by different security mechanisms — the former is defended by host-based mechanisms (e.g., intrusion prevention) and the latter is defended by network-based mechanisms (e.g., firewall).

Formalizing the preceding discussion, we naturally obtain the notion of *communication graph* $G = (V, E)$, where each vertex or node $v \in V$ represents (and runs) a program and each edge $(u, v) \in E$ represents that a pair of nodes are permitted to communicate with each other. Note that a computer is represented by a set of nodes in $G$ because it runs a set of programs. Since *programs* run at *nodes* in $G$, we use *programs* and *nodes* interchangeably to make succinct statements. We use the term *edges* in the standard way to indicate *undirected* graphs; a communication graph can be *directed* in principle, which is however rare in practice. In the example illustrated in Figure 4.1, where have $|V| = 13$ (i.e., 4 nodes running $\mathsf{APP}_1$, 4 nodes running $\mathsf{APP}_2$, and 5 nodes running OS), and the edge set $E$ is as illustrated.

We stress that communication graph $G = (V, E)$ is different from a networking-induced graph, for two reasons. First, a vertex or node in a communication graph represents a program running in a computer and each edge represents the communication between two programs. In contrast, a node in a networking-induced graph often represents a computer, which can run multiple programs. Second, a communication graph can encode access control policies, which may regulate which programs are (not) allowed to communicate with which other programs running in the same computer or different computers. This means that a communication graph may not be a complete graph because some programs may only be allowed to communicate with some of the others. Whereas, a networking-induced graph cannot encode access control policies and would be a complete graph because any computer can communicate with any other computer as long as they are routable.

In this paper we assume a communication graph $G = (V, E)$ is *time-independent*, meaning that the applications running in a computer are fixed. This means that the applications running in a computer do not change, but their specific implementations may change over time. This is plausible because we focus on quantifying the effectiveness of dynamic network diversity. Nevertheless, $G = (V, E)$ can be extended to time-dependent $G_t = (V_t, E_t)$ when desired.

**Representation of network diversity configuration**. Let $\mathsf{SW}$ denote the set of diversified implementations of the $\hbar$ programs running in the network, including applications and OSes. The *dynamic* network diversity configuration at time $t$ refers to the mapping from the set of programs to their specific implementations. This mapping can be described by a function $C_t : V \to \mathsf{SW}$ such that $C_t(v)$ is a specific implementation of the program running at node $v \in V$ at time $t$. For example, Figure 4.1 shows that there are 3 different programs (i.e., $\mathsf{APP}_1$, $\mathsf{APP}_2$, OS); each program has 3 diversified implementations (indicated by 3 different colors) and thus $|\mathsf{SW}| = 9$, where computer 1 runs a specific implementation of each of the three programs at $t \in [0, 2]$. Note that the preceding representation is general enough to accommodate the *static* network diversity, which corresponds to $C_0 = C_1 = \ldots = C_T$, and the *monolithic* software stack, which corresponds to each program having exactly one implementation (i.e., $|\mathsf{SW}|$ equals the number of different programs and $C_0 = C_1 = \ldots = C_T$).

**Representation of network-wide cybersecurity state and situation**. Cybersecurity state and situation can be defined at the *program* of $v \in V$ and at the computer level. The former is suitable for modeling and simulation purposes, and the latter is more suitable for cyber defense operation and management purposes. So, we consider both.

At the program level, we consider a communication graph $G = (V, E)$, which abstracts a network as described above. We use $s_{v,t} \in \{0, 1, 2\}$ to denote the state of node $v \in V$ at time $t$: $s_{v,t} = 0$ means $v$ (i.e., the program running at $v$) is *vulnerable* but not compromised, namely that $v$ contains a vulnerability but the vulnerability is not exploited by the attacker and the underlying OS is not compromised; $s_{v,t} = 1$ means $v$ is *compromised* either because its vulnerability is exploited or because the underlying OS is compromised; and $s_{v,t} = 2$ means $v$ is *invulnerable* (i.e., containing no vulnerabilities) and the underlying OS is not compromised. The *program-level* network-wide *cybersecurity state* at time $t$ is represented by vector $S_t = (s_{v,t})_{v \in V}$.

At the computer level, we say computer $i$ is *vulnerable* (or $s_{i,t} = 0$) if any program running in the computer is vulnerable, *compromised* (or $s_{i,t} = 1$) if any program running in the computer is compromised, and *invulnerable* (or $s_{i,t}$=2) if all programs running in the computer are invulnerable. Similarly, we can define *computer-level* network-wide cybersecurity state at time $t$ as $S'_t = (s_{i,t})_{i \in [1,n]}$, where $S'_t$ can be derived from $G$ and $S_t$. We further define the vector $((\mathsf{vc}(t), \mathsf{cc}(t), \mathsf{ic}(t)))$ to succinctly describe the computer-level network-wide cybersecurity effect at time $t$, where $\mathsf{vc}(t) = |\{i : s_{i,t} = 0\}|/n$ is the fraction of *vulnerable* computers, $\mathsf{cc}(t) = |\{i : s_{i,t} = 1\}|/n$ is the fraction of *compromised* computers, and $\mathsf{ic}(t) = |\{i : s_{i,t} = 2\}|/n$ is the fraction of *invulnerable* computers. Note that $\mathsf{cc}(t) + \mathsf{vc}(t) + \mathsf{ic}(t) = 1$.

The network-wide *cybersecurity situation* can be described by $\Sigma_t = (G, C_t, \Phi_t, S_t)$, or a tuple of the communication graph, the network diversity configuration, the set of vulnerabilities associated with each implementation, and the network-wide cybersecurity state. Note that we do not mention $S'_t$ because it can be derived from $S_t$.

### 4.2.2 Representation of Vulnerabilities

The software running in a computer, application and OS alike, may contain vulnerabilities. We use $\mathsf{VUL}$ to denote the universe of software vulnerabilities that may be present in the software stacks running in the computers of the network in question. Since vulnerabilities are associated with the nodes in $V$, we use function $\phi_t : C_t(V) \rightarrow 2^{\mathsf{VUL}}$ to describe the set of vulnerabilities that are present in the implementations of application and OS programs running in the computers, where $\phi_t(C_t(v)) = \emptyset$ means the implementation running at node $v$ is not vulnerable. We use vector $\Phi_t = (\phi_t(C_t(v)))_{v \in V}$ to denote the *ground-truth* vulnerabilities that are present in the software running in computers at time $t$. Note that this ground-truth vulnerability set may or may not be known to the attacker or defender, and that defining $\phi_t : C_t(V) \rightarrow 2^{\mathsf{VUL}}$ is equivalent to defining it as a function from the software set $\mathsf{SW}$ to $2^{\mathsf{VUL}}$. We choose the former because it simplifies subsequent discussion (in regards to dynamic network diversity). Let $Q_k \in [0, 1]$, $1 \leq k \leq \hbar$, denote the quality of program, which is measured by the ratio of the number of vulnerable implementations to the total number of diversified implementations of program $k$. The parameter $Q_k$, which is interpreted as the probability that program $k$ is vulnerable, would depend on the technologies that are employed to reduce vulnerabilities in the course of software development (e.g., vulnerability detection [80–82, 84, 169]). The smaller the $Q_k$, the higher the diversity quality of program $k$.

### 4.2.3 Representation of Attacks

We model an attacker $\mathcal{A}$ (i.e., threat model) with five attributes: *knowledge* (what $\mathcal{A}$ knows about a network), *goal* (what $\mathcal{A}$ attempts to achieve), *strategy* (what strategy $\mathcal{A}$ uses), *capability* (what exploits $\mathcal{A}$ possesses), and *decision-making* (i.e., what decision-making algorithms $\mathcal{A}$ uses). Intuitively, $\mathcal{A}$ with a certain knowledge attempts to achieve a goal by leveraging some capabilities to compromise some nodes or computers according to some strategies.

**Representation of attacker's knowledge**. We define attacker $\mathcal{A}$'s knowledge as vector $\Sigma_{\mathcal{A}} = (\Sigma_{\mathcal{A},t})_{t\in[0,T]}$ such that $\Sigma_{\mathcal{A},t} = (G_{\mathcal{A},t}, C_{\mathcal{A},t}, \Phi_{\mathcal{A},t}, S_{\mathcal{A},t})$ is $\mathcal{A}$'s perception of the target network $\Sigma_t = (G, C_t, \Phi_t, S_t)$ at time $t$, where $G_{\mathcal{A},t} = (V_{\mathcal{A},t}, E_{\mathcal{A},t}) \subseteq G$ is the attacker's perception of $G$, $C_{\mathcal{A},t}$ is the attacker's perception of $C_t$, $\Phi_{\mathcal{A},t}$ is the attacker's perception of $\Phi_t$, and $S_{\mathcal{A},t}$ is the attacker's perception of $S_t$. Note that $S_{\mathcal{A},t} = S_t$ because the attacker knows which programs are compromised by the attacker itself. Note also that the notion of *initial compromise* can be modeled as part of the attacker's knowledge at time $t = 0$, because it describes which nodes $v \in V$ are compromised at $t = 0$. We use $\mathsf{IniComp} = \{v \in V : s_{v,0} = 1\}$ to denote the set of programs that are compromised at $t = 0$.

**Representation of attacker's goal**. We define attacker $\mathcal{A}$'s goal as vector $\Omega_{\mathcal{A}} = (\Omega_{\mathcal{A},t})_{t\in[0,T]}$ where vector $\Omega_{\mathcal{A},t} = (\omega_{v,\mathcal{A},t})_{v\in V_{\mathcal{A},t}}$ is $\mathcal{A}$'s goal at time $t$, where $\omega_{v,\mathcal{A},t} \in \{\perp, 1\}$, $\omega_{v,\mathcal{A},t} = \perp$ means that $\mathcal{A}$ does not care about the state of $v \in V_{\mathcal{A},t}$ at time $t$, $\omega_{v,\mathcal{A},t} = 1$ means $\mathcal{A}$ attempts to make $v \in V$ compromised at time $t$. Since $\Omega'_{\mathcal{A},t} = (\omega_{i,\mathcal{A},t})_{i\in[1,n]}$ can be derived from $\Omega_{\mathcal{A},t}$, in what follows we do not have to mention $\Omega'_{\mathcal{A},t}$ explicitly. This representation is flexible because it can accommodate intuitive attack goals, such as: (i) attempting to compromise a fixed set of nodes or computers at time $t$; (ii) attempting to compromise as many nodes or computers as possible at time $t$; and (iii) attempting to cumulatively compromise as many nodes or computers as possible at time

$t$.

**Representation of attacker's strategy**. Inspired by the state-of-the-art industrial characterization of sophisticated cyber attacks, such as the Cyber Kill Chain [65] and Mitre's ATT&CK [**?**], we propose abstracting them into attacker's strategy. Since strategy is often fixed for $t \in [0, T]$, we specify attacker $\mathcal{A}$'s strategy via $\ell \geq 1$ phases, denoted by $\Gamma_{\mathcal{A}} = \{\gamma_{\mathcal{A},1}, \ldots, \gamma_{\mathcal{A},\ell}\}$. For example, we have $\ell = 7$ for the Cyber Kill Chain and $\ell = 12$ for Mitre's ATT&CK version 7. Since there are different kinds of strategies, we will demonstrate how to use a specific strategy in our case study.

**Representation of attacker's capability**. Given an $\ell$-phase strategy $\Gamma_{\mathcal{A}}$, attacker $\mathcal{A}$'s capability at phase $z$, where $z \in [1, \ell]$, is defined as a set of $m_z$ exploits applicable at phase $z$ and available to $\mathcal{A}$, denoted by $\Delta_{\mathcal{A},z} = \{\psi_{\mathcal{A},z,1}, \ldots, \psi_{\mathcal{A},z,m_z}\}$. We define $A$'s capability as vector $\Delta_{\mathcal{A}} = (\Delta_{\mathcal{A},z})_{z \in [1,\ell]}$. Since $\mathcal{A}$'s capability depends on exploits, we define $\mathcal{A}$'s *investment* as $\mathsf{AI} = \sum_{z=1}^{\ell} \sum_{q=1}^{m_z} cost_{\mathcal{A},z,q}$, where $cost_{\mathcal{A},z,q}$ is the cost to obtain the $q$-th exploit that is applicable at phase $z$.

**Representation of attacker's decision-making**. At time $t$, attacker $\mathcal{A}$ uses a decision-making algorithm $\mathcal{F}_{\mathcal{A},t}$ to make an attack plan, which specifies the utilization of some of $\mathcal{A}$'s capabilities, denoted by $\Lambda_{\mathcal{A},t} = (\lambda_{\mathcal{A},1}, \ldots, \lambda_{\mathcal{A},\ell})$ where $\lambda_{\mathcal{A},z} \in \Delta_{\mathcal{A},z}$ for $1 \leq z \leq \ell$. This can be denoted by

$$\lambda_{\mathcal{A},z} \leftarrow \mathcal{F}_{\mathcal{A},t}(\Sigma_{\mathcal{A},t}, \Omega_{\mathcal{A},t}, \Gamma_{\mathcal{A}}, \Delta_{\mathcal{A},z}), \tag{4.1}$$

where $\Sigma_{\mathcal{A},t} = (G_{\mathcal{A},t}, C_{\mathcal{A},t}, \Phi_{\mathcal{A},t}, S_{\mathcal{A},t})$ is $\mathcal{A}$'s knowledge at time $t$ as described above, $\Omega_{\mathcal{A},t}$ is $\mathcal{A}$'s goal at time $t$, $\Gamma_{\mathcal{A}}$ is $\mathcal{A}$'s strategy, and $\Delta_{\mathcal{A},z}$ is $\mathcal{A}$'s capability at phase $z$ of the attack strategy. Note the Eq. (4.1) can be extended to consider multiple exploits (rather than a single exploit $\lambda_{\mathcal{A},z}$) that may be used in an appropriate manner (e.g., sequential).

Putting the description together, we denote attacker $\mathcal{A} = (\mathcal{A}_t)_{t \in [0,T]}$ with $\mathcal{A}_t = (\Sigma_{\mathcal{A},t}, \Omega_{\mathcal{A},t}, \Gamma_{\mathcal{A}}, \Delta_{\mathcal{A}}, \mathcal{F}_{\mathcal{A},t})$.

### 4.2.4 Representation of Defenses

Similar to the attacker (or threat) model, we describe a defender $\mathcal{D}$ via five attributes: *knowledge* (what $\mathcal{D}$ knows), *goal* (what $\mathcal{D}$ aims to achieve), *strategy*, *capability* (what tools $\mathcal{D}$ possesses), and *decision-making* (what algorithms $\mathcal{D}$ uses).

**Representation of defender's knowledge**. We define defender $\mathcal{D}$'s knowledge as vector $\Sigma_{\mathcal{D}} = (\Sigma_{\mathcal{D},t})_{t\in[0,T]}$ such that $\Sigma_{\mathcal{D},t} = (G_{\mathcal{D}}, C_{\mathcal{D},t}, \Phi_{\mathcal{D},t}, S_{\mathcal{D},t})$ is $\mathcal{D}$'s perception of the ground-truth situation $\Sigma_t = (G, C_t, \Phi_t, S_t)$ at time $t$. In the case of *full* knowledge, we have $\Sigma_{\mathcal{D},t} = \Sigma_t$, meaning the defender knows everything about the ground-truth situation. In the more realistic case of *partial* knowledge, the defender only knows: (i) the communication graph $G$, namely $G_{\mathcal{D}} = G$ because the network is managed by the defender; (ii) the network configuration $C_t$, namely $C_{\mathcal{D},t} = C_t$ because the defender decides which nodes run which specific implementations; (iii) some information about the ground-truth vulnerabilities associated with the programs, namely $\Phi_{\mathcal{D},t} \subseteq \Phi_t$ because the defender may not know the 0-day ones that are known to the attacker; and (iv) some noisy information about the network's ground-truth cybersecurity state $S_t$ because of the false-positives and/or false-negatives in measuring or inferring cybersecurity states.

**Representation of defender's goal**. Corresponding to the program-level vs. computer-level distinction, $\mathcal{D}$'s goal can be defined at two levels. At the *program* level of $v \in V$, we define $\mathcal{D}$'s goal as vector $\Omega_{\mathcal{D}} = (\Omega_{\mathcal{D},t})_{t\in[0,T]}$ such that $\Omega_{\mathcal{D},t} = (\omega_{v,\mathcal{D},t})_{v\in V}$ is $\mathcal{D}$'s goal at time $t$, where $\omega_{v,\mathcal{D},t} \in [0,1]$ is the tolerable probability that program running at $v \in V$ is compromised at time $t$. For example, $\omega_{v,\mathcal{D},t} = 0$ means that a successful attack against $v$ cannot be tolerated; $\omega_{v,\mathcal{D},t} = 0.5$ can be interpreted as that compromise of $v$ for at most 50% of the time can be tolerated. At the *computer* level, we define $\mathcal{D}$'s goal as vector $\Omega'_{\mathcal{D}} = (\Omega'_{\mathcal{D},t})_{t\in[0,T]}$ such that $\Omega'_{\mathcal{D},t} = (\omega_{i,\mathcal{D},t})_{i\in[1,n]}$ is $\mathcal{D}$'s goal at time $t$, where $\omega_{i,\mathcal{D},t} \in [0,1]$ is the tolerable probability that computer $i$ is compromised at time $t$. One computer-level goal of particular interest is: $\omega_{i,\mathcal{D},t} \leq 1/3$ for $i \in [1,n]$ and $t \in [0,T]$; it describes cyber defense using Byzantine fault-tolerance techniques to tolerate the compromise of a certain threshold of computers [91]. Since $\Omega'_{\mathcal{D}}$ can be derived from $\Omega_{\mathcal{D}}$, we do not have to mention $\Omega'_{\mathcal{D}}$ except when we discuss computer-level effectiveness.

**Representation of defender's strategy**. We specify defender $\mathcal{D}$'s strategies in employing network diversity as a set $\Gamma_{\mathcal{D}} = \{\gamma_{\mathcal{D},1}, \ldots, \gamma_{\mathcal{D},\jmath}\}$. For example, $\gamma_{\mathcal{D},1}$ represents monoculture software stacks (i.e., the baseline strategy), $\gamma_{\mathcal{D},2}$ represents static diversity, $\gamma_{\mathcal{D},3}$ represents proactive diversity with fixed intervals, $\gamma_{\mathcal{D},4}$ represents reactive-adaptive diversity where the employment is triggered by security alerts, and $\gamma_{\mathcal{D},5}$ represents hybrid diversity (i.e., a combination of proactive diversity and reactive-adaptive diversity). A strategy can be accompanied by *some* of the following parameters. (i) the *proportion* of nodes in $V$ (re-)employing diversified implementations (e.g., all or some nodes), denoted by $\eta_{\mathcal{D},1}$; (ii) the *frequency* at which the diversified implementations will be dynamically re-employed at the nodes, denoted by $\eta_{\mathcal{D},2}$; (iii) the *condition* under which diversified implementations are re-employed, denoted by $\eta_{\mathcal{D},3}$. As an example showing that not every parameter is relevant to every strategy, we note that the preceding (ii) and (iii) are not relevant to the static diversity strategy; this can be indicated by setting $\eta_{\mathcal{D},2} = \texttt{NULL}$ and $\eta_{\mathcal{D},3} = \texttt{NULL}$.

**Representation of defender's capability**. We define defender $\mathcal{D}$'s capability as the diversified implementations that are available to $\mathcal{D}$. Recall that $\hbar$ different programs running in the network (including both applications and operation systems) and each program may have diversified implementations. $\mathcal{D}$'s capability with respect to program $k$, where $1 \leq k \leq \hbar$, is a set of $X_k$ diversified implementations, denoted by $\Delta_{\mathcal{D},k} = \{\delta_{\mathcal{D},k,1}, \ldots, \delta_{\mathcal{D},k,X_k}\}$. We define $\mathcal{D}$'s capability as the vector of diversified implementations that are available to $\mathcal{D}$, denoted by $\Delta_{\mathcal{D}} = (\Delta_{\mathcal{D},k})_{k \in [1,\hbar]}$. Since $\mathcal{D}$'s capability depends on the diversified implementations, we define the defender's investment as $\mathsf{DI} = \sum_{k=1}^{\hbar} \sum_{w=1}^{X_k} cost_{\mathcal{D},k,w}$, where $cost_{\mathcal{D},k,w}$ is the cost for obtaining the $w$-th diversified implementation of program $k$.

**Representation of defender's decision-making**. At time $t$, $\mathcal{D}$ uses decision-making algorithm $\mathcal{F}_{\mathcal{D},t}$ to make a defense plan $\Lambda_{\mathcal{D},t} = (\delta_{\mathcal{D},t}(v))_{v \in V}$, which specifies how to employ the diversified implementations of the $\hbar$ programs at nodes $v \in V$ at time $t$. Formally, $\Lambda_{\mathcal{D},t}$ is the output of the defender's decision-making algorithm $\mathcal{F}_{\mathcal{D},t}$ on a number of inputs, denoted by

$$\Lambda_{\mathcal{D},t} \leftarrow \mathcal{F}_{\mathcal{D},t}(\Sigma_{\mathcal{D},t}, \Omega_{\mathcal{D},t}, \Gamma_{\mathcal{D}}, \Delta_{\mathcal{D}}), \tag{4.2}$$

where $\Sigma_{\mathcal{D},t}$ is the defender's perception of the ground-truth situation $\Sigma_t = (G, C_t, \Phi_t, S_t)$ at time $t$, $\Omega_{\mathcal{D},t}$ is the defender's goal, $\Gamma_{\mathcal{D}}$ is defender's strategy as described above, and $\Delta_{\mathcal{D}}$ is the defender's capability.

Putting the description together, we denote defender $\mathcal{D} = (\mathcal{D}_t)_{t\in[0,T]}$ with $\mathcal{D}_t = (\Sigma_{\mathcal{D},t}, \Omega_{\mathcal{D},t}, \Gamma_{\mathcal{D}}, \Delta_{\mathcal{D}}, \mathcal{F}_{D,t})$.

### 4.2.5 Metrics for Measuring the Cybersecurity State of an Enterprise Network

**Measuring local effect of dynamic network diversity at the *node* level**. At any point in time, a program (i.e., node $v \in V$) is in one of the following three states: *vulnerable*, meaning that the program contains a vulnerability but the vulnerability has not been exploited by the attacker; *invulnerable*, meaning that the program contains no vulnerability; and *compromised*, either because the program contains a vulnerability that has been exploited, or because the underlying OS is compromised (causing any application program running on top of it to be compromised, no matter whether the program contains vulnerability or not).

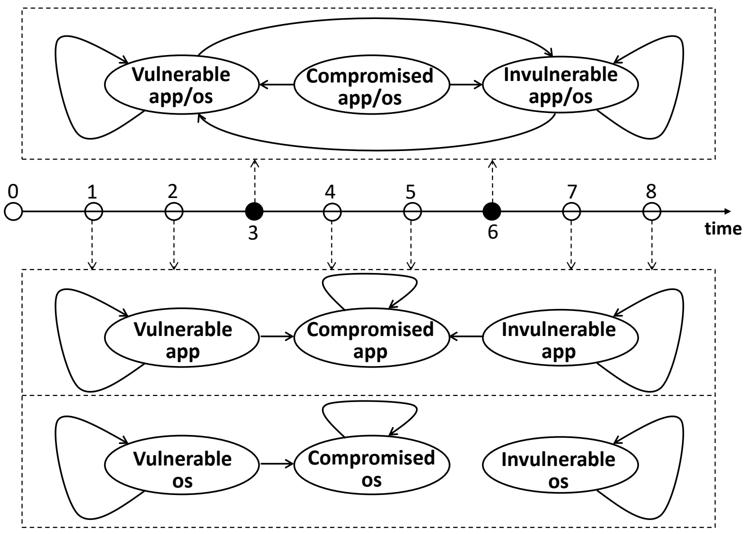


**Figure 4.3**: Modeling the local effect of dynamic network diversity at the program level (i.e., program running at $v \in V$).

Figure 4.3 highlights the effect of employing dynamic diversity at the program level. Suppose at time $t = 3, 6$ dynamic diversity is employed such that a different implementation of a program (i.e., application or OS) is employed at time $t$ to replace the implementation employed at time

$t-1$. The employment of dynamic diversity at time $t$ leads to one of the following state transitions (see upper half of Figure 4.3): (i) a vulnerable program is replaced with another vulnerable or an invulnerable program of the same functionality; (ii) an invulnerable program is replaced with a vulnerable or another invulnerable program; (iii) a compromised program is replaced with a vulnerable or invulnerable program. However, a vulnerable or invulnerable program is never replaced with a compromised program because a successful attack only occurs to a vulnerable program that has been employed and exploited (as diversified implementations are stored in a secure environment).

At any other point in time than the employment of dynamic diversity (e.g., $t \in \{1,2,4,5,7,8\}$ as shown in Figure 4.3), the security state transition of application programs is different from that of OSes. For applications, the state transitions are: (i) a vulnerable application program stays vulnerable or become compromised, either because its vulnerability is exploited or the underlying OS is compromised; (ii) a compromised application program stays in the compromised state because we focus on the defense based on dynamic diversity, without considering reactive defense that may detect and clean up the compromised application programs; (iii) an invulnerable application program stays invulnerable or becomes compromised because the underlying OS is compromised. For OSes, the state transitions are: (i) a vulnerable OS stays vulnerable or become compromised because its vulnerability is exploited; (ii) a compromised OS stays in the compromised state because we do not consider reactive defense; (iii) an invulnerable OS stays invulnerable.

**Measuring the global effect of dynamic network diversity at the *computer* level**. We quantify the global effect at the computer level via the following metrics.

**Definition 1** (time-to-succeed or TTS)**.** *This metric measures how long it takes attacker $\mathcal{A}$ to break defender $\mathcal{D}$'s goal specified by a program-level vector $\Omega_{\mathcal{D},t} = (\omega_{v,\mathcal{D},t})_{v \in V}$ or* computer-*level vector $\Omega'_{\mathcal{D},t} = (\omega_{i,\mathcal{D},t})_{i \in [1,n]}$, where $t \in [0,T]$, and $\omega_{v,\mathcal{D},t} \in [0,1]$ ($\omega_{i,\mathcal{D},t} \in [0,1]$) is the compromise probability of program $v$ (computer $i$) at time $t$ that can be tolerated. As mentioned above, a defense goal of particular interest is $\omega_{i,\mathcal{D},t} \leq 1/3$ for computers $i \in [1,n]$ at any time $t \in [0,T]$ because such compromises can be tolerated by Byzantine fault-tolerant techniques [91]. Formally,* *we define* $\mathsf{TTS}(\mathcal{A}, \mathcal{D}) = \min\{t : \mathsf{cc}(t) > \tau\}$*, where* $\mathsf{cc}(t)$ *is the fraction of compromised computers (i.e., attack damage) at time* $t$*.*

The *time-to-succeed* metric is reminiscent of the well-known *mean-time-to-compromise* metric. However, the former is defined as a random variable with respect to a specific goal (e.g., breaking defender's goal), where randomness is rooted in different attack strategies, capabilities and decision-makings algorithms. In contrast, the latter is defined as a number (or the mean value of a random variable).

**Definition 2** (attacker slow-down or ASD)**.** *Consider attacker* $\mathcal{A}$ *(i.e., a fixed threat model) that attempts to break defender* $\mathcal{D}$*'s goal* $\Omega'_{\mathcal{D}} = (\Omega'_{\mathcal{D},t})_{t\in[0,T]}$*, where* $\Omega'_{\mathcal{D},t} = (\omega_{i,\mathcal{D},t})_{i\in[1,n]}$ *and, for concreteness,* $\omega_{i,\mathcal{D},t} \leq \tau$ *is the tolerable compromise probability for every computer* $i \in [1, n]$ *and* $t \in [0, T]$*, while recalling that* $\tau = 1/3$ *corresponds to assuring the assumption that is needed by Byzantine fault-tolerant techniques [91]. We define* attacker slow-down *as* $\mathsf{ASD}_{\mathcal{D},q} = \mathsf{TTS}(\mathcal{A}, \mathcal{D}_q) - \mathsf{TTS}(\mathcal{A}, \mathcal{D}_1)$*, to measure the extent at which* $\mathcal{A}$ *is slowed down by the employment of network diversity defense strategy* $\gamma_{\mathcal{D},q}$ *(denoted by* $\mathcal{D}_q$*), where* $2 \leq q \leq 5$ *in this study, when compared with the baseline strategy* $\gamma_{\mathcal{D},1}$ *of monoculture software stacks (denoted by* $\mathcal{D}_1$*).*

**Definition 3** (attack worst damage or AWD)**.** *This metric measures how much damage an attacker* $\mathcal{A}$ *can cause in the worst case during* $t \in [0, T]$*, namely the maximum fraction of compromised computers at any time* $t \in [0, T]$*. Formally, we define* attack worst damage *as* $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}, T) = \max\{\mathsf{cc}(t) : t \leq T\}$ *where* $G, \hbar, X, Q$ *are defined above.*

**Definition 4** (attack extra cost or AEC)**.** *Consider attacker* $\mathcal{A}$ *(i.e., a fixed threat model) attempting to break defender* $\mathcal{D}$*'s goal* $\Omega'_{\mathcal{D}} = (\Omega'_{\mathcal{D},t})_{t\in[0,T]}$*, where* $\Omega'_{\mathcal{D},t} = (\omega_{i,\mathcal{D},t})_{i\in[1,n]}$ *and, for concreteness,* $\omega_{i,\mathcal{D},t} \leq \tau$ *is the tolerable compromise probability for computer* $i \in [1, n]$ *and* $t \in [0, T]$*. We define* attack extra cost *(*AEC*) metric,* $\mathsf{AEC}_{\mathcal{D},q} = \mathsf{AI}(\gamma_{\mathcal{D},q}) - \mathsf{AI}(\gamma_{\mathcal{D},1})$*, to measure the number of extra exploits* $\mathcal{A}$ *needs to obtain (i.e., extra investment) such that* $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T) > \tau$ *when* $\mathcal{D}$ *employs strategy* $\gamma_{\mathcal{D},q}$ *(i.e.,* $\mathsf{AI}(\gamma_{\mathcal{D},q})$*) than the baseline strategy of employing monoculture software stacks (i.e.,* $\mathsf{AI}(\gamma_{\mathcal{D},1})$*).*

Inspired by the notion of *fault-tolerance*, we define a *vulnerability-tolerance* metric to capture the upper bound of vulnerabilities in the network that can be tolerated by the defender $\mathcal{D}$ in achieving its goal $\Omega_{\mathcal{D}}$. The importance of this metric can be seen as follows. (i) When each computer is vulnerable with probability at least $1/3+\epsilon$ for some $\epsilon$, the attacker able to compromise all of them can render Byzantine fault-tolerance techniques useless. (ii) When dynamic network diversity is employed, the attacker may only be able to compromise 1/3 of the computers because the diversity configuration may have changed before the attacker compromises all of the vulnerable computers. Intuitively, the larger the $\epsilon$, the higher the vulnerability tolerance.

**Definition 5** (vulnerability-tolerance or $\mathsf{VT}$)**.** *Consider defender goal $\omega_{i,\mathcal{D},t} \leq \tau$ for every computer $i \in [1,n]$ and each time $t \in [0,T]$ and a fixed diversification quality $Q \in [0,1]$ for diversified implementation (when applying the same quality-enhancement techniques), we define $\mathsf{VT}_{\mathcal{D},q} = \max\{Q : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,T) \leq \tau\}$ with $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,T)$ specified in Definition 3.*

**Definition 6** (average operational cost or $\mathsf{AOC}$)**.** *We define the* operational cost *incurred by defender's strategy $\gamma_{\mathcal{D},q}$ at time $t$, denoted by $\mathsf{OC}_{\mathcal{D},q}(t) \in [0,1]$, as the fraction of programs that are replaced at time $t$, where $2 \leq q \leq 5$. We define the* average operational cost *up to time $T$ as $\mathsf{AOC}_{\mathcal{D},q}(T) = \sum_{t=1}^{T} \mathsf{OC}_{\mathcal{D},q}(t)/T$, which refers to the average fraction of programs that are re-installed at each time $t \in [1,T]$. Intuitively, the larger the $\mathsf{AOC}_{\mathcal{D},q}(T)$, the higher the operational cost. Given a defender's goal $\omega_{i,\mathcal{D},t} \leq \tau$ for computer $i \in [1,n]$ and $t \in [0,T]$, we define $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},q} = \max\{\mathsf{AOC}_{\mathcal{D},q}(T) : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,T) \leq \tau\}$ and $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},q} = \min\{\mathsf{AOC}_{\mathcal{D},q}(T) : \mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,T) \leq \tau\}$ as the maximum and minimum average operational cost to meet the defender's goal, respectively.*

Figure 4.4 illustrates the relationship between the metrics. The effectiveness metrics $\mathsf{TTS}$, $\mathsf{ASD}$, $\mathsf{AWD}$, $\mathsf{AEC}$ and $\mathsf{VT}$ depend on the attacker's goal in $\tau$, mission lifetime $T$, and attack damage $\mathsf{cc}(t)$. The $\mathsf{cc}(t)$ depends on the attack investment $\mathsf{AI}$, the defense investment $\mathsf{DI}$, and the defender's operational cost $\mathsf{AOC}$. The attack investment $\mathsf{AI}$ depends on the number of attack phases ($\ell$), the number of exploits applicable at each phase ($m_z$), and the cost to obtain exploits ($cost_{\mathcal{A},z,q}$).

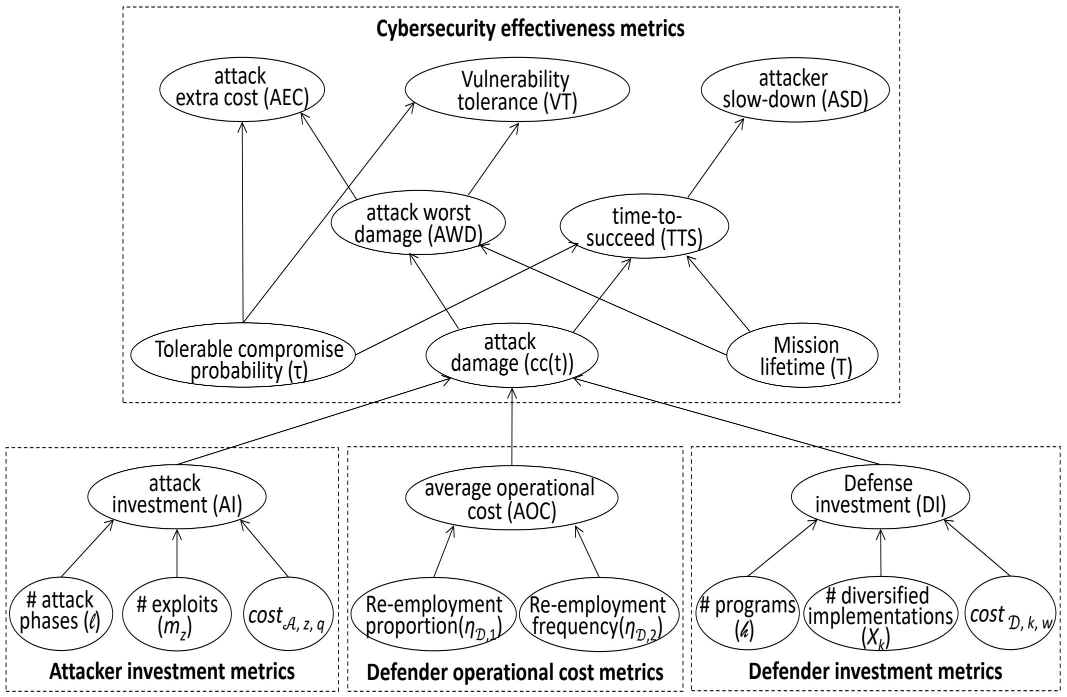

Cybersecurity effectiveness metrics
attack extra cost (AEC)
Vulnerability tolerance (VT)
attacker slow-down (ASD)
attack worst damage (AWD)
time-to-succeed (TTS)
Tolerable compromise probability (τ)
attack damage (cc(t))
Mission lifetime (T)
attack investment (AI)
average operational cost (AOC)
Defense investment (DI)
# attack phases (ℓ)
# exploits (m_z)
cost_{A, z, q}
Re-employment proportion(η_{D,1})
Re-employment frequency(η_{D,2})
# programs (ℏ)
# diversified implementations (X_k)
cost_{D, k, w}
Attacker investment metrics
Defender operational cost metrics
Defender investment metrics

**Figure 4.4**: Illustration of the relationships between the metrics, where "$X \rightarrow Y$" means $X$ is a factor in determining $Y$.

The defense investment depends on the number of programs ($\hbar$), the number of diversified implementations of each program ($X_k$), and the cost to obtain each implementation ($cost_{\mathcal{D},k,w}$). The defender's average operational cost AOC depends on the proportion $\eta_{\mathcal{D},1}$ and frequency $\eta_{\mathcal{D},2}$ of re-employment.

## 4.3 Simulation Experiments and Results

In order to characterize the effectiveness of dynamic network diversity, we propose investigating the following RQs by simulations, where RQ1-RQ3 correspond to defender's gains, RQ4-RQ5 corresponds to defender's cost.

- RQ1: To what extent can dynamic network diversity slow down the attacker?
- RQ2: How much extra cost can dynamic network diversity impose on the attacker?
- RQ3: To what extent can dynamic network diversity increase the defender's vulnerability tolerance?

- RQ4: To what extent can dynamic network diversity increase the defender's average operational cost?
- RQ5: Is it true that the more diversified implementations the better?

### 4.3.1 Simulation Setting and Methodology

The preceding framework is meant to be as realistic as we can be. This means that it does not make strong assumptions that would warrant analytical treatment. This explains why we pursue simulation-based empirical study to answer the RQs. We adopt agent-based simulation because agents can conduct activities concurrently, which can mimic real-world attack-defense interaction better than sequential simulation. We implement the agent-based simulation via multithreading, where each active local agent, attack and defense alike, is instantiated as one thread such that multiple events can take place concurrently. The simulation experiment is conducted on a computer with 32-CPU and 128GB RAM in the Python environment. In order to accommodate the randomness in the simulation experiment, we conduct 500 simulation runs for each experiment and take their average as the result. In the discrete-time models, all events occur at discrete points in time. In our experiment, we make every attack or defense activity take effect instantly, which can be extended to accommodate any delays if desired.

#### Simulating an Enterprise Network System

Figure 4.5 illustrates our agent-based simulation of attack-defense experiments. We use a master-agent architecture, where two master agents are responsible for scheduling attacks and defenses, respectively. Each simulated host runs a local defense agent by default, which receives instruction from the defense master server. Once a computer is compromised, a local attack agent is instantiated on the compromised computer to receive instructions from the attack master server. When the employment of dynamic network diversity is conducted *manually*, the defense agent can be compromised and re-installed from a clean version when the computer is compromised; when the re-employment process is *automatic*, the defense agent must not be compromised (even if the computer is compromised) so as to assure the re-employment of diversified program implementations. The master attack server updates its knowledge $\Sigma_{\mathcal{A},t}$ based on the information received from the attack agents and makes decisions correspondingly. The master defense server updates its knowledge $\Sigma_{\mathcal{D},t}$ in a similar fashion.

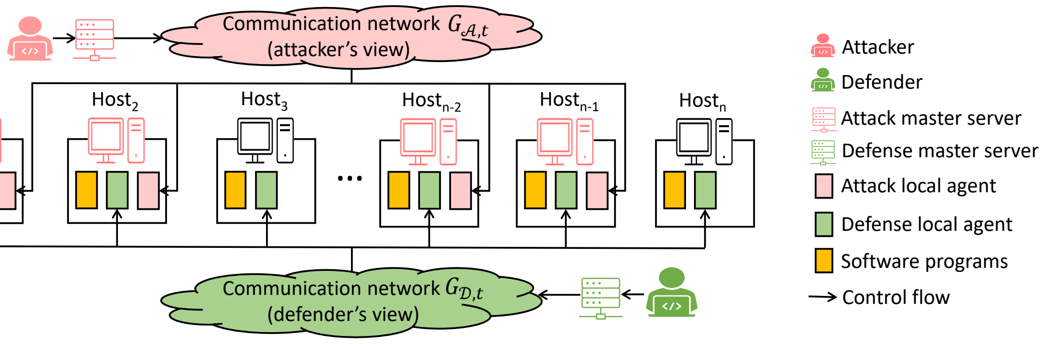


**Figure 4.5**: Illustration of our agent-based simulation of attack-defense experiments.

**Simulating computers' software stack and communications**. In order to make the communication graph $G = (V, E)$ as realistic as possible, we adopt two real-world social networks in Twitter and Friendfeed (which is a social media aggregator) [92], where the former has 5,702 users and the latter has 5,540 users. Together, there are 6,325 users because many users use both Twitter and Friendfeed. We construct the communication graph as follows: a Twitter user corresponds to a Twitter client program and a Friendfeed user corresponds to a Friendfeed client program. A user of Twitter and Friendfeed runs both client programs (i.e., the user's computer is represented as three nodes in the communication graph: one OS and two applications). An edge between two users in the social network means that they communicate with each other using the social network client program. Therefore, the communication graph accommodates the relationships in both social networks.

**Simulating network diversity configuration**. Since there are many ways to configure network diversity and the notion of an *optimal* algorithm is elusive (e.g., the algorithms for generating configuration $C_t$ for $t > 0$ can be different from the one for generating configuration $C_0$), we will consider multiple algorithms and empirically contrast them. In contrast, monoculture software stack is trivial to configure because each program has exactly one implementation.

## Simulating Software Vulnerabilities

In order to simulate vulnerabilities contained in the diversified program implementations, namely $\Phi_t$, we assume that each diversified program has the same diversity quality, namely $Q_1 = Q_2 = \ldots = Q_\hbar = Q$, and each implementation of a program is equally vulnerable with a certain probability; this is a somewhat simplifying assumption but is reasonable in the sense that the same vulnerability prevention and detection techniques may be equally applicable to all implementations. Since there are studies showing that different implementations often do not have the same vulnerability [47,54], we assume that the vulnerabilities are distinct in the sense that each requiring a different exploit.

## Simulating Attacks

**Simulating attacker's knowledge**. For describing attacker's knowledge at time $t = 0$, we assume that the attacker already compromised some vulnerable programs running at some nodes $v \in V$, namely the initial compromise denoted by $\mathsf{IniComp}$. This is reasonable because initial compromise typically follows reconnaissance or is waged by an insider threat, which is orthogonal to the main purpose of the present study. As attack proceeds, the attacker can increase its knowledge by learning more information about the communication graph $G = (V, E)$, and the programs running at the other nodes $v \in V$ that may not be known to the attacker at time $t = 0$. We assume the attacker knows the vulnerabilities associated with the diversified programs.

**Simulating attacker's goal**. For describing attacker's goal $\Omega_{\mathcal{A},t}$, we assume the attacker wants to cumulatively compromise as many programs as possible till time $t = T$. That means $\omega_{v,\mathcal{A},T} = 1$ for $v \in V_{\mathcal{A},t}$, where $t \in [0, T]$.

**Simulating attacker's strategy**. The framework aims to accommodate many attack strategies. In order to make our simulation experiment concrete, we adopt MITRE's ATT&CK [**?**] because it is widely used. The attack *tactics* in ATT&CK can be naturally mapped to the attack phases in our framework. In terms of attacker's strategy $\Gamma_\mathcal{A}$, we focus on the following phases: (i) *installation*, which corresponds to $\gamma_{\mathcal{A},1}$ in the framework and installing attack agents in compromised computers (this phase is called *command-and-control* in ATT&CK); (ii) *discovery*, which corresponds to $\gamma_{\mathcal{A},2}$ in the framework and allows the attacker to concurrently explore the other programs running at the nodes in the other computers of the network; (iii) *privilege escalation*, which corresponds to $\gamma_{\mathcal{A},3}$ in the framework and occurs when the attacker gains the root privilege in the compromised computer; (iv) *lateral movement*, which corresponds to $\gamma_{\mathcal{A},4}$ in the framework and allows the attacker or its malware to exploit the vulnerabilities in the a remote software stack; and (v) *causing damages*, which corresponds to $\gamma_{\mathcal{A},5}$ and allows the attacker to causes damages to a compromised computer (this phase accommodates the tactics of *collection*, *exfiltration*, and *impact* in ATT&CK).

**Simulating attacker's capability**. In order to describe attacker's capability $\Delta_{\mathcal{A}}$, we adopt ATT&CK attack *procedures* (i.e., attacks) as exploits. We assume the attacker possesses the following exploits: (i) an exploit for achieving *remote access*, denoted by $\psi_{\mathcal{A},1,1}$; (ii) an exploit for achieving *remote system discovery* or obtaining information about the software running in other computers, denoted by $\psi_{\mathcal{A},2,1}$; (iii) an exploit for discovering *system information* or getting detailed information about a compromised computer, denoted by $\psi_{\mathcal{A},2,2}$; (iv) a set of exploits for escalating privilege or compromising a vulnerable OS from a compromised application running on top of it, denoted by $\{\psi_{\mathcal{A},3,1},\ldots,\psi_{\mathcal{A},3,m_3}\}$; (iv) a set of exploits for *remote exploitation* capability or compromising a remote, vulnerable computer for lateral movement purposes, denoted by $\{\psi_{\mathcal{A},4,1},\ldots,\psi_{\mathcal{A},4,m_4}\}$; and (v) an exploit causes some damages (e.g., collecting sensitive data from the compromised software), denoted by $\psi_{\mathcal{A},5,1}$. For our purposes, it suffices to assume $cost_{\mathcal{A},z,q} = 1$ for $1 \leq z \leq \ell$, $1 \leq q \leq m_z$, meaning that each exploit incurs the same cost to the attacker (e.g., purchasing or developing an exploit). Future studies can extend this basic scenario to actual cost that may be incurred to the attacker.

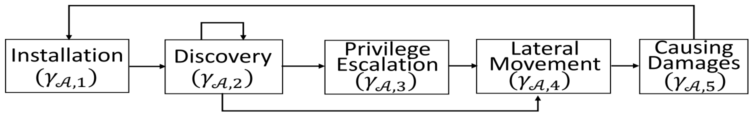

Installation ($\gamma_{\mathcal{A},1}$)
Discovery ($\gamma_{\mathcal{A},2}$)
Privilege Escalation ($\gamma_{\mathcal{A},3}$)
Lateral Movement ($\gamma_{\mathcal{A},4}$)
Causing Damages ($\gamma_{\mathcal{A},5}$)

**Figure 4.6**: Attacker $\mathcal{A}$'s 5-phase strategy (i.e., $\gamma_{\mathcal{A},1},\ldots,\gamma_{\mathcal{A},5}$) and possible decisions (i.e., the arrows), which are adopted from ATT&CK's attack simulator called CALDERA [9].

**Simulating attacker's decision-making**. In order to simulate the attacker's decision-making function $\mathcal{F}_{\mathcal{A}}$, we adopt the decision-making component of ATT&CK that is used by ATT&CK's subsystem known as CALDERA [9]. Figure 4.6 shows how attacks proceed according to the 5-phase strategy mentioned above: install remote access tools at compromised computers; discover local and remote targets; compromise vulnerable OS via privilege escalation (if applicable); compromise remote computers for lateral movement (if applicable); conduct malicious activities; and repeat these processes. Basically, the decision-making algorithm outputs the next exploit that is to be executed, as follows: (i) if the next phase is $\gamma_{\mathcal{A},1}$ or $\gamma_{\mathcal{A},5}$, the attacker will use exploit $\psi_{\mathcal{A},1,1}$ or $\psi_{\mathcal{A},5,1}$ because it only possesses one exploit at each phase; (ii) if the next phase is $\gamma_{\mathcal{A},2}$, the attacker will use exploits $\psi_{\mathcal{A},2,1}$ and $\psi_{\mathcal{A},2,2}$ simultaneously, where $\psi_{\mathcal{A},2,1}$ targets one or more remote computers and $\psi_{\mathcal{A},2,2}$ targets the compromised local computer; (iii) if the next phase is $\gamma_{\mathcal{A},3}$, the attacker will select one exploit from the set $\{\psi_{\mathcal{A},3,1}, \ldots, \psi_{\mathcal{A},3,m_3}\}$ according to the vulnerability information discovered in phase $\gamma_{\mathcal{A},2}$; (iv) if the next phase is $\gamma_{\mathcal{A},4}$, the attacker will select one exploit from the set $\{\psi_{\mathcal{A},4,1}, \ldots, \psi_{\mathcal{A},4,m_4}\}$ according to the vulnerability information discovered in phase $\gamma_{\mathcal{A},2}$.

**Simulating Defenses**

**Simulating defender's knowledge**. As described in the framework, the defender naturally knows, as a part of its knowledge $\Sigma_{\mathcal{D},t}$, the communication graph $G$ (i.e., $G_{\mathcal{D}}$ = $G$) and the network configuration $C_t$ (i.e., $C_{\mathcal{D},t} = C_t$) for any past and present time $t$. At time $t = 0$, we set that the defender's perception of the cybersecurity state $S_0$ as $S_{\mathcal{D},0} = (0, 0, \ldots, 0)_{|V|}$ because the attack-detection tool (if applicable) may start to run at $t = 0$. The defender does not know the information about the vulnerabilities because we allow zero-day vulnerabilities, meaning that $\hat{G}_{\mathcal{D},0} = \emptyset$. Putting these together, the defender's initial knowledge is $\Sigma_{\mathcal{D},0} = (G_{\mathcal{D}}, C_{\mathcal{D},0} S_{\mathcal{D},0}, \hat{G}_{\mathcal{D},0})$.

**Simulating defender's goal**. For describing a defender's goal $\Omega_{\mathcal{D},t}$, we assume the defender aims to keep the compromise rate at any time $t \leq T$ under a certain threshold $\tau_{\mathcal{D}}$. That is, we have $\omega_{i,\mathcal{D},t} = \tau_{\mathcal{D}}$ for $i \in [1, n]$ and $t \in [0, T]$ on average, where the average is over the 500 simulation runs of each experiment.

**Simulating defender's strategy**. We consider the following 5 defense strategies $\Gamma_{\mathcal{D}}$, (i) *monoculture* software stacks, denoted by $\gamma_{\mathcal{D},1}$, which corresponds to $\eta_{\mathcal{D},1} = \texttt{NULL}$, $\eta_{\mathcal{D},2} = \texttt{NULL}$, $\eta_{\mathcal{D},3} = \texttt{NULL}$. (ii) *Static diversity*, denoted by $\gamma_{\mathcal{D},2}$, which corresponds to $\eta_{\mathcal{D},1} = V$, $\eta_{\mathcal{D},2} = \texttt{NULL}$, $\eta_{\mathcal{D},3} = \texttt{NULL}$. (iii) *Proactive diversity*, denoted by $\gamma_{\mathcal{D},3}$, which corresponds to $\eta_{\mathcal{D},1} \neq \texttt{NULL}$, $\eta_{\mathcal{D},2} \neq \texttt{NULL}$, $\eta_{\mathcal{D},3} = \texttt{NULL}$. (iv) *Reactive-adaptive diversity*, denoted by $\gamma_{\mathcal{D},4}$, which corresponds to $\eta_{\mathcal{D},2} = \texttt{NULL}$, $\eta_{\mathcal{D},3} \neq \texttt{NULL}$, because dynamic network diversification is triggered by some security events, which may come from an intrusion detection system or the observed network-wide cybersecurity state $S_{\mathcal{D},t}$. Since such reactive intelligence is often noisy in practice, we incorporate false-negative rate (FNR) and false-positive rate (FPR) into such intelligence. When such intelligence is provided by an employed attack detection system, we assume that the attack detection system cannot be compromised in the present study. (iv) *Hybrid diversity*, denoted by $\gamma_{\mathcal{D},5}$, which combines the aforementioned proactive diversity and reactive-adaptive diversity and corresponds to $\eta_{\mathcal{D},2} \neq \texttt{NULL}$, $\eta_{\mathcal{D},3} \neq \texttt{NULL}$. In this case, dynamic network diversity is triggered periodically or by security events.

**Simulating defender's capability**. For simplicity, we assume that defender's capability $\Delta_{\mathcal{D}}$ includes: (i) the same number of diversified implementations for each program (for simplicity), namely $X_1 = X_2 = \ldots = X_{\hbar} = X$; and (ii) $cost_{\mathcal{D},k,w} = 1$ for $1 \leq k \leq \hbar$ and $1 \leq w \leq X_k$, meaning that each diversified implementation incurs the same cost to the defender. These simplifying assumptions, while arguably reasonable when diversification is an automated process, need to be extended to consider more general cases. It is worth mentioning that different versions of a program should not be counted as diversified implementations because they would have many vulnerabilities in common.

**Simulating defender's decision-making**. For describing defender's decision-making function $\mathcal{F}_{\mathcal{D},t}$, we first consider $\mathcal{F}_{\mathcal{D},0}$ at time $t$ = 0, which outputs the initial network diversity configuration $C_0$. We consider three decision-making algorithms for $\mathcal{F}_{\mathcal{D},0}$ at time $t = 0$: (i) the baseline *random coloring* algorithm, which assigns a random implementation of the program in question to run at node $v$; (ii) the *color flipping* algorithm [104], which uses the random coloring algorithm mentioned above as a starting point and then iteratively let nodes change their colors to reduce the number of *defective* edges (i.e., the edges with two end nodes having the same color or running the same implementation of a software); (iii) a new algorithm we propose, which leverages the degrees of the nodes to assign software implementations to nodes $v \in V$, by giving the large-degree nodes a high priority in running diversified programs. Our algorithm is different from the *color flipping* algorithm mentioned above because we prioritize large-degree nodes in the initial assignment.

For describing defender's decision-making function $\mathcal{F}_{\mathcal{D},t}$ for $t > 0$, we consider $\mathcal{F}_{\mathcal{D},1} = \ldots = \mathcal{F}_{\mathcal{D},T} = \texttt{NULL}$ for defense strategy $\gamma_{\mathcal{D},2}$ because the software deployment stays unchanged over time in the case of static network diversity. For dynamic diversity $\gamma_{\mathcal{D},q}$ where $3 \leq q \leq 5$, we consider a simple random decision-making function $\mathcal{F}_{\mathcal{D},t}$ for $t \in [1, T]$, namely that the defender randomly selects diversified implementations to replace the currently-employed implementations at some or all of the nodes $v \in V$ according to defender's strategy. More sophisticated decision-making functions are left for future studies.

#### Simulating Local Effect and Quantifying Global Effect of Dynamic Network Diversity

We simulate the local effect of dynamic network diversity at each node $v \in V$ as described in Figure 4.3 and the global effect according to the COODAL based attack-defense interactions described in Figure 4.2. We collect the network-wide cybersecurity state and situation over time $t$ to quantitatively answer the RQs.

### 4.3.2 Simulation Results and Analysis

Our simulation study is centered at measuring the metrics to quantify the security effectiveness of employing network diversity, and then leveraging these quantitative effectiveness to draw insights. While the model is general, the simulation study can only consider some specific parameter settings because it is not feasible to consider all parameter settings. That is, the simulation study only corresponds to some scenarios of the general model, and the findings drawn from the simulation study may not be generalized to other parameter settings. Researchers and practitioners can apply

**Algorithm 1** Diversified software stack assignment

**Input:** $G = (V, E)$ with $n$ computers, $\mathsf{SW}$, $X$

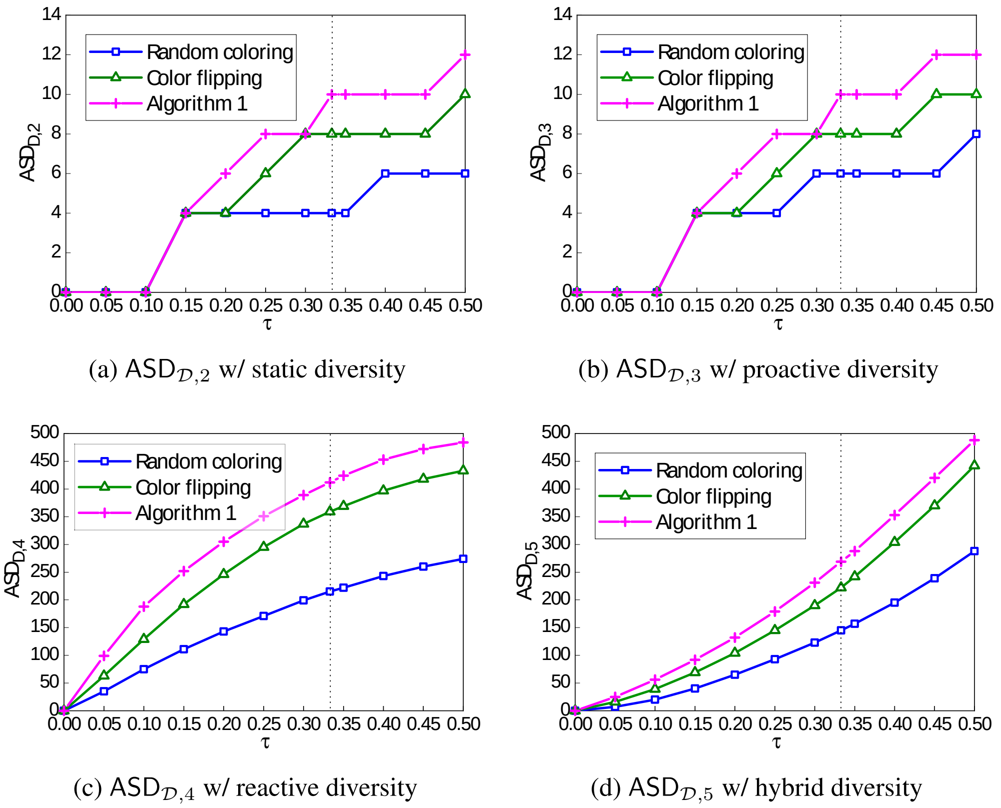
**Output:** $C_0 : V \to \mathsf{SW}$

```
 1: $Y \leftarrow$ number of different types of applications in $G$
 2: $V_{\mathsf{app}_j} \leftarrow \{\mathsf{app}_{1,j}, \ldots, \mathsf{app}_{n,j}\}$ where $1 \leq j \leq Y$
 3: $V_{\mathsf{os}} \leftarrow \{\mathsf{os}_1, \ldots, \mathsf{os}_n\}$
 4: for $1 \leq j \leq Y$ do
 5:     ORDERING($V_{\mathsf{app}_j}$)
 6:     COLORING($V_{\mathsf{app}_j}$)
 7:     SWITCHING($V_{\mathsf{app}_j}$)
 8: end for
 9: ORDERING($V_{\mathsf{os}}$)
10: COLORING($V_{\mathsf{os}}$)
11: SWITCHING($V_{\mathsf{os}}$)
12: function ORDERING($V^{'}$)
13:     Sort $V^{'}$ based on their degree in the descending order
14: end function
15: function COLORING($V^{'}$)
16:     Label the implementations from 1 to $X$
17:     for $v^{'} \in V^{'}$ do
18:         fetch an implementation from SW in turn and pre-assign it to node $v^{'}$
19:         if no defective edges linked to $v^{'}$ then
20:             approve the pre-assignment
21:         else
22:             find out some other implementation that does not lead to local defective edges and
                assign it $v^{'}$
23:             if no applicable implementation then
24:                 choose the implementation that leads to minimum local defective edges, if more
                    than one candidate exists, choose the implementation that is the same as that of the
                    adjacent node with the lowest degree
25:             end if
26:         end if
27:     end for
28: end function
29: function SWITCHING($V^{'}$)
30:     switch the implementation of $v^{'} \in V^{'}$ to another iteratively if that can lead to less defective
        edges
31: end function
```

our model to their specific parameter settings.

**RQ1: To what extent can dynamic network diversity slow down the attacker?**

In order to answer RQ1, we investigate how the attacker slower-down metric $\mathsf{ASD}_{\mathcal{D},q}$ depends on defender's goal $\tau$ with $2 \leq q \leq 5$. The experimental parameters are: $\hbar$ = 3 (3 different programs running in the network: Twitter, Friendfeed, OS), $X$ = 10 (each program has 10 diversified implementations), $Q$ = 1 (every diversified implementation is vulnerable), $|\mathsf{IniComp}|$ = 10 (10 programs/nodes are initially compromised), $m_3$ = 5 (the attacker has 5 exploits against OS), $m_4$ = 10 (the attacker has 5 exploits against Twitter and 5 exploits against Friendfeed), and $T$ = 500 (the simulation stops at $t$ = 500).



(a) $\mathsf{ASD}_{\mathcal{D},2}$ w/ static diversity (b) $\mathsf{ASD}_{\mathcal{D},3}$ w/ proactive diversity

(c) $\mathsf{ASD}_{\mathcal{D},4}$ w/ reactive diversity (d) $\mathsf{ASD}_{\mathcal{D},5}$ w/ hybrid diversity

**Figure 4.7**: Plots of attacker slow-down $\mathsf{ASD}_{\mathcal{D},q}$ under different diversity strategies (where reactive is short for reactive-adaptive) with varying tolerable compromise threshold $\tau$, where $2 \leq q \leq 5$ and dotted vertical lines indicate $\tau$ = 1/3.

Figure 4.7(a) plots attacker slow-down $\mathsf{ASD}_{\mathcal{D},2}$ with respect to defender's goal $\tau \in [0, 0.5]$, where the defender uses static diversity and different decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$ at time $t = 0$. We observe: (i) $\mathsf{ASD}_{\mathcal{D},2}$ = 0 when $\tau \leq 0.1$, meaning that static diversity cannot slow down the attacker when the defender can only tolerate no more than 10% of the nodes being compromised. (ii) $\mathsf{ASD}_{\mathcal{D},2} > 0$ when $\tau \geq 0.15$, meaning that static diversity can slow down the attacker at an extent that increases with the degree of tolerable compromise. This is reasonable because the attacker has to do more lateral movements in order to disrupt the defender's goal. (iii) For a fixed tolerable compromise threshold $\tau$, the (initial diversity) decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$ matters and our algorithm slows down the attacker most, with an average slow-down that is almost 2X of that of the random coloring algorithm, where the average is over the $\tau$'s.

Figure 4.7(b) plots attacker slow-down $\mathsf{ASD}_{\mathcal{D},3}$ with respect to defender's goal $\tau \in [0, 0.5]$, where the defender uses different decision-making algorithms $\mathcal{F}_{\mathcal{D},0}$ for initial diversity and proactively uses $\mathcal{F}_{\mathcal{D},t}$ for $t > 0$ with parameters $\eta_{\mathcal{D},1}$ = 0.5 and $\eta_{\mathcal{D},2}$ = 0.2 (diversified implementations are re-deployed at 50% of the nodes every 5 time steps). We make the same observations as that of $\mathsf{ASD}_{\mathcal{D},2}$. This is reasonable because proactive diversity can replace compromised programs with secure programs (i.e., benefiting the defender), but can also replace secure programs with vulnerable ones (i.e., benefiting the attacker).

Figure 4.7(c) plots attacker slow-down $\mathsf{ASD}_{\mathcal{D},4}$ with respect to defender's goal $\tau \in [0, 0.5]$, where the defender uses some decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$ for initial diversity and reactive-adaptively uses $\mathcal{F}_{\mathcal{D},t}$ for $t > 0$ while assuming parameters $\mathsf{FPR}$ = 0.1 and $\mathsf{FNR}$ = 0.1 (i.e., the attack-detection or threat intelligence has a 10% false-positive rate and a 10% false-negative rate). We make the following observations: (i) $\mathsf{ASD}_{\mathcal{D},4} > 0$ for $\tau \in (0, 0.5]$, meaning that reactive-adaptive diversity can always slow down the attacker at an extent that concavely increases with the tolerable compromise threshold $\tau$. This is reasonable because the defender can replace a likely-compromised software with another diversified implementation to benefit the defender. (ii) For a fixed tolerable compromise threshold $\tau$, the initial diversity algorithm $\mathcal{F}_{\mathcal{D},0}$ matters because our algorithm slows down the attacker most, with an average of almost 2X slow-down than that of the the random coloring algorithm.

Figure 4.7(d) plots attacker slow-down $\mathsf{ASD}_{\mathcal{D},5}$ with respect to defender's goal $\tau \in [0, 0.5]$, where the defender uses some decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$ for initial diversity and hybrid (of proactive and reactive-adaptive) decision-making algorithm $\mathcal{F}_{\mathcal{D},t}$ for $t > 0$ with parameters $\eta_{\mathcal{D},2}$ = 0.2, $\mathsf{FPR}$ = 0.1 and $\mathsf{FNR}$ = 0.1. We observe the same phenomena as in the case of reactive-adaptive diversity, except that the degree of slow-down incurred by hybrid diversity increases with the tolerable compromise threshold in a *convex* (rather than *concave*) fashion.

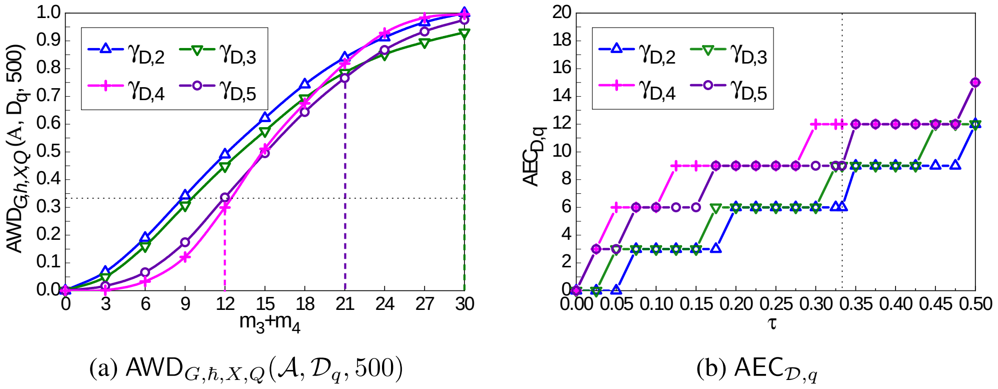
By comparing Figures 4.7(a)-4.7(d), we observe that for a fixed (initial diversity) decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$, we have $\mathsf{ASD}_{\mathcal{D},4} > \mathsf{ASD}_{\mathcal{D},5} \gg \mathsf{ASD}_{\mathcal{D},3} \approx \mathsf{ASD}_{\mathcal{D},2}$ for $\tau \in (0, 0.45]$, indicating that reactive-adaptive diversity outperforms hybrid diversity, which significantly outperforms proactive diversity and static diversity. Consider $\tau = 1/3$ as an example, we observe $\mathsf{ASD}_{\mathcal{D},4}$ = 412, $\mathsf{ASD}_{\mathcal{D},5}$ = 269, $\mathsf{ASD}_{\mathcal{D},3}$ = 10, and $\mathsf{ASD}_{\mathcal{D},2}$ = 10 when using our algorithm as (initial diversity) decision-making algorithm $\mathcal{F}_{\mathcal{D},0}$. Note that the curves in Fig. 4.7(a) are not smooth because, under the static diversity strategy, compromised programs are not cleaned up and can attack others. This also explains why the curves in Fig. 4.7(b) are not smooth, namely that proactive diversity always periodically selects random programs for re-employing dynamic diversity, causing some compromised computers to remain compromised for extended durations and allowing them to conduct further attacks. In contrast, the curves in Fig. 4.7(c) and Fig. 4.7(d) are smooth because these two strategies leverage reactive defense systems to identify and replace the compromised programs timely, which can limit the abrupt spreading of attacks and slow down the attacker. One reason for the reactive-adaptive strategy to perform better than the hybrid strategy is that the former can immediately clean up the compromised programs, but the latter waits until a period of time, which allows the attacker to compromise other vulnerable programs. That is, the difference is caused by whether there is a gap between when compromised programs are detected and when compromised programs are cleaned up.

**Insight 5.** *In terms of the attacker slow-down metric, reactive-adaptive diversity is the most effective strategy and the initial diversity configuration matters.*

**RQ2: How much extra cost can dynamic network diversity impose on the attacker?**

In order to answer RQ2, we investigate how the *attack extra cost* metric $\mathsf{AEC}_{\mathcal{D},q}$ increases with defender's goal $\tau$, where $2 \leq q \leq 5$. For this purpose, we need to see how attack cost affects the network-wide cybersecurity state, especially attack worst damage $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,T)$, where $2 \leq q \leq 5$. The simulation experiment parameters are: $\hbar = 3$ (i.e., 3 different programs running in the network: Twitter, Friendfeed, and OS), $X = 10$ (i.e., each program has 10 diversified implementations), $Q = 1$ (i.e., every diversified implementation is vulnerable), $|\mathsf{IniComp}| = 10$ (i.e., 10 programs or nodes are initially compromised), $\mathcal{F}_{\mathcal{D},0}$ is our algorithm for employing initial diversity, $\eta_{\mathcal{D},1} = 0.5$ (i.e., diversified implementations are dynamically re-employed at 50% of all nodes), $\eta_{\mathcal{D},2} = 0.2$ (i.e., diversified implementations are re-employed every 5 time steps), $\mathsf{FPR} = 0.1$ (i.e., 10% false-positive rate in detecting attacks), $\mathsf{FNR} = 0.1$ (i.e., 10% false-negative rate in attack detection), $T = 500$. We assume that the attacker uses the available exploits together, for the sake of reducing the uncertainty in the outcomes that may be incurred by the orders of exploits usage.

Figure 4.8(a) plots $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,500)$ for $2 \leq q \leq 5$, with respect to the total number $m_3 + m_4$ of exploits possessed by the attacker, where $m_3$ is the number of exploits that provide the attacker with privilege escalation capability and $m_4$ is the number of exploits that provide the attacker with lateral movement capability. We observe the following *phase-transition* phenomenon. When $0 < m_3 + m_4 \leq 12$ (i.e., the attacker possessing no more than 40% of the total 30 exploits, which correspond to all of the 30 vulnerabilities in the diversified implementations), the reactive-adaptive diversity strategy $\gamma_{\mathcal{D},4}$ leads to the lowest attack worst damage (i.e., the $\gamma_{\mathcal{D},4}$ curve); when $12 < m_3 + m_4 \leq 21$, the hybrid diversity strategy $\gamma_{\mathcal{D},5}$ leads to the lowest attack worst damage (i.e., the $\gamma_{\mathcal{D},5}$ curve); when $21 < m_3 + m_4 \leq 30$, the proactive diversity strategy $\gamma_{\mathcal{D},3}$ leads to the lowest attack worst damage (i.e., the $\gamma_{\mathcal{D},3}$ curve). This phenomenon can be explained as follows. In terms of the *attack worst damage* metric, reactive-adaptive diversity is the most effective strategy against a less capable attacker because the defender can detect and replace the small number of compromised programs; proactive diversity is the most effective strategy against a more capable attacker, which compromises a large number of computers and demands periodic enforcement of dynamic diversity at most, if not all, of the computers.



(a) $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,500)$ (b) $\mathsf{AEC}_{\mathcal{D},q}$

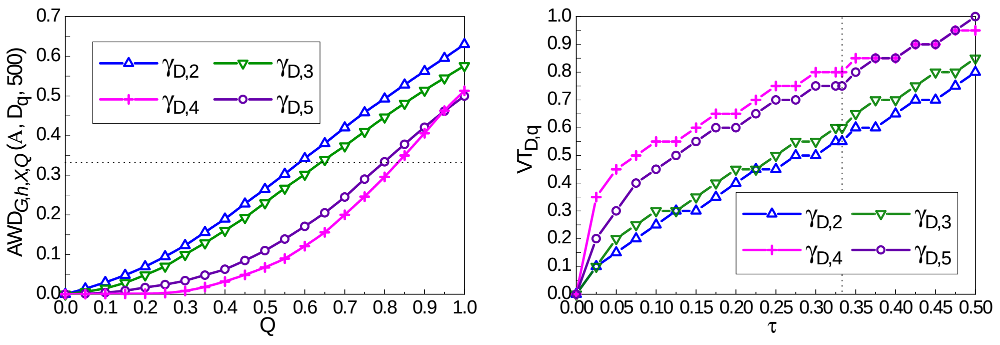
**Figure 4.8**: Plots of $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_q,500)$ and $\mathsf{AEC}_{\mathcal{D},q}$ with $2 \le q \le 5$.

From Figure 4.8(a), we can derive the attack extra cost $\mathsf{AEC}_{\mathcal{D},q}$ with respect to the defender's goal $\tau \in [0,0.5]$ (i.e., no more than a $\tau$ fraction of the nodes are compromised at *any* time $t \in [0,T]$) where $2 \le q \le 5$, which is plotted in Figure 4.8(b). We make the following observations: (i) $\mathsf{AEC}_{\mathcal{D},4} > 0$ and $\mathsf{AEC}_{\mathcal{D},5} > 0$ when $\tau \ge 0.025$, $\mathsf{AEC}_{\mathcal{D},3} > 0$ when $\tau \ge 0.05$, and $\mathsf{AEC}_{\mathcal{D},2} > 0$ when $\tau \ge 0.075$, meaning that employing dynamic network diversity defense strategy can impose attack extra cost on the attacker at an extent that increases with the degree of tolerable compromises. (ii) $\mathsf{AEC}_{\mathcal{D},4} \ge \mathsf{AEC}_{\mathcal{D},5} \ge \mathsf{AEC}_{\mathcal{D},3} \ge \mathsf{AEC}_{\mathcal{D},2}$ always holds for any $\tau \in (0,0.5]$. Consider $\tau$ = 1/3 as an example, we observe $\mathsf{AEC}_{\mathcal{D},4} = 40\% \ge \mathsf{AEC}_{\mathcal{D},5} = 30\% \ge \mathsf{AEC}_{\mathcal{D},3} = 30\% \ge \mathsf{AEC}_{\mathcal{D},2} = 20\%$, meaning that reactive-adaptive diversity imposes extra cost on the attacker more than hybrid diversity, which incurs more than proactive diversity and even more than static diversity.

**Insight 6.** *In order to reduce the attack worst damage, different diversity strategies should be used in different parameter regimes.*

**RQ3: To what extent can dynamic network diversity increase the defender's vulnerability tolerance?**

In order to answer RQ3, we investigate how the *vulnerability tolerance* metric $\mathsf{VT}_{\mathcal{D},q}$ depends on defender's goal $\tau$, where $2 \le q \le 5$. Since this dependence would rely on attack worst damage $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T)$ and the parameters $G$ and $\hbar$ are largely determined by the applications and $X$ is largely determined by what is available, we will investigate the impact of diversification quality $Q$. The simulation experiment parameters are: $\hbar = 3$ (3 different programs running in the network: Twitter, Friendfeed, and OS), $X = 20$ (each program has 20 diversified implementations), $|\mathsf{IniComp}| = 10$ (10 programs or nodes are initially compromised), $\mathcal{F}_{\mathcal{D},0}$ is our algorithm for employing initial diversity, $\overline{m_3} = 0.5 \times X \times Q$ (the attacker has 50% of the total $X \times Q$ exploits against the OSes on average), $\overline{m_4} = 0.5 \times 2X \times Q$ (the attacker has 50% of the total $X \times Q$ exploits against Twitter and 50% of the total $X \times Q$ exploits against Friendfeed on average), $\eta_{\mathcal{D},1} = 0.5$ (diversified implementations are dynamically re-employed at 50% of all nodes), $\eta_{\mathcal{D},2} = 0.2$ (diversified implementations are re-employed every 5 time steps), $\mathsf{FPR} = 0.1$ (10% false-positive rate in detecting attacks), $\mathsf{FNR} = 0.1$ (10% false-negative rate in attack detection), and mission lifetime $T = 500$.



(a) $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, 500)$ (b) $\mathsf{VT}_{\mathcal{D},q}$

**Figure 4.9**: Plots of $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, 500)$ and $\mathsf{VT}_{\mathcal{D},q}$ with $2 \le q \le 5$.

Figure 4.9(a) plots $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, 500)$ with respect to diversity quality $Q$ when the defender employs network diversity defense strategy $\gamma_{\mathcal{D},q}$, where $2 \le q \le 5$. We observe that $\mathsf{cc}_{\mathcal{D},4} < \mathsf{cc}_{\mathcal{D},5} < \mathsf{cc}_{\mathcal{D},3} < \mathsf{cc}_{\mathcal{D},2}$ when $0 < Q \le 0.95$, and $\mathsf{cc}_{\mathcal{D},5} < \mathsf{cc}_{\mathcal{D},4} < \mathsf{cc}_{\mathcal{D},3} < \mathsf{cc}_{\mathcal{D},2}$ when $0.95 \le Q \le 1$. This means that employing dynamic network diversity always leads to higher security than static diversity regardless of the diversification quality $Q$. We observe that the attack worst damage $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, 500)$ increases when the quality of diversified implementations drops, where $2 \le q \le 5$. This means that dynamic network diversity leads to an even higher security when the diversity quality is high (i.e., low $Q$'s).

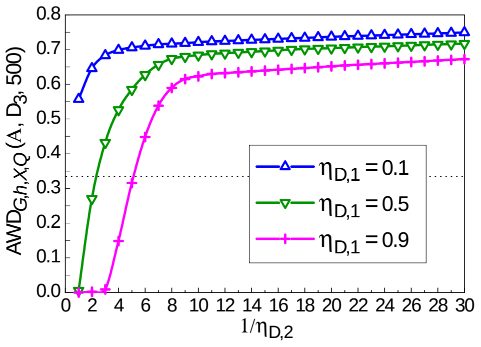
From Figure 4.9(a), we can derive the vulnerability tolerance $\mathsf{VT}_{\mathcal{D},q}$ with respect to the defender's goal $\tau \in [0, 0.5]$ where $2 \leq q \leq 5$, which is plotted in Figure 4.9(b). We observe that $\mathsf{VT}_{\mathcal{D},4} \geq \mathsf{VT}_{\mathcal{D},5} > \mathsf{VT}_{\mathcal{D},3} \geq \mathsf{VT}_{\mathcal{D},2}$ when $0 < \tau \leq 0.475$. Consider $\tau$ =1/3 as an example, we observe $\mathsf{VT}_{\mathcal{D},4}$ = 0.8, $\mathsf{VT}_{\mathcal{D},5}$ = 0.75, $\mathsf{VT}_{\mathcal{D},3}$ = 0.6, and $\mathsf{VT}_{\mathcal{D},2}$ = 0.55. This means that given an attacker that can exploit 50% of the vulnerabilities in the network on average, reactive-adaptive diversity strategy $\gamma_{\mathcal{D},4}$ makes the defender tolerate a $0.8 - 2/3 = 0.14$ or 14% extra vulnerabilities, and hybrid diversity strategy $\gamma_{\mathcal{D},5}$ make the defender tolerate a $0.75 - 2/3 = 0.09$ or 9% extra vulnerabilities; however, proactive and static diversity strategies cannot achieve this effectiveness.

**Insight 7.** *Reactive-adaptive diversity leads to a higher vulnerability-tolerance than proactive diversity does.*

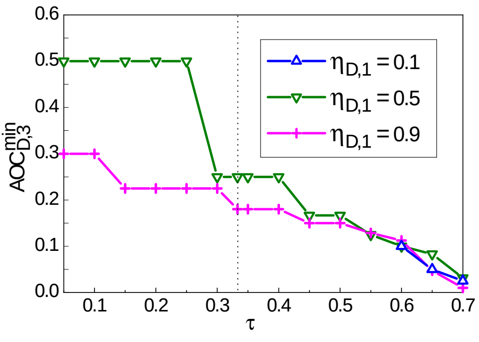
**RQ4: To what extent can dynamic network diversity increase the defender's average operational cost?**

In order to answer RQ4, we investigate how the *average operational cost* metric $\mathsf{AOC}_{\mathcal{D},q}$ increases with defender's goal $\tau$, where $2 \leq q \leq 5$. For this purpose, we need to know how attack worst damage $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_q, T)$ depends on defense diversity strategies $\gamma_{\mathcal{D},q}$, where $2 \leq q \leq 5$. The simulation experiment parameters are: $\hbar$ = 3 (3 different programs running in the network: Twitter, Friendfeed, OS), $X$ = 10 (each program has 10 diversified implementations), $Q$ = 1 (every diversified implementation is vulnerable), $|\mathsf{IniComp}|$ = 10 (10 programs/nodes are initially compromised), $\mathcal{F}_{\mathcal{D},0}$ is our algorithm for employing initial diversity, $m_3$ = 5 (the attacker has 5 exploits against OS), $m_4$ = 10 (the attacker has 5 exploits against Twitter and 5 exploits against Friendfeed), and mission lifetime $T = 500$.

Figure 4.10(a) plots $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_3, 500)$ with respect to $\eta_{\mathcal{D},1}$ and $1/\eta_{\mathcal{D},2}$, where $\eta_{\mathcal{D},1}$ is the the proportion of nodes where diversified programs are dynamically employed and $1/\eta_{\mathcal{D},2}$ is the time interval between two consecutive diversity employments. We observe the following: (i) When $\eta_{\mathcal{D},1}$ = 0.1, proactive diversity always leads to high attack worst damage, even if $1/\eta_{\mathcal{D},2}$ is



(a) $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_3, 500)$



(b) $\mathsf{AOC}^{\min}_{\mathcal{D},3}$

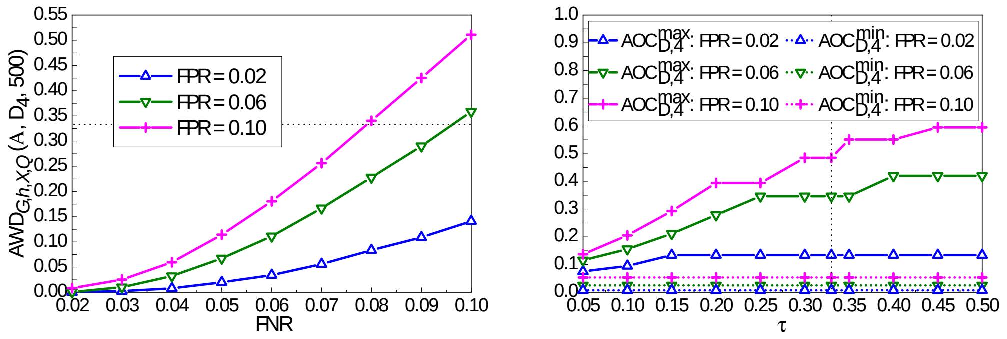
**Figure 4.10**: Plots of $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A}, \mathcal{D}_3, 500)$ and $\mathsf{AOC}^{\min}_{\mathcal{D},3}$.

small (i.e., high-frequency in employment). This means that proactive diversity is useless when dynamic diversity is employed at a few nodes. (ii) When $\eta_{\mathcal{D},1}$ = 0.5 or 0.9, proactive diversity can lead to low attack worst damage only when $1/\eta_{\mathcal{D},2}$ is small. Consider defense goal $\tau$ =1/3 as an example, $1/\eta_{\mathcal{D},2}$ must be no more than 2 when $\eta_{\mathcal{D},1}$ = 0.5 and no more than 5 when $\eta_{\mathcal{D},1}$ = 0.9. This means that proactive diversity is effective only when employed broadly and frequently. (iii) When $\eta_{\mathcal{D},1}$ = 0.1 and $1/\eta_{\mathcal{D},2} \geq 2$, or when $\eta_{\mathcal{D},1}$ = 0.5 and $1/\eta_{\mathcal{D},2} \geq 6$, or when $\eta_{\mathcal{D},1}$ = 0.9 and $1/\eta_{\mathcal{D},2} \geq 14$, proactive diversity leads to higher attack worst damage than static diversity. This means that proactive diversity can do more harm than good by making making more nodes exploitable over time.

**Insight 8.** *Proactive diversity improves security only when employed at most nodes at high frequency.*

From Figure 4.10(a), we can derive the minimum average operational cost $\mathsf{AOC}^{\min}_{\mathcal{D},3}$ with respect to the defender's goal $\tau \in [0, 0.7]$, which is shown in Figure 4.10(b). Consider $\tau$ = 1/3 as an example, the minimum average operational cost is $\eta_{\mathcal{D},1} \times \eta_{\mathcal{D},2} = 0.5 \times 1/2 = 0.25$ for $\eta_{\mathcal{D},1}$ = 0.5, and $\eta_{\mathcal{D},1} \times \eta_{\mathcal{D},2} = 0.9/5 = 0.18$ for $\eta_{\mathcal{D},1}$ = 0.9. Note that in the case $\eta_{\mathcal{D},1}$ = 0.1, $\mathsf{AOC}^{\min}_{\mathcal{D},3}$ is undefined when $\tau \in [0, 0.55]$ because the strategy can never prevent the attacker from breaking the defender's goal. We further observe that $\eta_{\mathcal{D},1}$ = 0.9 leads to a lower $\mathsf{AOC}^{\min}_{\mathcal{D},3}$ than $\eta_{\mathcal{D},1}$ = 0.5 when $0 \leq \tau \leq 0.5$, meaning that a higher proportion of dynamic diversity re-employment leads to a lower average operation cost.

**Insight 9.** *When proactive diversity is effective, a higher proportion of dynamic re-employment leads to a lower operational cost than what is incurred by a higher re-employment frequency.*

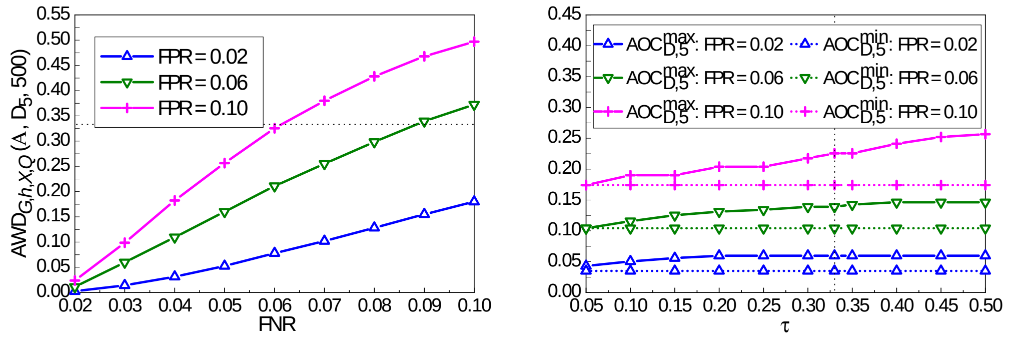


(a) $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_4,500)$ (b) $\mathsf{AOC}_{\mathcal{D},4}$

**Figure 4.11**: Plots of $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_4,500)$ and $\mathsf{AOC}_{\mathcal{D},4}$.

Figure 4.11(a) plots $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_4,500)$ with respect to FPR (false-positive rate) and FNR (false-negative rate), where FPR, FNR $\in [0.02, 0.1]$. We observe that a lower FNR and FPR (i.e., higher attack-detection capability) leads to a lower attack worst damage, meaning the effectiveness of reactive-adaptive diversity largely depends on the attack-detection capability. Consider defender's mission goal of $\tau = 1/3$ as an example, we observe that reactive-adaptive diversity can assure the mission when FPR $= 0.02$ and $0.02 \leq$ FNR $\leq 0.1$, when FPR $= 0.06$ and $0.02 \leq$ FNR $\leq 0.09$, and when FPR $= 0.1$ and $0.02 \leq$ FNR $\leq 0.07$. From Figure 4.11(a), we can derive the $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},4}$ and $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},4}$ for a given FPR with respect to $\tau \in [0, 0.5]$, which is plotted in Figure 4.11(b). We make the following observations. (i) For a fixed FPR, $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},4}$ can be very low and remains stable as $\tau$ increases, because $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},4}$ is always achieved when FNR = 0.02. The operational cost is low because a high attack-detection accuracy can detect compromised computers before they attack the others. (ii) For a fixed FPR, a higher $\tau$ often leads to a higher $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},4}$, because a higher $\tau$ (i.e., higher compromise-tolerance) can be achieved at a lower operational cost from a diversity-based defense standpoint. (iii) The defender's operational cost falls into a wide range as $\tau$ increases, meaning that the defender's operational cost largely depends on the attack-detection effectiveness.

Figure 4.12(a) plots $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_5,500)$ with respect to FPR and FNR, where FPR, FNR $\in$



(a) $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_5,500)$ (b) $\mathsf{AOC}_{\mathcal{D},5}$

**Figure 4.12**: Plots of $\mathsf{AWD}_{G,\hbar,X,Q}(\mathcal{A},\mathcal{D}_5,500)$ and $\mathsf{AOC}_{\mathcal{D},5}$.

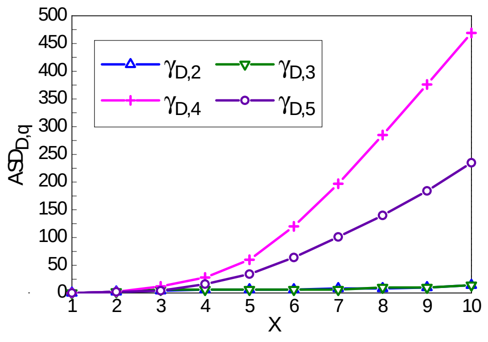
$[0.02, 0.1]$ and re-deployment frequency $\eta_{\mathcal{D},2} = 0.2$. From Figure 4.12(a), we derive $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},5}$ and $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},5}$, which is plotted in Figure 4.12(b). We observe that the defender's average operational cost is small. Consider $\tau$ = 1/3 as an example, we observe $\mathsf{AOC}^{\mathsf{min}}_{\mathcal{D},5} = 0.0349$ when $\mathsf{FPR} = 0.02$ and $\mathsf{AOC}^{\mathsf{max}}_{\mathcal{D},5} = 0.2254$ when FPR = 0.1, meaning $\mathsf{AOC}_{\mathcal{D},5} \in$[0.0349, 0.2254] when FPR $\in [0.02, 0.1]$ and FNR $\in [0.02, 0.1]$.

By comparing Figures 4.10(b), 4.11(b), and 4.12(b), we observe that for $\tau \in [0.05, 0.5]$, we have $\mathsf{AOC}_{\mathcal{D},3} \in [0.15, 0.5]$ when $\eta_1 \in [0.5, 0.9]$; we have $\mathsf{AOC}_{\mathcal{D},4} \in [0.0071, 0.5948]$ and $\mathsf{AOC}_{\mathcal{D},5} \in [0.0349, 0.2566]$ when FPR $\in [0.02, 0.1]$ and FNR $\in [0.02, 0.1]$, meaning that reactive-adaptive diversity incurs an average operational cost falling into a wider range than proactive diversity and an even wider range than hybrid diversity. We also observe that proactive diversity incurs a higher operational cost than reactive-adaptive diversity when the defender's tolerable compromise threshold $\tau$ is small, and the opposite is true when $\tau$ is large.

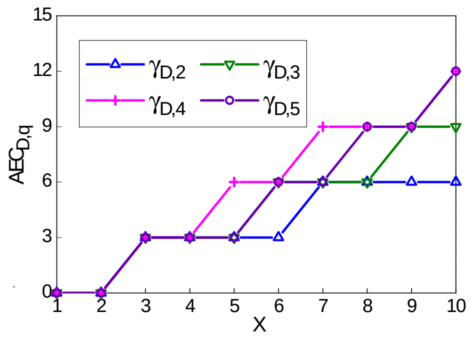
**RQ5: Is it true that the more diversified implementations the better?**

In order to answer RQ5, we investigate how the *attacker slow-down* metric $\mathsf{ASD}_{\mathcal{D},q}$, the *attack extra cost* metric $\mathsf{AEC}_{\mathcal{D},q}$, and the *vulnerability tolerance* metric $\mathsf{VT}_{\mathcal{D},q}$ depend on the number of diversified implementations $X$, where $2 \le q \le 5$. The experimental parameters are: $\hbar = 3$ (3 different programs running in the network: Twitter, Friendfeed, and OS), $Q = 1$ (every diversified implementation is vulnerable), $|\mathsf{IniComp}| = 10$ (10 programs or nodes are initially compromised),

$m_3$ = 1 (the attacker has 1 exploit against OS), $m_4$ = 2 (the attacker has 1 exploit against Twitter and 1 exploit against Friendfeed), $\mathcal{F}_{\mathcal{D},0}$ is our algorithm for employing initial diversity, $\eta_{\mathcal{D},1} = 0.5$ (diversified implementations are dynamically re-employed at 50% of all nodes), $\eta_{\mathcal{D},2} = 0.2$ (diversified implementations are re-employed every 5 time steps), $\mathsf{FPR} = 0.1$ (10% false-positive rate in detecting attacks), $\mathsf{FNR} = 0.1$ (10% false-negative rate in attack detection), and mission lifetime $T = 500$.



(a) $\mathsf{ASD}_{\mathcal{D},q}$



(b) $\mathsf{AEC}_{\mathcal{D},q}$

**Figure 4.13**: Plots of $\mathsf{ASD}_{\mathcal{D},q}$ and $\mathsf{AEC}_{\mathcal{D},q}$, where $2 \leq q \leq 5$.

Figure 4.13(a) plots $\mathsf{ASD}_{\mathcal{D},q}$ with respect to $X$ and $\tau = 0.01$, where $2 \leq q \leq 5$. We take $\tau = 0.01$ as an example because the attacker possessing one exploit for each program may only compromise about 1% computers during the mission lifetime $T = 500$ with $X$ =10. We observe that $\mathsf{ASD}_{\mathcal{D},4}$ increases rapidly with $X$, followed by $\mathsf{ASD}_{\mathcal{D},5}$, while $\mathsf{ASD}_{\mathcal{D},3}$ and $\mathsf{ASD}_{\mathcal{D},2}$ increases slowly with $X$ (e.g., $\mathsf{ASD}_{\mathcal{D},3}$ = $\mathsf{ASD}_{\mathcal{D},2}$ = 2 when $X$ =2 and $\mathsf{ASD}_{\mathcal{D},3}$ = $\mathsf{ASD}_{\mathcal{D},2}$ = 14 when $X$ = 10). When $X$ = 10, reactive-adaptive diversity slows down the attacker most.

Figure 4.13(b) plots $\mathsf{AEC}_{\mathcal{D},q}$ with respect to $X$ and $\tau$ = 1/3, where $2 \leq q \leq 5$. We observe that $\mathsf{AEC}_{\mathcal{D},q}$ shows an upward trend as $X$ increases for $q$ = 2, 3, 4, 5, meaning that the more diversified implementations, the higher the attack extra cost for disrupting the defender's mission goal (i.e., no more than a $\tau$ fraction of computers are compromised at any time $t \in [0, T]$). In addition, we observe $\mathsf{AEC}_{\mathcal{D},4} \geq \mathsf{AEC}_{\mathcal{D},5} \geq \mathsf{AEC}_{\mathcal{D},3} \geq \mathsf{AEC}_{\mathcal{D},2}$ for any $X \in [1, 10]$, meaning that reactive-adaptive diversity benefits most from more diversified implementations, followed by hybrid diversity, proactive diversity and static diversity.

In order to characterize the impact of the number of diversified implementations on the vulnerability tolerance $\mathsf{VT}_{\mathcal{D},q}$ where $2 \leq q \leq 5$, we observe $\mathsf{VT}_{\mathcal{D},q} = 0$ when $X = 1, 2$ and $\tau$ =1/3. This is reasonable because the attacker having one exploit for each program can easily break the defender's goal when the total number of diversified implementations is no more than 2. In contrast, given $\tau$ =1/3, we have $\mathsf{VT}_{\mathcal{D},q}$ = 1 when $X \geq 3$, where $2 \leq q \leq 5$, because Figure 4.13(b) shows that an attacker having one exploit for each program will never compromise 1/3 computers when $X \geq 3$; this is true even if all of the diversified implementations are vulnerable. This means that when the attacker has a limited capability, a substantially higher vulnerability rate (than $\tau$) can effectively be tolerated until the number of diversified implementations reaches a certain amount.

**Insight 10.** *The more diversified implementations with a similar quality, the higher the attacker slow-down, the attack extra cost, and the vulnerability tolerance.*

## 4.4 Chapter Summary

In this Chapter, we instantiated the proposed framework for quantifying the cybersecurity effectiveness of enforcing dynamic network diversity, including a suite of security metrics for measuring attacker's cost (incurred by obtaining exploits) and defender's operational cost (incurred by re-employing network diversity). We conducted simulation experiments to measure these metrics with respect to a number of dynamic diversity strategies and drew insights from the experimental results.

# CHAPTER 5: QUANTIFYING SECURITY EFFECTIVENESS OF FINE-GRAINED STATIC NETWORK DIVERSITY

## 5.1 Chapter Introduction

Software monoculture automatically amplifies the damage of cyber attacks because a single vulnerability in the *software stack* can cause the compromise of all of the computers running the same vulnerable software [51, 125], where *software stack* includes the application, library, and operating system layers. To cope with the problem, researchers have proposed the idea of diversifying the software stack [166]. *Network diversity* means the software stack is diversified in a computer network.

Software diversity can be achieved by two approaches: *natural diversity* and *artificial diversity*. *Natural diversity* often emerges from market competition, as witnessed by the presence of different vendors for the same functionality, such as Windows versus Linux for operating systems, or Chrome versus Firefox versus Internet Explorer for browsers. *Artificial diversity* refers to the different versions of a functionality that are independently implemented, such as *N-version programming* [10]. The concept of artificial diversity was originally introduced to enhance software reliability, but nowadays it has been adopted for achieving security purposes. This intuitive assumption seems reasonable because independent implementations of a functionality are highly unlikely to contain the same vulnerabilities.

The rule of thumb is that network diversity improves security when compared with the monoculture software stack. However, this perception has not been quantitatively validated with a scientific basis, which is necessary for justifying both the cost of artificial software diversity and the effectiveness of network diversity. In this work, we take a first step towards ultimately tackling this problem.

### 5.1.1 Chapter Contributions

This Chapter makes the following contributions.

First, we quantify the security effectiveness of network diversity from a whole-network perspective, namely viewing a network as a whole. We propose a framework for modeling attack-defense interactions in a network based on a graph-theoretic model, in which a *node* represents a software component or function, and an *arc* represents a certain relation between them and has security consequences. The framework is *fine-grained* because it treats individual applications, library functions, and operating system kernel functions as "atomic" entities. This granularity allows us to realistically model cyber attacks in a flexible manner. Moreover, the framework includes a suite of security metrics that can measure the attacker's effort, the defender's effort, and the security effectiveness of network diversity. To the best of our knowledge, this is the first framework geared towards quantifying the security effectiveness of network diversity.

Second, we conduct systematic simulations to quantify the security effectiveness of network diversity. The findings include (and will be elaborated in Section 5.3):

- Diversity does *not necessarily always* improve security from a whole-network perspective, because the security effectiveness of network diversity largely depends on the security quality of the diversified implementations.

- The *independence* assumption of vulnerabilities in the diversified implementations does cause an overestimate of security effectiveness in terms of the attacker's effort.

- Given a fixed attack capability, increasing diversity effort can lead to a higher security as long as there are always some vulnerabilities that cannot be exploited by the attacker.

- When diversity can improve security, enforcing diversity at multiple layers leads to higher security than enforcing diversity at a single layer.

- Two most effective defense strategies are (i) reducing software vulnerabilities or preventing attackers from obtaining exploits, and (ii) enforcing tight access control in host-based intrusion prevention systems (e.g., any function calls or communications that are not explicitly authorized are blocked).

### 5.1.2 Related Work

Software diversity has been advocated for security purposes [51, 125, 166]. The most closely related prior work is perhaps [104], which investigates how to configure diversified software implementations on computers. Their goal is to minimize the number of neighboring nodes that have the same software implementation, and/or maximize the number of subnetworks that run the same software. This means that [104] is *algorithmic* in nature. In contrast, we use the Cybersecurity Dynamics framework [152] to model and analyze the security effectiveness of enforcing network diversity. When compared with [104], our work can be characterized as follows: (i) we quantify the security effectiveness of enforcing network diversity, leading to insights that are not known until now; (ii) our model is fine-grained. Indeed, our model is finer grained than the numerous models in the Cybersecurity Dynamics framework (see, for example, [17, 22–24, 56, 79, 86, 89, 146, 148–152, 156, 159, 167, 168]). This is because we explicitly model the caller-callee dependence relation between software components.

The idea of software diversity, especially N-version programming [10, 26], was originally proposed to enhance fault tolerance under the assumption that software faults occur *independently* and *randomly*. Unfortunately, this assumption may not hold in general because programmers may make the same mistakes [40, 71], and because attacks are specifically geared towards software vulnerabilities (i.e., attacks are neither independent nor random). This means that the security value of enforcing software diversity must be re-examined in realistic threat models. To the best of our knowledge, the present study is the first effort aiming at systematically quantifying and characterizing the security effectiveness of software diversity *without* making the independence assumption between multiple implementations of the same software program. Indeed, we show that, in contrast to its fault-tolerance effectiveness, network diversity does *not* necessarily improve security when the diversified implementations possibly have the same security quality as the monoculture software implementation (i.e., containing the same amount of vulnerabilities). The issue of *dependence* in the cybersecurity domain has been investigated in the Cybersecurity Dynamics framework, including the dependence between random variables [36,146,147] and the dependence between cybersecurity time series data (instantiating stochastic processes) [108,143]. Indeed, dependence has been listed as one of the technical barriers that need to be adequately tackled for modeling and quantifying cybersecurity from a holistic perspective [152].

Another specific method for achieving diversity is to use compiler techniques [35,46,61,73]. In principle, the resulting diversified versions can be treated the same as the N-version programming. Moreover, our framework can accommodate a wide range of scenarios, from independence to dependence. We refer to [72] for an outstanding systematization of knowledge in this diversification approach. There are also proposals for runtime diversity, including address space randomization [14,41,45,140], instruction set randomization [11,70], and randomizing system calls [29]. Effectiveness and weakness of these techniques have been analyzed in [117, 120, 124] from a building-block perspective rather than from the perspective of looking at a network as a whole. Because these diversity techniques are complementary to software diversity, which is the focus of the present work, the present framework may be extended to investigate the security effectiveness of these techniques as well. Moreover, researchers have proposed the notion of N-variant systems to achieve higher assurance in detecting attacks [34,59,71].

Quantifying security is related to security metrics, for which there are three recent surveys [103,106,112] and some recent advancements are [23,39]. It is worth mentioning that our graph-theoretic framework is different from the framework of Attack Graphs [7,8,28,60,109,114,121,134,164], because the former models the dynamics (i.e., time-dependent) and the latter is *combinatorial* in nature (i.e., time-independent).

### 5.1.3 Chapter Organization

The rest of the Chapter is organized as follows. Section 5.2 presents the proposed framework. Section 5.3 describes our simulation experiments and insights drawn from our experimental results.

Section 5.4 concludes the work.

## 5.2 Instantiating the Framework to Quantify Security Effectiveness of Fine-Grained Static Network Diversity

In order to quantify the security effectiveness of static network diversity, we propose a security quantification framework, specifying: (i) how to represent an enterprise network; (ii) how to represent software vulnerabilities in the network; (iii) how to represent attacks; (iv) how to represent static network diversity and other defenses; (v) how to define security metrics for computing the security effectiveness of static network diversity. In the following sections, we elaborate the key components of the security quantification framework. Table 1 summarizes the key notations.

### 5.2.1 Representation of Enterprise Networks

A network is represented by software stacks, computers, inter-computer communication relations, and internal-external communication relations.

#### Representation of Software Stacks

In order to represent the software stacks of computers in a network, we first identify a *granularity*, namely the "atomic" unit (e.g., treating a computer vs. a software component as an atomic object). For quantifying the security effectiveness of network diversity, treating each computer as an atomic unit is too coarse-grained. Instead, we consider three types of software running on a computer: *applications*, including the library functions defined by the applications; *libraries*, including standard library functions and non-standard library functions (e.g., system or third-party ones); *operating system* running in the kernel space.

**Representation of applications**. There are many kinds of applications, including client (e.g., browsers and email clients), server (e.g., web server, email server), peer-to-peer (P2P), and stand-alone (e.g., word processor). An application may include some library functions defined by the application developer. We treat each application as an atomic object, because (i) application is a

| | |
|---|---|
| application | $\mathsf{APP}$ is the universe of applications; $\eta : \mathsf{APP} \rightarrow \{0,1,2\}$ indicates the type of an application (client vs. server vs. other); $\mathsf{app}_{i,z} \in \mathsf{APP}$ is the $z$-th application running on computer $i$ |
| library | $\mathsf{LIB}$ is the universe of libraries; $\mathsf{lib}_{i,j} \in \mathsf{LIB}$ is the $j$-th library running on computer $i$; $f_{i,j,z} \in \mathsf{lib}_{i,j}$ is the $z$-th library function in $\mathsf{lib}_{i,j}$ |
| os | $\mathsf{OS}$ is the universe of operating systems; $\mathsf{os}_i \in \mathsf{OS}$ is the operating system running on computer $i$; $k_{i,z} \in \mathsf{os}_i$ is the $z$-th operating system function in $\mathsf{os}_i$ |
| $G_i$ | $G_i = (V_i, E_i)$ represents a computer; $V_i = V_{i,app} \cup V_{i,lib} \cup V_{i,os}$; $E_i = E_{i,a} \cup E_{i,l} \cup E_{i,al} \cup E_{i,lk} \cup E_{i,ak} \cup E_{i,kk}$ |
| $G = (V, E)$ | $G$ represents a network of $n$ computers, where $V = V_1 \cup \ldots \cup V_n$ and $E = E_1 \cup \ldots \cup E_n \cup E_0 \cup E_*$ |
| diversity | $Y^{(N)}$ is the universe of diversified implementations of $Y \in \{\mathsf{APP}, \mathsf{LIB}, \mathsf{OS}\}$; $N_Z$ represents the number of independent implementations of $Z$ |
| vulnerability | $B$ is the universe of software vulnerabilities; $\phi(v) \subseteq B$ is the set of vulnerabilities a node $v \in V$ contains; $\mathsf{zd}(\mathsf{vul})$ indicates whether $\mathsf{vul} \in B$ is known ('0') or zero-day ('1'); $\mathsf{loc}(\mathsf{vul})$ indicates whether $\mathsf{vul}$ can be exploited remotely ('1') or not ('0'); $\mathsf{priv}(\mathsf{vul})$ indicates whether the exploitation of $\mathsf{vul}$ causes the attacker to obtain the `root` privilege ('1') or not ('0'); $\psi(v)$ for $v \in V_i$ indicates whether the user of computer $i$ is ('1') or is not ('0') vulnerable to social engineering attacks |
| $\gamma$ | $\gamma \in [0,1]$ is the failure probability of network-based intrusion prevention mechanism; $\gamma_{(\mathsf{app}_1, \mathsf{app}_2)} \in [0,1]$ is the failure probability in blocking attacks from $\mathsf{app}_1$ to $\mathsf{app}_2$; $\gamma_{(*,\mathsf{app})} \in [0,1]$ is the failure probability in blocking inbound attacks |
| $\alpha$ | $\alpha(v) \in [0,1]$ is the failure probability that a social engineering attack against $v$ is not blocked |
| $\mathsf{state}(v,t)$ | The probability $v \in V_{(app)} \cup V_{(os)}$ is compromised at time $t$ |
| exploit | $X$ is the set of exploits available to the attacker; $\rho(x, \mathsf{vul})$ is the probability that $x \in X$ can exploit vulnerability $\mathsf{vul} \in B$; $\omega$ is fraction of initially compromised targets; $\mathsf{cap} = \lvert \mathsf{vul} \in B : \exists x \in X,\ \rho(x, \mathsf{vul} = 1) \rvert / \lvert B \rvert$ is the fraction of software vulnerabilities that can be exploited by the attacker |
| metrics | $M = \{m_i\}$ is a set of security metrics |
| simulation | $\zeta(v)$ is the probability that a software running at $v \in V$ or simply $v \in V$ is vulnerable; $\vartheta(\mathsf{vul})$ is the probability $\mathsf{vul} \in B$ is remotely exploitable; $\tau(\mathsf{vul})$ is the probability that $\mathsf{vul} \in B$ is zero-day |

**Table 5.1**: Summary of key notations used for quantifying security effectiveness of static network diversity.

natural unit of diversified implementation; (ii) an application is a privilege entity, meaning that if any part of an application is compromised, the entire application is compromised; (iii) an application can be an entry-point for an attacker to remotely penetrate into a computer (e.g., remote code execution); and (iv) attack damages are caused by the invocation of system calls (i.e., syscalls) made by applications.

Let $\mathsf{APP}$ denote the universe of applications. For computer $i$ in a network, we denote by $\mathsf{app}_{i,z}$ the $z$-th application running on the computer, where $\mathsf{app}_{i,z} \in \mathsf{APP}$. We distinguish applications by defining the following mathematical function:

$$\eta : \mathsf{APP} \to \{0, 1, 2\} \tag{5.1}$$

such that

$$\eta(\mathsf{app}) = \begin{cases} 0 & \mathsf{app} \in \mathsf{APP} \text{ is a client } \mathsf{app} \\ 1 & \mathsf{app} \in \mathsf{APP} \text{ is an Internet-facing server } \mathsf{app} \\ 2 & \mathsf{app} \in \mathsf{APP} \text{ is an internal server } \mathsf{app} \end{cases}$$

This classification is plausible because each class may be subject to different attacks. For example, client applications (e.g., browsers and email clients) may be vulnerable to social engineering attacks, but the others may not be. An external attacker may directly compromise an Internet-facing server, but not an internal server unless the attacker already penetrated into the network.

**Representation of libraries**. We treat each library function as an atomic object because of the following: (i) if a library function has a vulnerability, then an application that invokes this function is compromised when the vulnerability is exploited; and (ii) we need to distinguish the library functions that make system calls from those which do not. This is important because a library function that makes system calls can be leveraged to exploit a vulnerability in the operating system, but a library function that does not make any system call cannot be leveraged to exploit a vulnerable operating system. Let $\mathsf{LIB}$ denote the universe of libraries. For computer $i$ in a network, we denote by $\mathsf{lib}_{i,j}$ the $j$-th library running on the computer, and by $f_{i,j,z}$ the $z$-th library function in $\mathsf{lib}_{i,j}$, where $\mathsf{lib}_{i,j} \in \mathsf{LIB}$ and $f_{i,j,z} \in \mathsf{lib}_{i,j}$.

**Representation of operating systems**. Similar to the treatment of library functions, we treat each $\mathsf{OS}$ function as an atomic object. This is because an $\mathsf{OS}$ function may have a vulnerability, but the vulnerability can be exploited only when the $\mathsf{OS}$ function is *syscalled*. That is, we should differentiate the $\mathsf{OS}$ functions that are *syscalled* from those which are not. Let $\mathsf{OS}$ denote the universe of operating systems. For computer $i$ in a network, we denote by $\mathsf{os}_i$ the operating system running on the computer and denote by $k_{i,z}$ the $z$-th operating system function in $\mathsf{os}_i$, where $\mathsf{os}_i \in \mathsf{OS}$ and $k_{i,z} \in \mathsf{os}_i$.

## Representation of Computers

Fig. 5.1 shows a toy example of a computer, running three applications denoted by $\mathsf{app}_{i,1}$, $\mathsf{app}_{i,2}$ and $\mathsf{app}_{i,3}$. Let $V_{i,app}$ denote the set of application running on computer $i$, and be defined by:

$$V_{i,app} = \{\mathsf{app}_{i,1}, \mathsf{app}_{i,2}, \mathsf{app}_{i,3}\}. \tag{5.2}$$

There are three libraries, each of which is composed of multiple functions. For example, library $\mathsf{lib}_{i,1}$ consists of functions $f_{i,1,1}, f_{i,1,2}, f_{i,1,3}$, and thus denoted by $\mathsf{lib}_{i,1} = \{f_{i,1,1}, f_{i,1,2}, f_{i,1,3}\}$. Let $V_{i,lib}$ denote the library functions running on computer $i$, which is defined by:

$$\begin{aligned} V_{i,lib} &= \mathsf{lib}_{i,1} \cup \mathsf{lib}_{i,2} \cup \mathsf{lib}_{i,3} \\ &= \{f_{i,1,1}, f_{i,1,2}, f_{i,1,3}, f_{i,2,1}, f_{i,2,2}, f_{i,3,1}, f_{i,3,2}\}. \end{aligned} \tag{5.3}$$

The operating system, $\mathsf{os}_i$, has ten kernel functions, meaning

$$\mathsf{os}_i = \{k_{i,1}, k_{i,2}, k_{i,3}, k_{i,4}, k_{i,5}, k_{i,6}, k_{i,7}, k_{i,8}, k_{i,9}, k_{i,10}\}.$$

Since operating system functions run at the same privilege level, we use $V_{i,os}$ to denote the operating system functions of computer $i$, and $V_{i,os}$ is given by:

$$V_{i,os} = \mathsf{os}_i = \{k_{i,1}, k_{i,2}, k_{i,3}, \ldots, k_{i,10}\}. \tag{5.4}$$

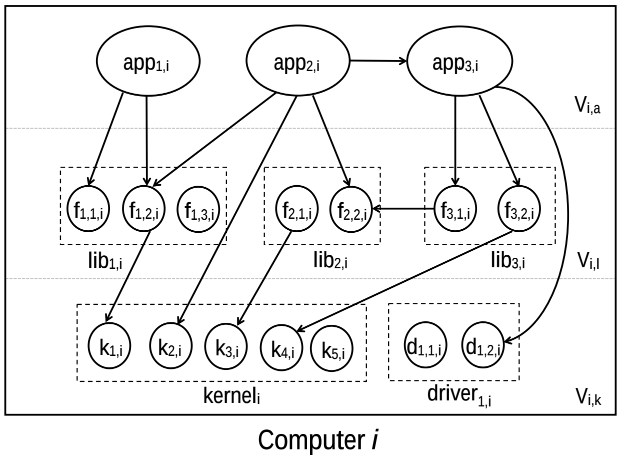


**Figure 5.1**: A graph-theoretic representation of computer $i$ (or the $i$-th computer), denoted by $G_i = (V_i, E_i)$.

We further consider the *dependence* relations between the atomic objects, namely the *caller-callee* relation. For example, an application may make a syscall directly or indirectly (i.e., an application calls a library function which further makes a *syscall*). The dependence relation should be accommodated because a vulnerability in a callee can be exploited by a caller. Fig. 5.1 illustrates the following dependence and communication relations. Correspondingly, we model computer $i$ as a graph $G_i = (V_i, E_i)$, where $V_i$ is the *node set* and $E_i$ is the *arc set* (meaning that the graph is directed in general). $V_i$ is denoted by:

$$V_i = V_{i,app} \cup V_{i,lib} \cup V_{i,os}, \tag{5.5}$$

where $V_{i,app}$, $V_{i,lib}$, and $V_{i,os}$ are respectively given by Eqs. (5.2), (5.3), (5.4). The arc set is denoted by:

$$E_i = E_{i,al} \cup E_{i,ll} \cup E_{i,lk} \cup E_{i,ak} \cup E_{i,kk} \tag{5.6}$$

describes the following relations:

- $E_{i,al}$ represents the *dependence* relation between applications and the library functions. For example, in Fig. 5.1 we have $E_{i,al} = \{(\mathsf{app}_{i,1}, f_{i,1,1}), (\mathsf{app}_{i,1}, f_{i,1,2}), (\mathsf{app}_{i,2}, f_{i,1,2}), (\mathsf{app}_{i,2}, f_{i,2,2}), (\mathsf{app}_{i,3}, f_{i,3,1}), (\mathsf{app}_{i,3}, f_{i,3,2})\}$.

- $E_{i,ll}$ represents the *dependence* relation between library functions. For example, in Fig. 5.1, we have $E_{i,l} = \{(f_{i,3,1}, f_{i,2,2})\}$ because $f_{i,3,1}$ calls $f_{i,2,2}$.

- $E_{i,lk}$ represents the *dependence* relation between the library functions and the operating system functions. For example, in Fig. 5.1 we have $E_{i,lk} = \{(f_{i,1,2}, k_{i,1}), (f_{i,2,2}, k_{i,5})\}$.

- $E_{i,ak}$ represents the *dependence* relation between applications and the operating system functions. For example, in Fig. 5.1, we have $E_{i,ak} = \{(\mathsf{app}_{i,2}, k_{i,3})\}$.

- $E_{i,kk}$ represents the *dependence* relation between the operating system functions. For example, in Fig. 5.1 we have $E_{i,kd} = \{(k_{i,4}, k_{i,1}), (k_{i,8}, k_{i,10})\}$.

Putting the preceding discussion together, we obtain the representation of computer $i$ as a graph

$$G_i = (V_i, E_i) \tag{5.7}$$

where $V_i$ and $E_i$ are respectively defined in Eqs. (5.5) and (5.6).

**Representation of Inter-Computer Communication Relations Within a Network**

Fig. 5.2 illustrates a toy network of computer $i$ and computer $j$, which are respectively described by graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$. The inter-computer *communication* relation describes which applications running on one computer are designed to communicate with which other applications running on another computer. We formally use arc set $E_0$ to represent the inter-computer communication relation between applications on computer $i$ and applications on computer $j$, where

$$E_0 \subseteq \{V_{i,app} \times V_{j,app}\} \cup \{V_{j,app} \times V_{i,app}\}, \tag{5.8}$$

where $1 \leq i, j \leq n$ for a network of $n$ computers and $i \neq j$. In Fig. 5.2, $\mathsf{app}_{i,2}$ running on computer $i$ is allowed to communicate with $\mathsf{app}_{j,1}$ running on computer $j$ (e.g., browser to web server). Therefore, we have $E_0 = \{(\mathsf{app}_{i,2}, \mathsf{app}_{j,1})\}$. Note that $e = (\mathsf{app}_1, \mathsf{app}_2) \in E_0$ does not necessarily correspond to a physical network link, be it wired or wireless. Instead, $e \in E_0$ often corresponds to a communication path.

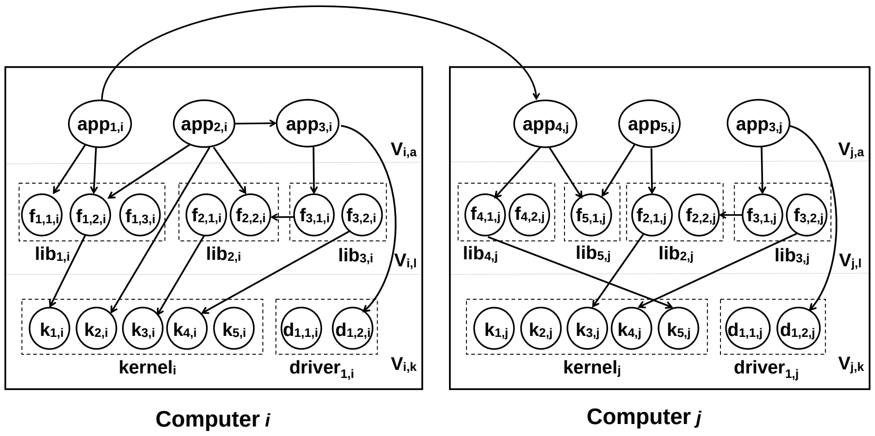


**Figure 5.2**: Illustration of the *communication relation* with $E_0 = \{(\mathsf{app}_{i,2}, \mathsf{app}_{j,1})\}$.

We accommodate the communication relation because compromised computers are often used as "stepping stones" to attack other computers. For example, $(\mathsf{app}_{i,2}, \mathsf{app}_{j,1}) \in E_0$ means that the compromise of $\mathsf{app}_{i,2}$ may cause the compromise of $\mathsf{app}_{j,1}$ when $\mathsf{app}_{j,1}$ has a vulnerability that can be exploited remotely. In order to distinguish between two kinds of attacks that can be waged over $e \in E_0$, we partition $E_0$ into $E_{00}$ and $E_{01}$ such that $E_0 = E_{00} \cup E_{01}$, where $E_{00}$ represents the attacks against clients (including peers in peer-to-peer application), and $E_{01}$ represents the attacks against servers. More specifically, we have:

- $(\mathsf{app}_{i,x}, \mathsf{app}_{j,y}) \in E_{00}$ corresponding to computers $i$ and $j$ in the network: This set represents attacks that can be launched from a server or client or peer application, say $\mathsf{app}_{j,y}$, against a vulnerable client application say $\mathsf{app}_{i,x}$ with $\eta(\mathsf{app}_{i,x}) = 0$.

- $E_{01} = E_0 \setminus E_{00}$: Any inter-computer communication other than what are accommodated by $E_{00}$.

### Representation of Internal-External Communication Relations

A network is often a part of the Internet. This means that the computers in a network may communicate with the computers outside the network. Therefore, we model the following two relations. One is the *internal-to-external* communication relation. A computer, say $V_i$, communicates with computers outside the network. This is done by some application running on $V_i$, say $\mathsf{app}_{i,1}$. We use arc set $E_{*,io} = \{(\mathsf{app}_{i,1}, *)\}$ to describe such internal-to-external communications, where the wildcat "$*$" means any computer that resides outside of the network. The other is the *external-to-internal* communication relation. A computer, typically a server say $V_j$, is outfacing, meaning that a server application, say $\mathsf{app}_{j,1}$, can be accessed from any computer outside the network. We use an arc set $E_{*,oi} = \{(*, \mathsf{app}_{j,1})\}$ to describe such external-to-internal communications, where the wildcat "$*$" means that any computer outside the network can communicate with $\mathsf{app}_{j,1}$. Correspondingly, we define $E_*$ as:

$$E_* = E_{*,io} \cup E_{*,oi}. \tag{5.9}$$

Modeling $E_*$ is important because it is related to *initial compromise*, which deals with how an attacker penetrates into a network. For example, $E_{*,io}$ can be leveraged to wage social engineering attacks (e.g., spearfishing) and $E_{*,oi}$ can be leveraged to compromise an outfacing server.

### Representation of Networks

Putting together what we have discussed, a network of $n$ computers is represented by $G = (V, E)$, where

$$V = V_1 \cup \ldots \cup V_n \text{ and } E = E_1 \cup \ldots \cup E_n \cup E_0 \cup E_*, \tag{5.10}$$

where $G_i = (V_i, E_i)$ is given by Eq. (5.7) and represents computer $i$, $E_0$ is given by Eq. (5.8) and represents the inter-computer communication relations between computers in the network, and $E_*$ is given by Eq. (5.9) and represents the internal-external communication relations.

For ease of reference, we will use $V_{(app)}$, $V_{(lib)}$ and $V_{(os)}$ to respectively denote the set of applications, libraries and operating systems running in the computers of a network, namely

$$\begin{aligned} V_{(app)} &= V_{1,app} \cup \ldots \cup V_{n,app}, \quad (5.11) \\ V_{(lib)} &= V_{1,lib} \cup \ldots \cup V_{n,lib}, \quad (5.12) \\ V_{(os)} &= V_{1,os} \cup \ldots \cup V_{n,os}. \quad (5.13) \end{aligned}$$

We will use $v \in V$ to indicate an arbitrary node $v$.

### 5.2.2 Representation of Vulnerabilities

Let $B$ denote the set of vulnerabilities that exist in the network comprising $n$ computers. We represent the intrinsic characteristics of each vulnerability $\mathsf{vul} \in B$ by five attributes. The first three attributes involves in the *exploitation* of a vulnerability, meaning the ease and technical means by which the vulnerability can be exploited. They are defined following the common vulnerability scoring system (CVSS) [96] but a bit simplified. The fourth attribute focuses on the *reachability* of a vulnerability, meaning whether there exists function paths through which an attacker can reach the vulnerability. The last attribute considers the *impact* of a vulnerability, meaning the direct consequence of a successful exploit of the vulnerability.

- $\mathsf{loc}$: This attribute describes whether $\mathsf{vul}$ can be exploited remotely or not. We define predicate $\mathsf{loc}$ such that $\mathsf{loc}(\mathsf{vul}) = 0$ means $\mathsf{vul}$ cannot be exploited remotely, and $\mathsf{loc}(\mathsf{vul}) = 1$ otherwise.

- $\mathsf{AC}$: This attribute describes how difficult it is to exploit $\mathsf{vul}$. We define predicate $\mathsf{AC}$ such that $\mathsf{AC}(\mathsf{vul}) = 0$ means high attack complexity (we consider such attack as impossible for simplicity), and $\mathsf{AC}(\mathsf{vul}) = 1$ means low or medium attack complexity.

- $\mathsf{auth}$: This attribute describes the type of authentication required before exploiting $\mathsf{vul}$. We define predicate $\mathsf{auth}$ such that $\mathsf{auth}(\mathsf{vul}) = 0$ means $\mathsf{vul}$ can be exploited without any authentication, and $\mathsf{auth}(\mathsf{vul}) = 1$ otherwise.

- $\mathsf{rch}$: This attribute describes the reachability of $\mathsf{vul}$. We use predicate $\exp$ such that $\mathsf{rch}(\mathsf{vul}) = 0$ means the function containing $\mathsf{vul}$ is not in the attack surface of the targeted computer, and $\mathsf{rch}(\mathsf{vul})=1$ means otherwise.

- $\mathsf{priv}$: This attribute describes the access privileges an attacker can obtain by exploiting $\mathsf{vul}$. We use predicate $\mathsf{priv}$ such that $\mathsf{priv}(\mathsf{vul}) = 0$ means an exploitation of $\mathsf{vul}$ does not give the attacker the `root` privilege, and $\mathsf{priv}(\mathsf{vul}) = 1$ means otherwise.

These attributes allow us to accommodate attacks, such as *remote-2-user* attacks [38, 52, 111] that exploit $\mathsf{vul}$ with $\mathsf{loc}(\mathsf{vul}) = 1$, $\mathsf{AC}(\mathsf{vul}) = 1$, $\mathsf{rch}(\mathsf{vul})=1$ and $\mathsf{priv}(\mathsf{vul}) = 0$, *remote-2-root* attacks [68, 111, 121] that exploit $\mathsf{vul}$ with $\mathsf{loc}(\mathsf{vul}) = 1$, $\mathsf{AC}(\mathsf{vul}) = 1$, $\mathsf{rch}(\mathsf{vul})=1$ and $\mathsf{priv}(\mathsf{vul}) = 1$, and *user-2-root* attacks [52, 111, 127] that exploits $\mathsf{vul}$ with $\mathsf{loc}(\mathsf{vul}) = 1$, $\mathsf{AC}(\mathsf{auth}) = 1$, $\mathsf{AC}(\mathsf{vul}) = 1$, $\mathsf{rch}(\mathsf{vul})=1$ and $\mathsf{priv}(\mathsf{vul}) = 1$. We admit that the present model does not accommodate all attacks (e.g., side-channel attacks), which will need to be accommodated in future work.

### 5.2.3 Representation of Attacks

We describe attacks by distinguishing the *exploits* that are available to an attacker, and the *strategies* prescribing how the exploits will be used collectively. To describe the attack strategies, we define a predicate, $\mathsf{state}(v, t)$, for $v \in V_{(app)} \cup V_{(os)}$ such that $\mathsf{state}(v, t) = 0$ means $v$ is not compromised at time $t$ while $\mathsf{state}(v, t) = 1$ means $v$ is compromised at time $t$. Note that application $v \in V_{(app)}$ can be compromised because a software vulnerability in the application or in the library function it calls is exploited, or because the underlying operating system is compromised. Note that the predicate $\mathsf{state}(v, t)$ is *not* defined for $v \in V_{(lib)}$ because library functions are loaded into the program space of an application at runtime. Now we discuss how ‘exploits’ and ‘attack strategies’ are represented in the proposed framework as below.

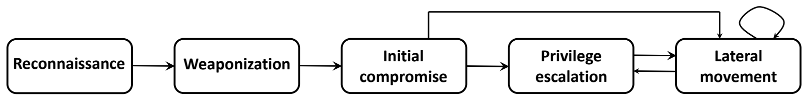


**Figure 5.3**: The proposed attack lifecycle model inspired by [65, 95].

### Representation of Attack Capabilities

Let $X$ denote the set of exploits available to the attacker. We define the following mathematical function:

$$\rho : X \times B \rightarrow [0, 1] \tag{5.14}$$

such that $\rho(x, \mathsf{vul})$ is the success probability when applying exploit $x \in X$ against vulnerability $\mathsf{vul} \in B$. For simplicity, we only consider $\rho(x, \mathsf{vul}) = 0$ or $\rho(x, \mathsf{vul}) = 1$.

### Representation of Attack Strategies

We represent attack strategies according to the attack lifecycle model highlighted in Fig. 5.3. This flexible model is adapted from the Cyber Kill Chain [65] and the Attack Life Cycle [95]. The model has seven phases: *reconnaissance*, *weaponization*, *initial compromise*, *escalate privileges*, *lateral movement*, *persistence*, and *completion*. These phases are elaborated below.

**Phase 1: Reconnaissance.** An attacker uses reconnaissance to collect information about a target network by identifying vulnerabilities the attacker can possibly exploit. In a given network $G = (V, E)$, we associate each node $v \in V$ with the following data structure to represent its vulnerability information: the type $\eta(v)$ of the application running at $v \in V_{(app)}$, namely client or server; a set $\phi(v)$ of software vulnerability $v \in V$ contains; and human factor vulnerability $\psi(v)$ of $v \in V$. Moreover, for each vulnerability $\mathsf{vul} \in \phi(v)$, the attacker further knows that its attributes include the following: (i) $\mathsf{zd}(\mathsf{vul})$, whether the vulnerability is zero-day or not; (ii) $\mathsf{loc}(\mathsf{vul})$, whether the vulnerability can be remotely exploited or not; and (iii) $\mathsf{priv}(v)$, whether the exploitation of a vulnerability can cause a privilege escalation or not.

**Phase 2: Weaponization.** Given $G = (V, E)$, an outcome of the reconnaissance step, and an attacker's set of exploits $X$, the attacker now selects some nodes $v \in V_i$ for initial compromise.

For this purpose, the attacker needs to annotate the vulnerabilities that can be exploited by its exploits. There are two cases.

In the case that $v \in V_i$ is client application $\mathsf{app} \in \mathsf{APP}^{(N)}$ running on computer $i$, meaning $\eta(v) = \eta(\mathsf{app}) = 0$ and $(\mathsf{app}, *) \in E_{*,io}$, the attacker can exploit social-engineering attacks to compromise $v$ under one of the following two conditions: (i) $\mathsf{app}$ contains a software vulnerability, namely $\exists \mathsf{vul} \in \phi(v) = \phi(\mathsf{app})$, or (ii) the $\mathsf{app}$ contains no vulnerability but a library function or operating system function that is called by the $\mathsf{app}$ contains a software vulnerability (i.e., there existing an access path from a secure $\mathsf{app}$ to a vulnerable library function or operating system function).

To be specific, the client application $\mathsf{app}$ is considered by the attacker as a candidate for *initial compromise* only when the following condition holds:

$$\begin{aligned}(\exists \mathsf{vul} \in \phi(v), \exists x \in X : \psi(v) = 1 \wedge \rho(x, \mathsf{vul}) = 1) \vee \\ (\exists \mathsf{vul} \in \phi(u), \exists x \in X : (u \in V_{i,lib} \cup V_{i,os}) \wedge (v \in V_{i,app}) \wedge \\ \mathsf{dep\_path}(v, u) \wedge \psi(u) = 1 \wedge \rho(x, \mathsf{vul}) = 1).\end{aligned} \tag{5.15}$$

The set of client applications that can be leveraged to penetrate into a network is:

$$\mathsf{Weapon}_0 = \{v \in V_{i,app} : \eta(v) = 0 \wedge \text{condition (5.15) holds}\}.$$

In the case that $v \in V_{i,app}$ is a server application $\mathsf{app} \in \mathsf{APP}^{(N)}$ running on outfacing computer $i$, meaning $\eta(v) = \eta(\mathsf{app}) = 1$ and $(*, \mathsf{app}) \in E_{*,io}$, we assume that the user of the server is not vulnerable to social engineering attacks as discussed above, meaning $\psi(v) = \psi(\mathsf{app}) = 0$. Then, server application $\mathsf{app}$ can be compromised under one of the following two conditions: (i) $\mathsf{app}$ contains a remotely-exploitable software vulnerability, namely $\exists \mathsf{vul} \in \phi(v)$, where $\phi(v) = \phi(\mathsf{app})$, such that $\mathsf{loc}(\mathsf{vul}) = 1$; or (ii) there is a library function or operating system function that is called by $v$ (i.e., the $\mathsf{app}$) and that contains a remotely exploitable vulnerability.

More precisely, a server application $\mathsf{app}$ is considered by the attacker as a candidate for *initial* *compromise* only when the following condition holds:

$$
\begin{aligned}
(\exists \mathsf{vul} \in \phi(v), \exists x \in X : \mathsf{loc}(\mathsf{vul}) = 1 \wedge \rho(x, \mathsf{vul}) = 1) \vee \\
(\exists \mathsf{vul} \in \phi(u), \exists x \in X : (u \in V_{i,lib} \cup V_{i,os}) \wedge (v \in V_{i,app}) \wedge \\
\mathsf{dep\_path}(v, u) \wedge \mathsf{loc}(\mathsf{vul}) = 1 \wedge \rho(x, \mathsf{vul}) = 1.
\end{aligned}
\tag{5.16}
$$

The set of server applications that can be leveraged to penetrate into a network is:

$$\mathsf{Weapon}_1 = \{v \in V_{i,app} : \eta(v) = 1 \wedge \text{condition (5.16) holds}\}.$$

Summarizing the above, the set of applications that can be leveraged to penetrate into the network is defined by:

$$\mathsf{Weapon} = \mathsf{Weapon}_0 \cup \mathsf{Weapon}_1. \tag{5.17}$$

**Phase 3: Initial compromise.** Having determined Weapon according to Eq. (5.17), the attacker will select a subset of them to penetrate into the network, according to some attack tactics. In this paper, we consider the following tactics to reduce the chances that the attack is detected by the defender:

1. If the attacker can compromise the operating system by exploiting a vulnerability in $v \in V_{i,os}$, the attacker will choose to do so, even if the attacker can compromise some application belonging to $V_{i,app}$. This is because compromising the operating system causes the compromise of every $v \in V_{i,app} \cup V_{i,os}$ automatically. This tactic prevents the attacker from launching redundant attacks, and therefore possibly reduces the chance of its attacks being detected.

2. If the attacker cannot compromise the operating system on computer $i$, the attacker will compromise all of the applications on computer $i$ that can be compromised, which is defined by:

$$\{v \in V_{i,app} : v \in \mathsf{Weapon} \wedge (\exists x \in X, \exists \mathsf{vul} \in \phi(v) : \rho(x, v) = 1). \tag{5.18}$$

Other tactics may be possible (e.g., the attacker may compromise some applications, depending on its objective). These attack tactics will guide the attacker to select a subset of nodes for initial compromise, denoted by:

$$\mathsf{IniComp} = \{v \in \mathsf{Weapon} : \text{attacker selects } v \text{ to attack}\}.$$

**Phase 4: Privilege escalation.** We assume the attacker wants to get the `root` privilege whenever possible. Suppose the attacker only has obtained the `user` privilege at a computer, meaning that the attacker has compromised some $v \in V_{i,app}$ but not $v \in V_{i,os}$. In order to escalate to the `root` privilege, there are two cases, depending on the preventive defense policy is *tight* or *loose*.

In the case of *tight* policies, a whitelist-like mechanism is used to record the legitimate applications as well as the operating system functions they are authorized to call. Unless the host-based intrusion prevention system is compromised (i.e., the operating system is compromised), the attacker with a user privilege (by compromising an application) can neither run an arbitrary malicious program nor make any calls to unauthorized operating system functions *even if the latter vulnerable*. That is, a privilege escalation occurs under the following condition:

$$\begin{gathered}\exists v \in V_{i,app}, \exists u \in V_{i,os}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \\ \mathsf{state}(v,t) = 1 \land \mathsf{dep\_path}(v,u) \land \rho(x,\mathsf{vul}) = 1.\end{gathered}$$

In the case of *loose* policies, there is no whitelist-like mechanism. This means that an attacker with a user privilege (by compromising an application) can run an arbitrary malicious program or make calls to any vulnerable operating system functions to compromise them. That is, a privilege escalation occurs under the following condition:

$$\begin{aligned}\exists v \in V_{i,app}, \exists u \in V_{i,os}, \exists \mathsf{vul} \in \phi(u), \exists x \in X :& \\ \mathsf{state}(v,t) = 1 \land \rho(x,\mathsf{vul}) = 1.&\end{aligned}$$

**Phase 5: Lateral movement.** Suppose the attacker has compromised computer $i$, denoted by $\mathsf{state}(v,t) = 1$. Lateral movement means that the attacker attempts to compromise other computers in the network. There are two cases, depending on whether the network-based preventive defense is *tight* or *loose*.

In the case the network-based preventive defense is *tight*, communication over $(v,u) \notin E_0$ is blocked and therefore cannot be abused to wage attacks unless the enforcement mechanism or reference monitor is compromised (e.g., firewall). This forces the attacker to use an existing inter-computer communication relation $e \in E_{01}$ to attempt to attack another computer. Formally, a lateral movement from a compromised computer $i$ to vulnerable computer $j$ can happen under one of the following two conditions:

$$(\exists u \in V_{j,app}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \mathsf{state}(v,t) = 1 \wedge$$
$$\mathsf{state}(u,t) = 0 \wedge (v,u) \in E_0 \wedge \rho(x,\mathsf{vul}) = 1 \wedge \mathsf{loc}(\mathsf{vul}) = 1) \tag{5.19}$$
$$\vee(\exists u \in V_{j,app}, \exists w \in V_{j,lib} \cup V_{j,os}, \exists \mathsf{vul} \in \phi(w), \exists x \in X :$$
$$(\mathsf{state}(v,t) = 1) \wedge (\mathsf{state}(w,t) = 0) \wedge (v,u) \in E_0 \wedge$$
$$\mathsf{dep\_path}(u,w) \wedge \rho(x,\mathsf{vul}) = 1 \wedge \mathsf{loc}(\mathsf{vul}) = 1). \tag{5.20}$$

The first condition, Eq. (5.19), says a vulnerable application on computer $j$ can be exploited from a compromised application on computer $i$. The second condition, Eq. (5.20), says a vulnerable library or operating system function on computer $j$ can be exploited from a compromised application on computer $i$.

In the case the network-based preventive defense is *loose*, communication over $(v,u) \notin E_0$ is *not* blocked by any enforcement mechanism or reference monitor and therefore can be leveraged to launch attacks. Formally, a lateral movement from a compromised computer $i$ to a vulnerable computer $j$ can happen under one of the following two conditions:

$$\begin{aligned}
&(\exists u \in V_{j,app}, \exists \mathsf{vul} \in \phi(u), \exists x \in X : \mathsf{state}(v,t) = 1 \wedge \\
&\qquad \mathsf{state}(u,t) = 0 \wedge \rho(x,\mathsf{vul}) = 1 \wedge \mathsf{loc}(\mathsf{vul}) = 1) \qquad (5.21) \\
&\vee(\exists u \in V_{j,app}, \exists w \in V_{j,lib} \cup V_{j,os}, \exists \mathsf{vul} \in \phi(w), \exists x \in X : \\
&\qquad \mathsf{state}(v,t) = 1 \wedge \mathsf{state}(w,t) = 0 \wedge \\
&\qquad \mathsf{dep\_path}(u,w) \in E_j \wedge \rho(x,\mathsf{vul}) = 1 \wedge \mathsf{loc}(\mathsf{vul}) = 1). \qquad (5.22)
\end{aligned}$$

Note that Eqs. (5.21) and (5.22) are respectively the same as Eqs. (5.19) and (5.20), except that there is no requirement for $(v,u) \in E_0$ because the network-based preventive defense is loose.

### 5.2.4 Representation of Defenses

#### Representation of Static Network Diversity

A well-known approach to diversifying software is called *N-version programming* [10], meaning that a software has multiple independent implementations that are unlikely to have the same software bugs or vulnerabilities. Corresponding to the representation of software stacks, we let $\mathsf{APP}^{(N)}$ denote the universe of diversified implementations of the applications, where superscript $^{(N)}$ indicates N-version programming. For application $\mathsf{app} \in \mathsf{APP}$, there is a set of independent implementations, denoted by $\mathsf{app}^{(N)}$, where $|\mathsf{app}^{(N)}| \geq 1$. Note that $|\mathsf{app}^{(N)}| = 1$ means that there is no diversity for this application. Similarly, we let $\mathsf{LIB}^{(N)}$ denote the universe of diversified implementations of the libraries, where a library $\mathsf{lib} \in \mathsf{LIB}$ has a set of independent implementations, denoted by $\mathsf{lib}^{(N)}$, with $|\mathsf{lib}^{(N)}| \geq 1$. Let $\mathsf{OS}^{(N)}$ denote the universe of diversified implementations of the operating systems, where an operating system $\mathsf{os} \in \mathsf{OS}$ has a set of independent implementations, denoted by $\mathsf{os}^{(N)}$, with $|\mathsf{os}^{(N)}| \geq 1$.

Static network diversity is to diversify the software stacks of computers in networks. Consider network $G = (V,E)$ of $n$ computers, as defined in Eq. (5.10). The *software stack configuration* of computer $i$ is an assignment of specific implementations of applications, libraries, and operating system to run in computer $i$. For this purpose, we define a tuple of mathematical functions:

$$C = (C_{(app)}, C_{(lib)}, C_{(os)}), \tag{5.23}$$

where $C_{(app)} : V_{(app)} \rightarrow \mathsf{APP}^{(N)}$ assigns a specific implementation of application $\mathsf{app}_{i,j} \in \mathsf{APP}^{(N)}$ to run at node $v \in V_{(app)}$, $C_{(lib)} : V_{(lib)} \rightarrow \mathsf{LIB}^{(N)}$ assigns a specific implementation of library $\mathsf{lib}_{i,j} \in \mathsf{LIB}^{(N)}$ to run at node $v \in V_{(lib)}$, and $C_{(os)} : V_{(os)} \rightarrow \mathsf{OS}^{(N)}$ assigns a specific implementation of operating system $\mathsf{os}_i \in \mathsf{OS}^{(N)}$ to run at node $v \in V_{(os)}$.

**Representation of Other Defenses**

In this work, we only consider host-based defenses that prevent attacks from succeeding, including Host-based Intrusion Prevention System (HIPS) and patching. Since there are many defense mechanisms that might not be feasible to model individually, we simply model their *effect*. Note that the term *effect* is different from the term *effectiveness* as follows: effect includes changes introduced by a defense mechanism (e.g., cost, performance, and/or security changes), but effectiveness is limited in a positive aspect of influence (e.g., enhanced security). We consider two types of preventive defenses, *tight* and *loose*, in the contexts of network-based and computer-based defenses.

For network-based preventive defenses, a *tight* policy is mainly related to the enforcement of a whitelist, such that any communication attempts that are not specified by $E_0 \cup E_*$ will be blocked because these communications are not deemed as necessary by the applications. In contrast, a *loose* policy does not block the traffic not complying to $E_0 \cup E_*$. For example, we can consider the following:

- Consider communication link $e = (\mathsf{app}_1, \mathsf{app}_2)$, where $\mathsf{app}_1, \mathsf{app}_2 \in \mathsf{APP}^{(N)}$ run on two different computers. If the preventive defense is *tight*, $e \notin E_0$ means the traffic over $e$ is blocked; otherwise, the traffic is further examined by a network-based intrusion prevention mechanism, which fails to detect an attack launched from $\mathsf{app}_1$ to $\mathsf{app}_2$ with a probability, denoted by $\gamma_{(\mathsf{app}_1, \mathsf{app}_2)} \in [0, 1]$. For simplicity, we assume that these probabilities are arc-independent, meaning $\gamma = \gamma_{(\mathsf{app}_1,\mathsf{app}_2)}$ for any $\mathsf{app}_1, \mathsf{app}_2 \in \mathsf{APP}^{(N)}$; this corresponds to the case that the defender deploys the same network-based intrusion prevention system network-wide.

- Consider communication link $e = (*, \mathsf{app}) \in E_{*,oi}$, where $\mathsf{app} \in \mathsf{APP}^{(N)}$ and $\eta(\mathsf{app}) = 1$. We associate $e$ with a parameter $\gamma_{(*,\mathsf{app})} \in [0,1]$, which describes the probability that an inbound attack is *not* detected or blocked. For simplicity, we assume that these probabilities are arc-independent, meaning $\gamma = \gamma_{(*,\mathsf{app})}$ for any $\mathsf{app} \in \mathsf{APP}^{(N)}$; this corresponds to the case that the same network-based intrusion detection system is used in the entire network.

For computer-based or host-based preventive defenses, a *tight* policy is essentially the enforcement of a whitelist, including the applications authorized to run on a computer and the list of operating system functions these applications are authorized to call. As a result, the compromise of an application does not necessarily mean the attacker can abuse the compromised application to make calls to unauthorized, but vulnerable operating system functions. Moreover, the attacker cannot run a malicious application provided by itself. In contrast, a *loose* policy does not have such a whitelist. As a consequence, the compromise of an application allows the attacker to abuse the compromised application to make calls to any vulnerable operating system functions (e.g., privilege escalation). Moreover, the attacker can run any malicious application provided by itself.

In order to model computer-based or host-based preventive defenses against social engineering attacks that may be waged over $e \in E_{00} \cup E_{*,io}$, we associate each node of the following set

$$\{v \in V_{(app)} : \eta(v) = 0 \wedge ((v,*) \in E_{*,io} \vee (v,u) \in E_{00})\}, \tag{5.24}$$

with parameter $\alpha_{\mathsf{app}} \in [0,1]$ to describe the probability that a social engineering attack against $v$ is not detected or blocked. Note that all of the applications running on computer $i$ have the same parameter $\alpha_i$. For simplicity, we may assume the same $\alpha$ applies to all nodes belonging to the set of Eq. (5.24).

### 5.2.5 Metrics for Measuring the Cybersecurity State of an Enterprise Network

We further define security metrics to measure defender's effort, attacker's effort, and security effectiveness.

#### Defender's Effort

Defender's effort can be measured by the following metrics:

- *Diversity parameter* ($N$): This represents the number of implementations, $N$, each software has. Note that it is straightforward to extend this uniform parameter $N$ into a vector $(N_{(app)}, N_{(lib)}, N_{(os)})$ to accommodate that different software has different numbers of implementations.
- *Preventive defense effort*: This metric has two categories, *tight* vs. *loose*, where *tight* means the defender needs to make extra effort in figuring out which applications have to communicate with which other applications and which applications can call which libraries or syscalls. On the other hand, *loose* does not require any extra effort.

#### Attacker's Effort

Attacker's effort can be measured by the following metrics:

- *Initial compromise effort*: It refers to the fraction $\omega$ of initial compromises the attacker makes, denoted by $\omega \times |\mathsf{Weapon}|$.
- *Fraction* $\mathsf{cap}$ *of vulnerabilities*: This indicates the fraction of vulnerabilities that can be exploited by the attacker, denoted by $\mathsf{cap} = |X|/|B|$.

### Security Effectiveness

The attack consequence is represented by the predicate of $\mathsf{state}(v)$ of $v \in V$. We can define the state of an operating system as

$$\mathsf{state}(os_i) = \begin{cases} 1 & \exists v \in \mathsf{os}_i \text{ s.t. } \mathsf{state}(v) = 1 \\ 0 & otherwise \end{cases}$$

Moreover, we have

$$\forall \mathsf{app}_{i,j} \in V_{i,app} : \mathsf{state}(os_i) = 1 \Longrightarrow \mathsf{state}(\mathsf{app}_{i,j}) = 1.$$

Security effectiveness can be measured by two time-dependent metrics: *percentage of compromised applications* (pca) and *percentage of compromised operating systems* (pcos) at time $t$, namely

$$\mathsf{pca}(t) = |\{v \in V_{(app)} : \mathsf{state}(v,t) = 1\}|/|V_{(app)}|, \tag{5.25}$$

$$\mathsf{pcos}(t) = |\{v \in V_{(os)} : \mathsf{state}(v,t) = 1\}|/|V_{(os)}|. \tag{5.26}$$

while noting that we do not consider the state of libraries.

## 5.3 Simulation Experiments and Results

In this section, we use simulations to answer the following Research Questions (RQ):

- RQ1: Does natural diversity always lead to higher security? If not, when?
- RQ2: Does artificial diversity always lead to higher security? If not, when?
- RQ3: Does the use of natural and artificial diversity together always lead to higher security? If not, when?
- RQ4: What are the most effective defense strategies in the presence of network diversity?

### 5.3.1 Simulation Setting and Methodology

#### Simulating an Enterprise Network System

Fig. 5.4 is an example network, from which $G = (V, E)$ will be derived for the simulation study. The network has a DMZ (DeMilitarized Zone) consisting of a web server and an email server, a database zone with a database server, and 10 subnets with each having 200 hosts. In total, the network has 2,003 computers.

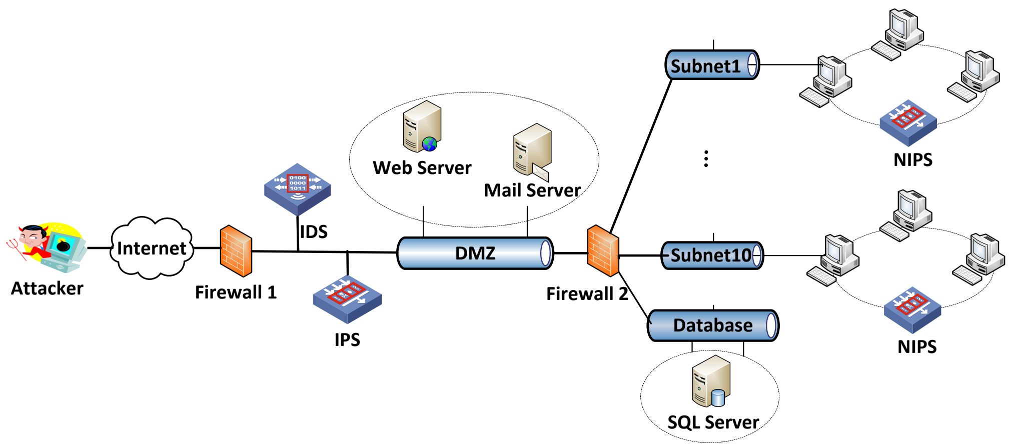


**Figure 5.4**: Example network used for the simulation study.

For the application layer, suppose each of the three servers only runs one application (i.e., web server, email server, and database, respectively). Suppose each computer in a subnet runs 4 applications, namely $\mathsf{APP} = \{$browser, email client, P2P, word processor$\}$, except for the experiment aiming to characterize the impact of the number of applications running on a computer. For the operating system layer, suppose there are two operating systems, namely $\mathsf{OS} = \{\mathsf{OS}_1, \mathsf{OS}_2\}$, where $\mathsf{OS}_1$ offers 350 syscalls (reflecting Linux [1]) and $\mathsf{OS}_2$ offers 1,200 syscalls (reflecting Windows [2]). For the library layer, suppose there are 10 libraries for $\mathsf{OS}_1$, including the standard library with 2,000 functions (2,000 is the number of standard libc functions in Linux [4]) and the 9 other libraries with each having 200 functions. Suppose there are 20 libraries for $\mathsf{OS}_2$, including the standard library with 5,000 functions (5,000 is an approximation of the standard library functions in Windows [97]) and the 19 other libraries with each having 300 functions.

For the dependence relation in a computer, namely $E_i = E_{i,al} \cup E_{i,ll} \cup E_{i,lk} \cup E_{i,ak} \cup E_{i,kk}$, we note that $E_i$ depends on the specific software stack diversity configuration. We observe (i) precisely obtaining the $E_i$ of a given computer requires a substantial effort, and (ii) the representativeness of the given computer is always debatable. These observations suggest us to make the following simplifying assumptions.

- For $E_{i,al}$, we assume (i) the standard library is always called by each application, but each of the other libraries is called by each application with a $50\%$ probability; and (ii), if a library is called by an application, each function of the library is called by the application with a probability of $5\%$.
- For $E_{i,ll}$, we assume each standard library function is called by each function in the other libraries with a probability of $5\%$.
- For $E_{i,lk}$, we assume each operating system function is called by each standard library function with a probability of $5\%$ and is called by each function in the other libraries with a probability of $1\%$.
- We set $E_{i,ak} = \emptyset$ because most applications will make syscalls through some libraries, rather than making syscalls directly.
- For $E_{i,kk}$, we assume that each operating system function is called by other operating system functions with a probability of $5\%$.

For the inter-computer communication relation $E_0$, we make the following assumptions: a browser is allowed to communicate with the web server in the DMZ; an email client can connect to the email server in the DMZ to retrieve and send emails; the web server needs to communicate with the SQL server; the email clients need to communicate with each other in the enterprise network (i.e., sending emails to, and receiving emails from, each other); a P2P application needs to communicate with the other P2P applications within the same sub-network; any computer in subnet 1 can communicate with any computer in subnet 3, and any computer in subnet 2 can communicate with subnet 8; any inter-computer communication not specified above is not allowed (i.e., blocked when *tight* preventive defense is enforced).

For the internal-external communication relation $E_*$, we assume that a browser can access the web server outside of the network, the computers can exchange emails with the outside of the network, the P2P applications need to communicate with their peers outside of the enterprise network, the word processors can open text files received from the external network, and Internet-facing servers (i.e., web server, email server) can be accessed by external computers.

## Simulating Vulnerabilities

For each software belonging to $\mathsf{APP}^{(N)} \cup \mathsf{LIB}^{(N)} \cup \mathsf{OS}^{(N)}$, we use parameter $\zeta \in [0, 1]$ to represent the probability that the software contains a vulnerability and is therefore vulnerable. For a software belonging to $\mathsf{LIB}^{(N)} \cup \mathsf{OS}^{(N)}$, the vulnerability is located at one of its functions that is chosen uniformly at random, while noting that this matter is not relevant for $\mathsf{APP}^{(N)}$ because an application is treated as a whole. The attributes of vulnerability $\mathsf{vul} \in B$ is determined as follows. If $\mathsf{vul}$ is in a operating system function, then $\mathsf{priv}(\mathsf{vul})$=1; otherwise, $\mathsf{priv}(\mathsf{vul})$=0. We use parameter $\vartheta(\mathsf{vul})$ to represent the probability that $\mathsf{vul}$ can be exploited remotely, namely $\Pr(\mathsf{loc}(\mathsf{vul}) = 1)$. We use parameter $\tau(\mathsf{vul})$ to represent the probability that $\mathsf{vul}$ is zero-day, namely $\Pr(\mathsf{zd}(\mathsf{vul}) = 1)$.

For human factor vulnerabilities, we assume that any client computer $i$ is subject to social-engineering attacks, because it may get compromised when accessing a malicious web server or when attacked by spearfishing, namely $\psi(v) = 1$ for $v \in V_i$.

## Simulating Attacks

We consider an attacker outside of the network attempting to penetrate into the network and compromise as many computers as possible. Attacks proceed according to the strategy described in Fig. 5.3. All of the applications associated with $E_*$ can be initial compromise targets. Moreover, a compromised P2P client or email client may send a malicious message to another client to exploit the latter's vulnerability (if any). A word processor can be exploited to spread attacks by formulating malicious payload that will be sent through either the email or the P2P application. For a given software stack diversity configuration $C = (C_{(app)}, C_{(lib)}, C_{(os)})$ of the software stacks, we use parameter $\mathsf{cap}$ to represent the fraction of vulnerabilities the attacker can exploit, where $\mathsf{cap} = 1$ means the attacker can exploit every vulnerability. We use parameter $\omega$ to represent the fraction of initial compromise targets, where $\omega = 1$ means the attacker will initially compromise every node that is vulnerable.

**Simulating Defenses**

**Simulating static network diversity**. For simplicity, we assume that every software has the same number $N$ of independent implementations. Given $N$ independent implementations, the tuple of configuration functions $C = (C_{(app)}, C_{(lib)}, C_{(os)})$ assign a specific implementation of an application, library, or operating system to run at the corresponding layer of a computer. We will consider 5 kinds of configurations that will be compared against each other: (i) $C_0$: $N = 1$ (i.e., the monoculture case); (ii) $C_1$: The application, library, and operating system layers are also diversified with $N$ implementations; (iii) $C_2$: The application layer is diversified with $N$ implementations, but the other layers are monoculture; (iv) $C_3$: The library layer is diversified with $N$ implementations, but the other layers are monoculture; and (v) $C_4$: The operating system layer is diversified with $N$ implementations, but the other layers are monoculture.

**Simulating other defenses**. As shown in Fig. 5.4, the network uses Firewall 1 to separate the Internet from the network, uses Firewall 2 to separate the subnets from each other, and uses a NIPS to protect each subnet. We further assume that each computer runs a HIPS, which has a success probability $1 - \alpha$ in detecting and blocking social-engineering attacks against computer $i$ that (i.e., its user) has a human factor vulnerability, namely $\psi(v) = 1$ for $v \in V_i$ as mentioned above. For the web server and the email server in the DMZ, ports other than the specific service ports are all disabled. Firewall 2 allows the web server in the DMZ to communicate with the database server in the database zone, but block any other traffic from the DMZ to the other part of the network.

**Simulation Algorithm**

Algorithm 2 describes the simulation algorithm, which proceeds according to the attack strategy mentioned above. The input includes $G = (V, E)$, the software stack diversity configuration $C$, the attacker's capabilities, the description of vulnerabilities $B$, and the description of defense $D$. The simulation results are presented in the pca and pcos metrics and are averaged over 100 simulation runs.

**Algorithm 2** Simulation algorithm.

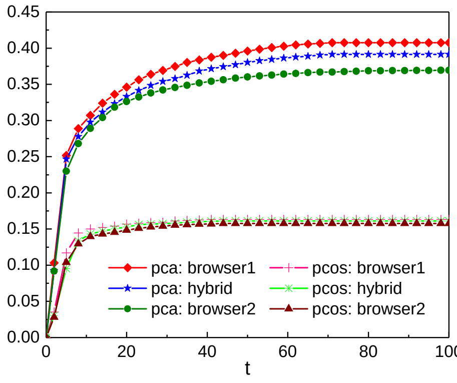
```
Input: $G = (V, E)$ with $\eta(v)$; $A = (X, \zeta, \omega, \mathsf{cap})$; $\mathsf{APP}^{(N)}$, $\mathsf{LIB}^{(N)}$, $\mathsf{OS}^{(N)}$; $B$ with $\zeta(\mathsf{vul}), \vartheta(\mathsf{vul}), \tau(\mathsf{vul})$ for $\mathsf{vul} \in B$; $C$; $D = (\alpha, \gamma,$HIPS,NIPS$)$ with HIPS,NIPS$\in \{tight, loose\}$; $T$
Output: $\mathsf{state}(v, t)$ for $v \in V$ and $t = 1, \ldots, T$
 1: Configure software stacks according to $C$
 2: Assign model parameters $\alpha$ to $v$, $\gamma$ to $e \in E$, HIPS to $V_i \in V$, NIPS to $e \in E$
 3: Simulate reconnaissance
 4: Compute Weapon according to Eq. (5.17)
 5: Select IniComp based on Weapon and $\omega$
 6: for $v \in V$ do
 7:     $\mathsf{state}(v, 0) = 0$
 8: end for
 9: for $v \in$ IniComp do
10:     Simulate initial compromise
11:     if $v$ is compromised then
12:         $\mathsf{state}(v, 1) = 1$
13:     end if
14: end for
15: for $t \in \{2, \ldots, T\}$ do
16:     for each app $\in V_{(app)}$ with $\mathsf{state}(v, t-1) = 1$ do
17:         Simulate privilege escalation and lateral movement
18:     end for
19: end for
20: Return $\mathsf{state}(v, t)$ for $v \in V$ and $t = 1, \ldots, T$
```

### 5.3.2 Simulation Results and Analysis

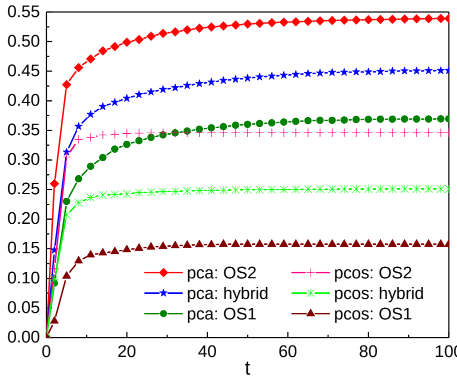
**RQ1: Does natural diversity always lead to higher security? If not, when?**

We consider natural diversity at both the application layer and the operating system layer, meaning $N = 1$ for every application and operating system. Moreover, we have $N = 1$ for every library.

First, we measure the security effectiveness of application-layer natural diversity by considering two browsers: $\mathsf{browser}_1$ and $\mathsf{browser}_2$ (as a simplified setting). We consider three scenarios: (i) each computer runs $\mathsf{browser}_1$; (ii) each computer runs $\mathsf{browser}_2$; (iii) the "hybrid" case in which each computer runs either $\mathsf{browser}_1$ or $\mathsf{browser}_2$ with probability 0.5. The other parameters are: the operating system is $\mathsf{OS}_1$, $\gamma = 0.2$ (failure probability of NIPS), $\alpha = 0.2$ (failure probability of HIPS), $\vartheta(\mathsf{vul}) = 0.8$ (the probability that $\mathsf{vul}$ can be exploited remotely), $\tau(\mathsf{vul}) = 0.05$ (the probability that $\mathsf{vul}$ is zero-day), $\zeta(v) = 0.2$ for $v \in V - \{\mathsf{browser}_1, \mathsf{browser}_2\}$ (the probability that the software running on node $v$ is vulnerable), $\mathsf{cap} = 1$ (the worst-case scenario that the attacker has exploit for every vulnerability), $\omega = 0.2$ (20% of the nodes in $\mathsf{Weapon}$ are initially compromised), NIPS = $tight$, and HIPS = $tight$.



(a) Browser natural diversity



(b) OS natural diversity

**Figure 5.5**: Plots of $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$ with natural diversity.

Fig. 5.5(a) plots $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$ with browser natural diversity, with $\zeta(\mathsf{browser}_1) = 0.4$ (the probability that $\mathsf{browser}_1$ is vulnerable) and $\zeta(\mathsf{browser}_2) = 0.2$ (the probability that $\mathsf{browser}_2$ is vulnerable). We observe that a higher (lower) browser vulnerability probability $\zeta$ leads to a higher (lower) $\mathsf{pca}(t)$, and the hybrid of them leads to a $\mathsf{pca}(t)$ somewhere in-between them. On the other hand, $\mathsf{pcos}(t)$ is not affected because the underlying operating system is the same. In addition, the percentage of compromised applications, namely $\mathsf{pca}(100)$, is always greater than the application vulnerable probability 0.2. This is beause an application can be compromised by exploiting a vulnerability in the application, by exploiting a vulnerability in the libraries the application invokes, or by compromising the operating system underlying it. On the other hand, the percentage of compromised operating systems, $\mathsf{pcos}(100)$, is always lower than the operating system vulnerable probability $0.2$. This is because some vulnerabilities cannot be reached, and therefore cannot be exploited, by the attacker; this can happen when the HIPS enforces the $tight$ policy.

Second, we measure security effectiveness of operating system-layer natural diversity by considering the case of two operating systems: $\mathsf{OS}_1$ and $\mathsf{OS}_2$ (as a simplified setting to demonstrate the competition between various versions of Unix and Windows). We consider three scenarios: (i) every computer runs $\mathsf{OS}_1$; (ii) every computer runs $\mathsf{OS}_2$; (iii) the "hybrid" case in where every computer runs either $\mathsf{OS}_1$ or $\mathsf{OS}_2$ with probability 0.5. The other parameters are: browser being the $\mathsf{browser}_1$ mentioned above, software stack diversity configuration $C_1$ with $N = 1$ (which is equivalent to $C_0$ with two operating systems), $\gamma = 0.2$, $\alpha = 0.2$, $\vartheta(\mathsf{vul}) = 0.8$, $\tau(\mathsf{vul}) = 0.05$, $\zeta(v) = 0.2$ for $v \in V - \{\mathsf{OS}_1, \mathsf{OS}_2\}$, $\mathsf{cap} = 1$, NIPS= $tight$, and HIPS= $tight$.

Fig. 5.5(b) plots $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$ with operating system natural diversity, with $\zeta(\mathsf{OS}_1) = 0.2$ (the probability that $\mathsf{OS}_1$ is vulnerable) and $\zeta(\mathsf{OS}_2) = 0.4$ (the probability that $\mathsf{OS}_2$ is vulnerable). We observe that a higher (lower) operating system vulnerability probability $\zeta$ leads to a higher (lower) $\mathsf{pcos}(t)$, and the hybrid of them leads to a $\mathsf{pcos}(t)$ somewhere in between them. When compared with the browser vulnerability probability, the operating system vulnerability probability has a more significant impact on $\mathsf{pca}(t)$ because a compromised operating system causes the compromise of any application running on a computer.

Summarizing the preceding discussion, we observe that when market competition leads to the emergence of a lower quality of software, security is degraded. For example, the emergence and deployment of $\mathsf{OS}_2$ with $\zeta = 0.4$ causes $\mathsf{pca}(100)$ increases from 0.3694 to 0.4508 in the hybrid case, meaning a $22.04\%$ security degradation. However, if market competition leads to higher quality of software, security can be improved. For example, the emergence and deployment of $\mathsf{browser}_2$ with $\zeta = 0.2$ causes $\mathsf{pca}(100)$ decreases from 0.4076 to 0.3893 in the hybrid case, meaning a $4.49\%$ security improvement.

**Insight 11.** *Natural diversity can lead to higher security as long as the diversified software implementations have a higher security quality (i.e., containing fewer software vulnerabilities); otherwise, natural diversity can lead to lower security.*

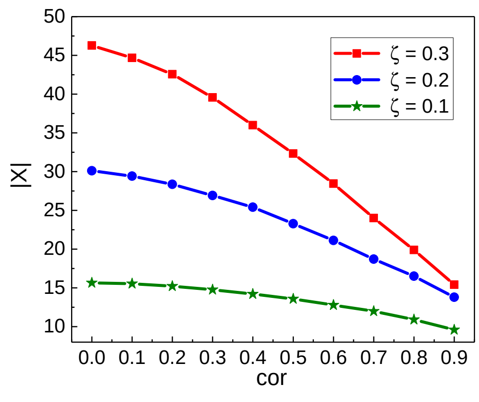
**RQ2: Does artificial diversity always lead to higher security? If not, when?**

In order to answer RQ2, we investigate a range of related sub-questions, including the impact of the dependence of vulnerabilities between diversified implementations. Studies [40, 71] have showed that the independence assumption in N-version programming is questionable because programmers tend to make the same mistakes (for example, incorrect treatment of boundary conditions). Therefore, we need to accommodate dependence between vulnerabilities. In order to describe dependence, we define the following *vulnerability correlation* metric.

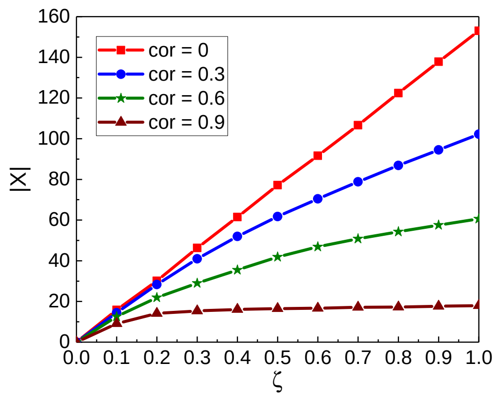
**Definition 7** (vulnerability correlation metric)**.** *Let $M$ denote the number of* independent *vulnerabilities in the $N$ implementations of a software program, where each vulnerability requires a different exploit. The vulnerability correlation metric, denoted by* $\mathsf{cor}$, *is defined* $\mathsf{cor} = 1 - M/N$, *where* $\mathsf{cor} = 0$ *corresponds to the extreme case that all of the $N$ vulnerabilities are independent (i.e., requiring $N$ exploits), and* $\mathsf{cor} = (N-1)/N$ *corresponds to the other extreme case that all of the $N$ vulnerabilities can be exploited by a single exploit.*

Note that Definition 7 implicitly assumes that each software program has at most one vulnerability. While this may be true in many cases, the definition can be extended to accommodate the more general case of a software program contains multiple vulnerabilities.

**Impact of vulnerability correlation** $\mathsf{cor}$ **on attacker effort**. In order to see the effect of vulnerability correlation $\mathsf{cor}$, we conduct an experiment with $N = 10$ (i.e., each software has 10



(a) |X| vs. cor



(b) |X| vs. $\zeta$

**Figure 5.6**: Plots of attacker's effort |X| with respect to fixed $\zeta$ and cor.

implementations), the operating system is $\mathsf{OS}_1$, and a fixed $\zeta$ (the probability that a software is vulnerable).

Fig. 5.6(a) shows that for a fixed $\zeta$, $|\mathsf{X}|$ (the number of exploits the attacker needs to obtain in order to compromise all of the vulnerable software) decreases as cor increases. Moreover, the decrease in $|\mathsf{X}|$ is nonlinear, and gets faster with a larger cor. This confirms that dependence between vulnerabilities will reduce the security effectiveness of network diversity in terms of the attacker's effort. Furthermore, a higher software vulnerability probability $\zeta$ leads to a more substantial reduction of the attacker's effort $|\mathsf{X}|$ when cor increases, implying that the attacker will benefit even more when the diversified implementations contain more vulnerabilities that are "correlated" with each other (i.e., lower quality). In terms of the attacker's effort with respect to a fixed vulnerability correlation cor, Fig. 5.6(b) shows that the attacker's effort $|\mathsf{X}|$ grows with the software vulnerability probability $\zeta$ (indicating an increasing number of vulnerabilities). However, the growth is nonlinear except in the case of cor $= 0$ (i.e., the vulnerabilities are independent of each other). The stronger the vulnerability correlation (e.g., cor $= 0.9$), the slower the increase to the attacker's effort.

**Insight 12.** *The independence assumption of vulnerabilities in diversified implementations does cause an overestimate of the security effectiveness of enforcing network diversity in terms of the*

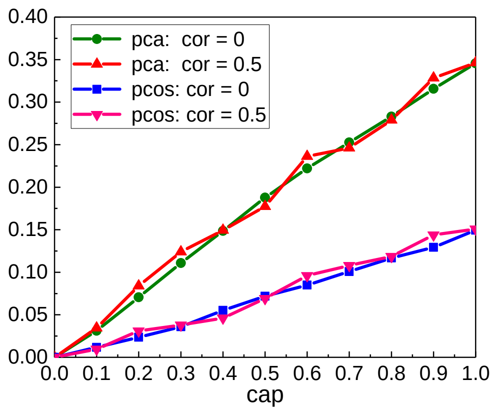


**Figure 5.7**: Plots of $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$.

*attacker's effort. The lower the security quality of the diversified implementations, the higher the benefit to the attacker, and the less useful the artificial diversity.*

**Impact of vulnerability correlation** $\mathsf{cor}$ **on** $\mathsf{pca}$ **and** $\mathsf{pcos}$. Fig. 5.7 plots $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$ with the increase of attacker capability $\mathsf{cap}$ (the fraction of vulnerabilities for which the attacker has exploits). For a fixed vulnerability probability $\zeta = 0.2$, we compare the security consequence of $\mathsf{cor} = 0$ and $\mathsf{cor} = 0.5$. The result shows that $\mathsf{pca}(100)$ with respect to $\mathsf{cor} = 0$ and $\mathsf{pca}(100)$ with respect to $\mathsf{cor} = 0.5$ are almost the same when $\mathsf{cap} = 1$ (the attacker having exploit for every vulnerability). The same phenomenon is exhibited by $\mathsf{pcos}(100)$ with respect to $\mathsf{cor} = 0$ and by $\mathsf{pcos}(100)$ with respect to $\mathsf{cor} = 0.5$. This means that vulnerability correlation $\mathsf{cor}$ has no effect on the security of the network against a powerful attacker because the attacker can exploit any vulnerability. On the other hand, $\mathsf{cor} = 0.5$ can lead to substantially higher damage in terms of $\mathsf{pca}$ and $\mathsf{pcos}$ when compared with the case of independent vulnerabilities $\mathsf{cor} = 0$.

**Insight 13.** *The independence assumption of vulnerabilities in diversified implementations does* not *cause an overestimate of the security effectiveness of enforcing network diversity in terms of metrics* $\mathsf{pca}$ *and* $\mathsf{pcos}$.

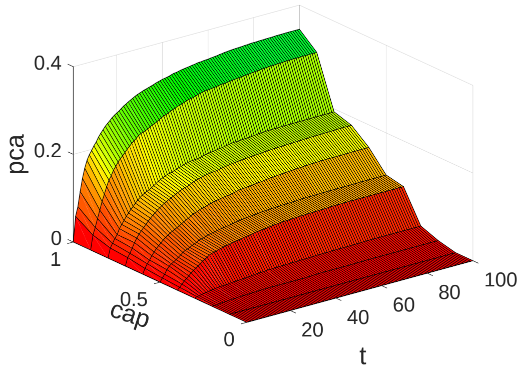
**When does artificial diversify lead to higher security?** We have observed that when the attacker can exploit *any* vulnerability, the vulnerability correlation $\mathsf{cor}$ has no effect on the security with regard to $\mathsf{pca}$ and $\mathsf{pcos}$. Therefore, we need to know when artificial diversity is useful. In order to answer this question, we consider $\mathsf{cor} = 0$ as a representative example scenario. We focus on $C_0$ and $C_1$ because $C_0$ in a sense reflects the state-of-the-art and $C_1$ reflects the ideal case. The other parameters are: $\mathsf{APP} = \{$browser, email client, P2P, word processor$\}$, operating system is $\mathsf{OS}_1$, $N = 10$, $\gamma = 0.2$, $\alpha = 0.2$, $\vartheta(\mathsf{vul}) = 0.8$, $\tau(\mathsf{vul}) = 0.05$, $\omega = 0.2$, NIPS= $tight$, HIPS= $tight$.

Fig. 5.8 plots pca and pcos with respect to attacker capability cap (the fraction of vulnerabilities that can be exploited by the attacker) and time. Figs. 5.8(a) and 5.8(c) show that monoculture, $C_0$, can lead to sudden "jumps" in terms of compromised applications and operating systems, namely that a single vulnerability can cause the compromise of many software. However, Figs. 5.8(b) and 5.8(d) show that the enforcement of network diversity, $C_1$, does not suffer from this problem. This shows that diversity can make the damage increases smoothly rather than abruptly, or make security degrades gradually rather than abruptly, with respect to increasing attack capabilities.

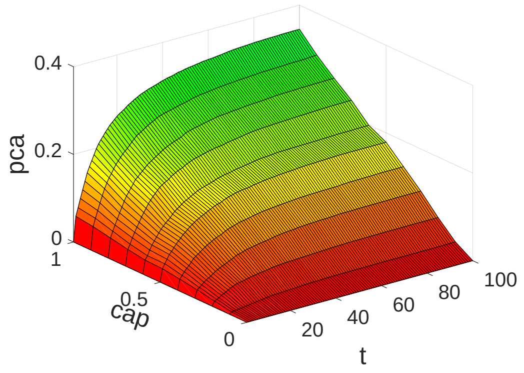
Suppose $\zeta$ is fixed, meaning that the security quality of independent implementations (in terms of their probabilities of being vulnerable) is the same. Suppose the attacker capability of the attacker cap, namely the number of exploits the attacker has, is proportional to the number of vulnerabilities. Figs. 5.8(a) and 5.8(b) show that $\mathsf{pca}(100)$ with respect to $C_0$ and $\mathsf{pca}(100)$ with respect to $C_1$ are almost the same, meaning that enforcing software diversity does not lead to better security. This is because increasing $N$ also increasing $N \times \zeta \times \mathsf{cap}$ proportionally for fixed $\zeta$ and cap. This example highlights when diversity neither increases nor decreases security.

By comparing Figs. 5.8(a) and 5.8(f), we observe that diversity indeed leads to higher security when the diversified software have fewer vulnerabilities than the monoculture case. Indeed, the security resulting from diversity is almost proportional to the improvement in software security quality, namely the improvement in terms of reducing $\zeta$ (the probability that a software is vulnerable) from 0.2 to 0.1. This example highlights when diversity leads to higher security.

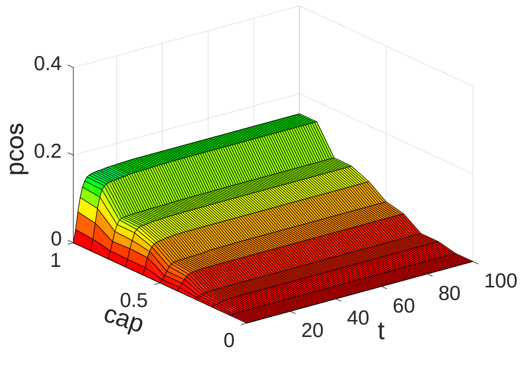
When comparing Figs. 5.8(b) and 5.8(e), we observe that diversity actually can lead to lower security when the security quality of diversified implementations is poor. Indeed, the security resulting from using diversified low-quality software is almost proportional to the security quality of the software. For example, considering $\mathsf{cap} = 1$ and $\mathsf{pca}(100)$, the damage of using low-



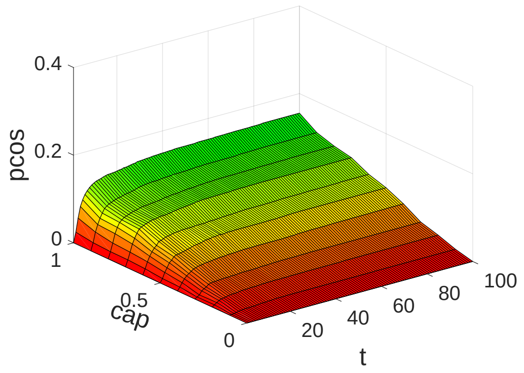
(a) Configuration $C_0$, $\zeta = 0.2$



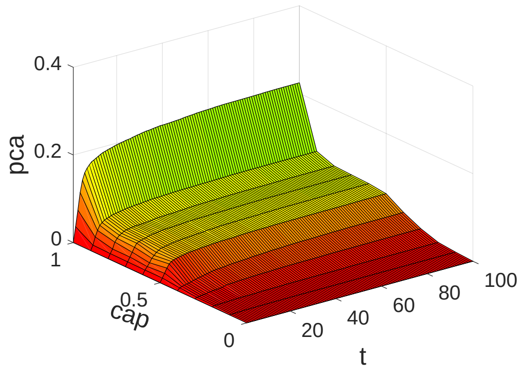
(b) Configuration $C_1$, $\zeta = 0.2$



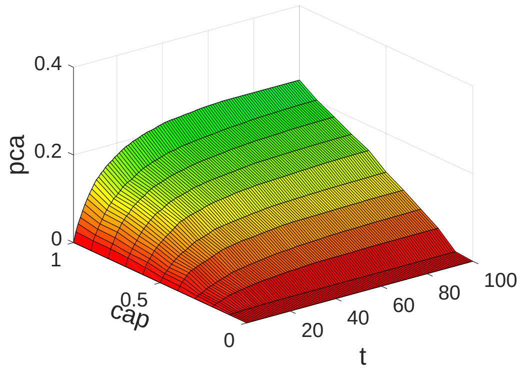
(c) Configuration $C_0$, $\zeta = 0.2$



(d) Configuration $C_1$, $\zeta = 0.2$



(e) Configuration $C_0$, $\zeta = 0.1$



(f) Configuration $C_1$, $\zeta = 0.1$

**Figure 5.8**: Plots of pca and pcos highlighting the smooth, rather than abrupt, decreases in security with increasing attacker capabilities.

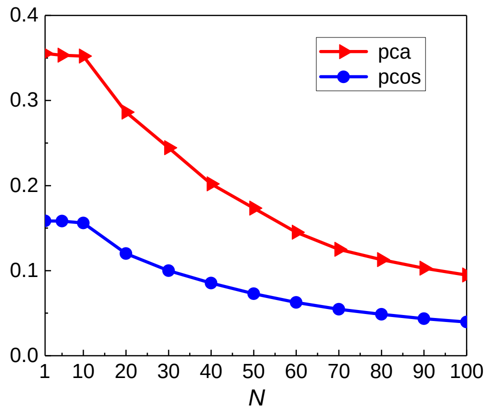


**Figure 5.9**: $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$.

quality diversified software ($\zeta = 0.2$) is 1.5 times of the damage of using high-quality monoculture ($\zeta = 0.1$). This highlights that diversity actually can lead to lower security when the diversified implementations are actually more vulnerable; this is possible because independently implementing multiple versions would incur a higher cost.

**Insight 14.** *Network diversity can lead to gradually (rather than abruptly) increasing damages when the attacker gets more powerful. However, the security effectiveness of network diversity largely depends on the security quality of the diversified implementations, meaning that diversity can increase, make no difference, or decrease security, depending on the relative quality between the diversified implementations and the monoculture implementation.*

**Is the resulting security linear to the degree of diversity $N$ when attack capability is fixed?** Suppose the attacker has a fixed number of exploits, say $|X| = 2$ for each software. Because enforcing diversity at all layers ($C_1$) may lead to a higher security, we now investigate the effect of $N$. The parameters are: $\gamma = 0.2$, $\alpha = 0.2$, $\vartheta(\mathsf{vul}) = 0.8$, $\tau(\mathsf{vul}) = 0.05$, $\zeta = 0.2$, $\mathsf{cap} = 0.2$, $\omega = 0.2$, NIPS= $tight$, HIPS= $tight$, and $t = 100$. Fig. 5.9 plots $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$ with varying $N$. We observe that when $N \leq 10$, increasing $N$ does not lead to any significantly better security in terms of the two metrics, because in this case each software has no more than $0.2 \times 10 = 2$ vulnerabilities among all of the implementations, which can all be exploited. When $N > 10$, increasing $N$ does lead to better security because the attacker has only 2 exploits or exploit 2 vulnerabilities, even though there are, for example when $N = 100$, $0.2 \times 100 = 20$ vulnerabilities in the diversified implementations. We further observe that security effectiveness, namely $1 - \mathsf{pca}(100)$ and $1 - \mathsf{pcos}(100)$, increases faster when $N \in [20, 60]$ than $N \in [60, 100]$. This manifests a kind of "diminishing return." Therefore, we have:

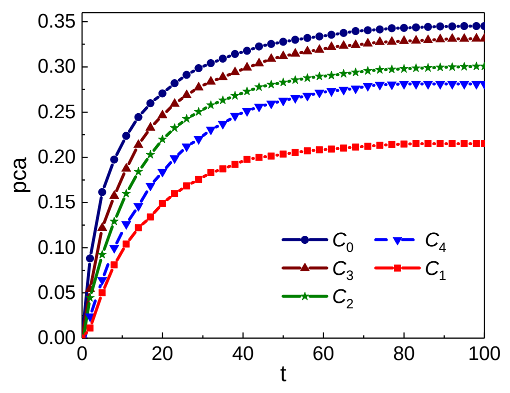
**Insight 15.** *Given a fixed attack capability, increasing $N$ (diversity effort) leads to a higher security only when some vulnerabilities cannot be exploited by the attacker. Moreover, there appears to be a "diminishing return" in security effectiveness, highlighting the importance of considering cost-effectiveness in achieving network diversity.*

**RQ3: How to prioritize static network diversity at the layers when diversity indeed improves security?**

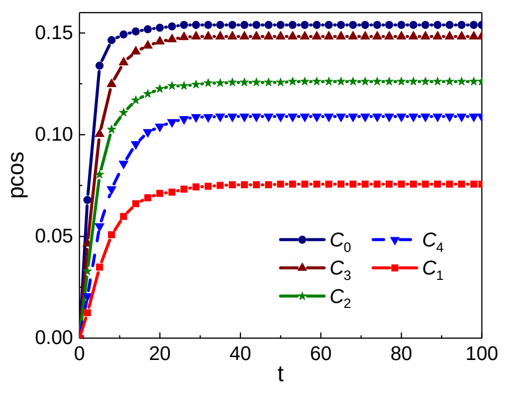
Recall that configuration $C_0$ means monoculture, $C_1$ means enforcing diversity at all three layers, and $C_2, C_3, C_4$ respectively means enforcing diversity at the application, library, and operating system layer. In order to compare their effectiveness, we consider the following parameters: $\mathsf{APP} = \{$browser, email client, P2P, word processor$\}$, operating system is $\mathsf{OS}_1$, $N = 10$ (in the case of $C_1, \ldots, C_4$), $\gamma = 0.2$, $\alpha = 0.2$, $\vartheta(\mathsf{vul}) = 0.8$, $\tau(\mathsf{vul}) = 0.05$, $\zeta(v) = 0.2$ for $v$ not enforcing diversity, $\zeta(v) = 0.1$ for $v$ enforcing diversity, $\mathsf{cap} = 1.0$, $\omega = 0.2$, NIPS = $tight$, and HIPS = $tight$.

Fig. 5.10 plots $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$ with different software stack configurations. We observe that for the same configuration and parameters, $\mathsf{pca}(t) > \mathsf{pcos}(t)$ for any $t$, meaning that there are more compromised applications than compromised operating systems. Moreover, we observe that both metrics $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$ show $C_0 \prec C_3 \prec C_2 \prec C_4 \prec C_1$, where $C_a \prec C_b$ means configuration $C_b$ leads to a higher security than $C_a$. For example, $\mathsf{pca}(100)$ for $C_0$ is 1.64 times of $\mathsf{pca}(100)$ for $C_1$, meaning that enforcing diversity at all three layers can reduce 39% of the damage when compared with the case of monoculture. This leads to:

**Insight 16.** *When diversity can improve security, enforcing diversity at multiple layers leads to higher security than enforcing diversity at a single layer. Enforcing diversity at the operating*



(a) $\mathsf{pca}(t)$



(b) $\mathsf{pcos}(t)$

**Figure 5.10**: $\mathsf{pca}(t)$ and $\mathsf{pcos}(t)$.

*system layer leads to higher security than enforcing diversity at the application layer, which leads to higher security than enforcing diversity at the library layer.*

**RQ4: Does the use of natural and artificial diversity together always lead to higher security? If not, when?**

In order to evaluate the security effectiveness of hybrid (i.e., natural and artificial) diversity, we consider a computer runs either $\mathsf{OS}_1$ or $\mathsf{OS}_2$. Suppose $\mathsf{OS}_1$ is diversified into $N_{\mathsf{OS}_1}$ implementations and $\mathsf{OS}_2$ is diversified into $N_{\mathsf{OS}_2}$ implementations. We consider three cases: (i) $N_{\mathsf{OS}_1} = 0$ and $N_{\mathsf{OS}_2} = 1$, meaning that all the computers run a monoculture operating system $\mathsf{OS}_2$; (ii) $N_{\mathsf{OS}_1} = 1$ and $N_{\mathsf{OS}_2} = 1$, meaning that each computer runs either $\mathsf{OS}_1$ or $\mathsf{OS}_2$ with probability 0.5, which corresponds to natural diversity; (iii) $N_{\mathsf{OS}_1} = 10$ and $N_{\mathsf{OS}_2} = 10$, meaning that each computer runs either $\mathsf{OS}_1$ or $\mathsf{OS}_2$ with probability 0.5, but both $\mathsf{OS}_1$ and $\mathsf{OS}_2$ have 10 independent implementations to choose, which corresponds to using hybrid (i.e., natural and artificial) diversity. The other parameters are: $\mathsf{APP} = \{$browser, email client, P2P, word processor$\}$, $C_1$ (every layer is artificially diversified), $\gamma = 0.2$ (the probability that attacks are not blocked by NIPS), $\alpha = 0.2$ (the probability that attacks that are not blocked by HIPS), $\vartheta(\mathsf{vul}) = 0.8$ (the probability that a vulnerability can be exploited remotely), $\tau(\mathsf{vul}) = 0.05$ (the probability that a vulnerability is zero-day), $\zeta(\mathsf{OS}_2) = 0.4$, $\zeta(\mathsf{OS}_1) = 0.2$, $\zeta(v) = 0.2$ for $v \in V - \{\mathsf{OS}_1, \mathsf{OS}_2\}$, $\mathsf{cap} = 1$ (the attacker can

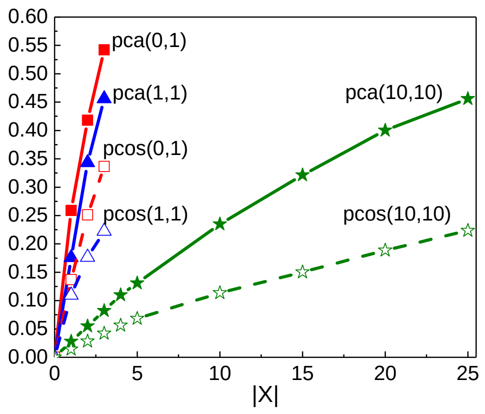


**Figure 5.11**: Plot of $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$ with respect to $(N_{\mathsf{OS}_1}, N_{\mathsf{OS}_2})$.

exploit every vulnerability, which corresponds to the worst-case scenario), $\omega = 0.2$ (20% of the nodes of Weapon are initially compromised), NIPS= $tight$, and HIPS= $tight$.

Fig. 5.11 plots $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$ with respect to $(N_{\mathsf{OS}_1}, N_{\mathsf{OS}_2})$. First, when comparing $\mathsf{pca}(0,1)$ and $\mathsf{pca}(1,1)$, we observe that natural diversity can lead to higher security once a higher quality of software is introduced, but the attacker's effort (the number of exploits $|\mathsf{X}|$) to compromise all of the vulnerable softwares remains the same. Second, by comparing $\mathsf{pca}(1,1)$ and $\mathsf{pca}(10,10)$, we observe that artificial diversity has no impact on security against a powerful attacker, but the attacker needs to pay almost 10 times the effort (or cost) to obtain exploits. On the other hand, if the attacker's number of exploits, $|\mathsf{X}|$, is fixed, artificial diversity can lead to a higher security. Third, by comparing $\mathsf{pca}(0,1)$ and $\mathsf{pca}(10,10)$, we observe that a comprehensive natural and artificial diversity not only can lead to a higher security, but also can substantially enhance the attacker's effort if diversified software implementations have higher security quality (i.e., less vulnerable). The same observations can be drawn from $\mathsf{pcos}(100)$. This leads to:

**Insight 17.** *It is beneficial to use both natural and artificial diversity (or "diversifying the diversity methods"), assuming that the diversified implementations have at least the same quality.*

**RQ5: What are the most effective defense strategies in the presence of network diversity?**

As mentioned above, an important research goal is to obtain Eq. (2.1), namely $m_i = \mathcal{F}_i(G, A, B, C, D)$. Due to the lack of real data, we use, as a first step, *linear regression* to extract Eq. (2.1) from the simulation data with respect to $t = 100$ (i.e., the empirical steady state) and both natural and artificial diversities. As a result, we can quantify the influence of each factor on $\mathsf{pca}$ and $\mathsf{pcos}$ and prioritize the factors that should be paid most attention in improving security.

For regression, we use the 14 explanatory variables $x_1, \ldots, x_{14}$ that are listed and explained in Table 5.2, except for $x_5$ which is determined as follows: $C_0, C_1, C_2, C_3, C_4$ respectively corresponds to $x_5 = 5, 1, 3, 4, 2$. This can be justified by the decreasing order of $\mathsf{pcos}(100)$ as shown in Fig. 5.10b, namely $C_0, C_3, C_2, C_4, C_1$. For the explanatory variables that are not defined over $[0, 1]$, we first standardize them into $[0, 1]$. The data consists of 568 rows of these variables as well as the corresponding $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$. Because 7 (out of the 14) variables are highly correlated with each other (e.g., the Pearson correlation coefficient between $x_5$ and $x_9$ is -0.3148, the coefficient between $x_6$ and $x_{10}$ is 0.4034, the coefficient between $x_{14}$ and $x_8$ is -0.45) and the amount of data is relatively small when compared with the number of variables (i.e., 14), we use the *partial least squares* method [129] to estimate the regression coefficients. Since it should be the case that $\mathsf{pca}(100) = \mathsf{pcos}(100) = 0$ when $x_1 = \ldots = x_{14} = 0$, the regression results are:

$$\mathsf{pca} = \sum_{i=1}^{14} a_i x_i \ \text{ and } \ \mathsf{pcos} = \sum_{i=1}^{14} b_i x_i, \tag{5.27}$$

where the $a_i$'s and $b_i$'s are respectively given in the 3rd and 4th column of Table 5.2. The cumulative R-square of $\mathsf{pca}(100)$ and $\mathsf{pcos}(100)$ in the fitted models is respectively 0.78 and 0.75, which means that the model fitting is accurate.

Table 5.2 shows the following. On one hand, the factors that have a significant influence on $\mathsf{pca}$ are $x_4, x_6, x_7, x_9, x_{10}$. The most significant factor is $x_9$, namely the fraction of vulnerabilities that can be exploited by the attacker. This suggests that the most significant strategy is to reduce the fraction of vulnerabilities that can be exploited by the attacker. On the other hand, the factors that

| parameter | meaning | pca | pcos |
|---|---|---|---|
| | | $a_i$ | $b_i$ |
| $x_1$ | fraction of $\mathsf{OS}_1$ in network | -0.075043802 | -0.087834035 |
| $x_2$ | # apps running on a computer | 0.088057385 | 0.101100359 |
| $x_3$ | fraction of $\mathsf{browser}_2$ in network | 0.030891731 | 0.021767401 |
| $x_4$ | $N$ | -0.135383668 | -0.132484815 |
| $x_5$ | (see text) | 0.052477615 | 0.06748185 |
| $x_6$ | $\zeta \in [0,1]$ | 0.134057115 | 0.191437005 |
| $x_7$ | $\vartheta \in [0,1]$ | 0.329776965 | 0.195279456 |
| $x_8$ | $\tau \in [0,1]$ | 0.034082221 | -0.011789502 |
| $x_9$ | $\mathsf{cap} \in [0,1]$ | 0.730355334 | 0.593032 |
| $x_{10}$ | $\omega \in [0,1]$ | 0.170030442 | 0.087308213 |
| $x_{11}$ | $\alpha \in [0,1]$ | 0.050954156 | 0.05991876 |
| $x_{12}$ | $\gamma \in [0,1]$ | 0.0001 | 0.0001 |
| $x_{13}$ | NIPS=1 (i.e., *tight*) | -0.049658855 | - 0.035677035 |
| $x_{14}$ | HIPS=1 (i.e., *tight*) | -0.037981485 | -0.310823305 |

**Table 5.2**: Description of the explanatory variables and the dependent variables pca and pcos.

have a significant influence on pcos are $x_2, x_4, x_6, x_7, x_9, x_{14}$. The most significant factor is $x_{14}$, namely the tightness of HIPS. This suggests that the most significant strategy to improve operating system-layer security is to enforce tight HIPS, namely to preventing unauthorized applications, even if compromised, from waging privilege escalation attacks.

**Insight 18.** *The most significant defense strategy is to reduce software vulnerabilities. The second most significant defense strategy is to enforce tight HIPS.*

## 5.4 Chapter Summary

In this Chapter, we proposed a fine-grained theoretical framework to model network diversity, including a suite of security metrics for measuring attacker's effort, defender's effort, and security effectiveness of network diversity. We considered both natural and artificial diversities. We conducted simulation experiments to measure these metrics and draw insights from the experimental results. We characterized the conditions under which software diversity can lead to higher or lower security, or make no difference.

# CHAPTER 6: CONCLUSION

## 6.1 Summary of the Dissertation

This PhD Dissertation makes a significant step towards quantifying cybersecurity from a holistic perspective, by following the Cybersecurity Dynamics approach and focusing on specific cyber defense mechanisms. In order to achieve the research goals, we propose a systematic time-dependent framework to model cyber attack-defense interactions from a whole-network perspective. The framework contains a suite of security metrics for measuring the effectiveness of cyber attacks and defenses. To demonstrate the usefulness of the proposed framework, we investigate three application scenarios: quantifying the security effectiveness of firewalls and DMZ, quantifying the security effectiveness of dynamic network diversity, and quantifying the security effectiveness of static network diversity. We design and implement simulations to mimic real-world attack-defense interactions. We draw a number of insights into the security effectiveness of these defense mechanisms and architecture.

In terms of the security effectiveness of firewalls and DMZ, the study leads to the following insights:

- When operating systems are not vulnerable, the security effectiveness of a fixed combination of firewall and DMZ decreases as the fraction of vulnerable applications increases.
- Firewall and DMZ are not effective when few or most computers are vulnerable.
- Employing the perimeter firewall lone has a little security impact, but a comprehensive use of firewall and DMZ can substantially increases security.
- Employing perimeter firewall and DMZ can substantially increase the security of sever applications.

In terms of the security effectiveness of dynamic network diversity, the study leads to the following insights:

- In terms of attacker slow-down, reactive-adaptive diversity is the most effective strategy and the initial diversity configuration matters.
- In order to reduce the attack worst damage, different diversity strategies should be used in different parameter regimes.
- Reactive-adaptive diversity leads to a higher vulnerability-tolerance than proactive diversity does.
- Proactive diversity improves security only when dynamic diversity is widely re-employed at a high frequency, which however incurs a high operational cost.

In terms of the security effectiveness of static network diversity, the study leads to the following insights:

- Static network diversity does *not necessarily always* improve security from a whole-network perspective, because the security effectiveness of network diversity largely depends on the security quality of the diversified implementations.
- The *independence* assumption of vulnerabilities in the diversified implementations does cause an overestimate of security effectiveness in terms of the attacker's effort.
- Given a fixed attack capability, increasing diversity effort can lead to a higher security as long as there are always some vulnerabilities that cannot be exploited by the attacker.
- When diversity can improve security, enforcing diversity at multiple layers leads to higher security than enforcing diversity at a single layer.

## 6.2 Future Research Directions

### 6.2.1 Future Research Related to Quantifying Security Effectiveness of Firewalls and DMZs

We identify the following limitations of the study described in Chapter 3. These limitations need to be addressed in future investigations. First, it is important to extend the simulation study to consider broader parameter regimes (e.g., the case of running multiple kinds of OS in enterprise networks). Second, it is important to conduct case study to derive the structure $G = (V, E)$ of real-world enterprise networks. Third, It is important to validate the proposed framework using real-world datasets. Fourth, it is important to examine more hostile scenarios in which firewalls can be compromised.

### 6.2.2 Future Research Related to Quantifying Security Effectiveness of Coarse-Grained Dynamic Network Diversity

We identify the following limitations of the study described in Chapter 4. These limitations need to be addressed in future investigations.

First, the framework has two limitations. (i) We assume $G$ is fixed. This implicitly assumes the network defense tools are not compromised because a successful attack against a defense tool can effectively change $G$. While this is reasonable for missions with a short lifetime $T$, it is interesting to accommodate dynamic $G_t$ for missions of long lifetime $T$ and the case that the network defense tools can be compromised, as outlined in [148, 149, 152]. (ii) We assume that the attacker selects one tool to use at each phase of an attack strategy. This can be extended to using multiple tools in a sequential manner.

Second, the simulation study has some limitations. (i) We only consider simple decision-making algorithms, which are sufficient for demonstrating the usefulness of the framework but need to accommodate more sophisticated decision-making algorithms. (ii) We use some "synthetic" and simplifying scenarios owing to the lack of real data, meaning that the findings may not be generalized to other scenarios. Specifically, the assumption that each attack phase takes place at one time step may limit the validity of Insight 5; the assumption that each exploit incurs the same cost to the attacker may limit the validity of Insight 6; the assumption that each implementation of a program is equally vulnerable may limit the validity of Insight 7; the independence assumption that different implementations do not have common vulnerabilities holds in some settings [47, 54] but may not hold in general. In order to see the potential impact of these assumptions, we conduct additional experiments where different implementations can contain common vulnerabilities. While omitting the experimental details owing to space limit, they do show that the independence assumption does cause overestimates of the effectiveness of employing network diversity. This resonates the results of earlier theoretical studies [36, 146, 147].

### 6.2.3 Future Research Related to Quantifying Security Effectiveness of Fine-Grained Static Network Diversity

We identify the following limitations of the study described in Chapter 5. These limitations need to be addressed in future investigations. First, the framework does not consider insider threats. Second, the framework focuses mainly on preventive defenses. Future research will include the investigation of a broader spectrum of defense mechanisms including reactive and adaptive defenses. Third, the simulation study assumes that firewalls cannot be compromised. This assumption can be eliminated, by accommodating the consequence of compromised firewalls (e.g., the network-based *tight* preventive defense enforced by a compromised firewall needs to become a *loose* preventive defense). Nevertheless, we already considered the security effectiveness of network-based *loose* preventive defense, which corresponds to the worst-case scenario in which all of the firewalls are compromised. It is worth mentioning that this issue is already resolved for host-based preventive defense, which is enforced by an operating system, because the compromise of an operating system already causes the compromise of the entire computer. Fourth, the $G = (V, E)$ used in the simulation study is heuristically generated, rather than derived from real-world network software stacks.

### 6.2.4 Future Research Towards the Ultimate Goal

The present Dissertation focuses on some small components in the Cybersecurity Dynamics framework mentioned in Chapter 2, namely quantifying security effectiveness of cyber defenses in some relatively general, but not general enough, settings. There are tons of Open Problems that yet to be tackled, such as tackling the challenges of cybersecurity metrics and quantification as described in [32, 76, 106, 154] and overcoming the technical barriers as described in [148, 149, 152, 153].

Another important research direction is to quantify the attack capabilities of sophisticated cyber attacks, such as adversarial malicious websites [141, 142] and adversarial malware examples [18, 25, 74, 75, 78, 135, 158].

Yet another important problem is to leverage the high-fidelity simulation framework presented in this Dissertation to generate realistic datasets for various cybersecurity research purposes. One particularly important scenario is to leverage such data to achieve cyber threats forecasting, which has been somewhat systematically investigated in a sequence of studies [27, 43, 44, 107, 108, 143, 145, 161–163] but much more research needs to be done.

I hope the present study will inspire many more PhD Dissertations in the near future.

# BIBLIOGRAPHY

[1] https://syscalls.kernelgrok.com/.

[2] http://j00ru.vexillium.org/syscalls/nt/64/.

[3] Att&ck matrix for enterprise. https://attack.mitre.org/.

[4] The c library. https://www.gnu.org/software/libc/manual/html_node/Function-Index.html/.

[5] A new defense for navy ships: Protection from cyber attacks. https://www.onr.navy.mil/en/Media-Center/Press-Releases/2015/RHIMES-Cyber-Attack-Protection.aspx.

[6] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba, and Masum Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE journal on selected areas in communications*, 23(10):2069–2084, 2005.

[7] Massimiliano Albanese, Sushil Jajodia, and Steven Noel. Time-efficient and cost-effective network hardening using attack graphs. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE, 2012.

[8] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[9] Andy Applebaum, Doug Miller, Blake Strom, Chris Korban, and Ross Wolf. Intelligent, automated red team emulation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 363–373, 2016.

[10] Algirdas Avizienis. The n-version approach to fault-tolerant software. *IEEE Transactions on software engineering*, (12):1491–1501, 1985.

[11] Elena Gabriela Barrantes, David H Ackley, Trek S Palmer, Darko Stefanovic, and Dino Dai Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 281–289. ACM, 2003.

[12] Matthew P Barrett. Framework for improving critical infrastructure cybersecurity. *National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep*, 2018.

[13] Benoit Baudry and Martin Monperrus. The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Computing Surveys (CSUR)*, 48(1):1–26, 2015.

[14] Sandeep Bhatkar, Daniel C DuVarney, and Ron Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *USENIX Security Symposium*, volume 12, pages 291–301, 2003.

[15] Daniel Borbor, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Diversifying network services under cost constraints for better resilience against unknown attacks. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 295–312. Springer, 2016.

[16] John Boyd. The essence of winning and losing, 28 June 1995.

[17] Hasan Cam. *Controllability and Observability of Risk and Resilience in Cyber-Physical Cloud Systems*, pages 325–343. Springer New York, New York, NY, 2014.

[18] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57, 2017.

[19] T. Carroll, M. Crouse, E. Fulp, and K. Berenhaut. Analysis of network address shuffling as a moving target defense. In *Proc. ICC'14*, pages 701–706. IEEE, 2014.

[20] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–26, 2008.

[21] Geoff Chappell. kernel-mode windows. `https://www.geoffchappell.com/studies/windows/km/index.htm?tx=10`.

[22] Huashan Chen, Hasan Cam, and Shouhuai Xu. Quantifying cybersecurity effectiveness of dynamic network diversity. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[23] Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. Quantifying the security effectiveness of firewalls and dmzs. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, pages 1–11, 2018.

[24] Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. Quantifying the security effectiveness of network diversity: poster. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security, HoTSoS 2018, Raleigh, North Carolina, USA, April 10-11, 2018*, page 24:1. ACM, 2018.

[25] L. Chen, S. Hou, Y. Ye, and S. Xu. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In *Proc. 2018 IEEE/ACM ASONAM*, pages 782–789, 2018.

[26] Liming Chen. N-version programming: A fault-tolerant approach to reliability of software operation. In *Proc. International Symposium on Fault Tolerant Computing*, pages 3–9, 1978.

[27] Y. Chen, Z. Huang, S. Xu, and Y. Lai. Spatiotemporal patterns and predictability of cyberattacks. *PLoS One*, 10(5):e0124472, 05 2015.

[28] Yi Cheng, Julia Deng, Jason Li, ScottA. DeLoach, Anoop Singhal, and Xinming Ou. Metrics of security. In *Cyber Defense and Situational Awareness*, volume 62, pages 263–295. 2014.

[29] Monica Chew and Dawn Song. Mitigating buffer overflows by operating system randomization. 2002.

[30] J. Cho, P. Hurley, and S. Xu. Metrics and measurement of trustworthy systems. In *Proc. IEEE MILCOM*, 2016.

[31] Jin-Hee Cho, Dilli P Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J Moore, Dong Seong Kim, Hyuk Lim, and Frederica F Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22(1):709–745, 2020.

[32] Jin-Hee Cho, Shouhuai Xu, Patrick M. Hurley, Matthew Mackay, Trevor Benjamin, and Mark Beaumont. Stram: Measuring the trustworthiness of computer-based systems. *ACM Comput. Surv.*, 51(6):128:1–128:47, 2019.

[33] INFOSEC Research Council. Hard problem list. `http://www.infosec-research.org/docs_public/20051130-IRC-HPL-FINAL.pdf`, 2007.

[34] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. N-variant systems: A secretless framework for security through diversity. In *USENIX Security Symposium*, pages 105–120, 2006.

[35] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*, 2015.

[36] Gaofeng Da, Maochao Xu, and Shouhuai Xu. A new approach to modeling and analyzing security of networked systems. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, pages 1–12, 2014.

[37] Lucas Davi, Christopher Liebchen, Ahmad-Reza Sadeghi, Kevin Z Snow, and Fabian Monrose. Isomeron: Code randomization resilient to (just-in-time) return-oriented programming. In *NDSS*, 2015.

[38] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proceedings*

*of the 14th ACM conference on Computer and communications security*, pages 204–213. ACM, 2007.

[39] P. Du, Z. Sun, H. Chen, J. H. Cho, and S. Xu. Statistical estimation of malware detection metrics in the absence of ground truth. *IEEE Transactions on Information Forensics and Security*, 13(12):2965–2980, 2018.

[40] Dave E Eckhardt, Alper K. Caglayan, John C. Knight, Larry D. Lee, David F. McAllister, Mladen A. Vouk, and John P. J. Kelly. An experimental evaluation of software redundancy as a strategy for improving reliability. *IEEE Transactions on software engineering*, 17(7):692–702, 1991.

[41] Hiroaki Etoh. Gcc extentions for protecting applications from stack-smashing attacks. *http://www. research. ibm. com/trl/projects/security/ssp/*, 2000.

[42] Isaac Evans, Sam Fingeret, Julian Gonzalez, Ulziibayar Otgonbaatar, Tiffany Tang, Howard Shrobe, Stelios Sidiroglou-Douskos, Martin Rinard, and Hamed Okhravi. Missing the point (er): On the effectiveness of code pointer integrity. In *2015 IEEE Symposium on Security and Privacy*, pages 781–796. IEEE, 2015.

[43] X. Fang, M. Xu, S. Xu, and P. Zhao. A deep learning framework for predicting cyber attacks rates. *EURASIP J. Information Security*, 2019:5, 2019.

[44] Zijian Fang, Peng Zhao, Maochao Xu, Shouhuai Xu, Taizhong Hu, and Xing Fang. Statistical modeling of computer malware propagation dynamics in cyberspace. *Journal of Applied Statistics*, pages 1–26, 2020.

[45] Stephanie Forrest, Anil Somayaji, and David H Ackley. Building diverse computer systems. In *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, pages 67–72. IEEE, 1997.

[46] Michael Franz. From fine grained code diversity to JIT-ROP to execute-only memory: The cat and mouse game between attackers and defenders continues. In *Proceedings of the Second ACM Workshop on Moving Target Defense, MTD 2015*, page 1, 2015.

[47] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *Proc. IEEE/IFIP DSN*, pages 383–394, 2011.

[48] M. Garcia, A. Bessani, and N. Neves. Lazarus: Automatic management of diversity in bft systems. In *Proc. Middleware*, pages 241–254, 2019.

[49] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Softw. Pract. Exper.*, 44(6):735–770, June 2014.

[50] Richard Garcia-Lebron, David J Myers, Shouhuai Xu, and Jie Sun. Node diversification in complex networks by decentralized colouring. *Journal of Complex Networks*, 7(4):554–563, 2019.

[51] Daniel Geer, Rebecca Bace, Peter Gutmann, Perry Metzger, Charles P. Pfleeger, John S. Quarterman, and Bruce Schneier. Cyberinsecurity: The cost of monopoly. `http://cryptome.org/cyberinsecurity.htm`, 27 September 2003.

[52] Anup K Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Workshop on Intrusion Detection and Network Monitoring*, volume 51462, pages 1–13, 1999.

[53] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. Aslr on the line: Practical cache attacks on the mmu. In *NDSS*, volume 17, page 13, 2017.

[54] Jin Han, Debin Gao, and Robert H Deng. On the effectiveness of software diversity: A systematic study on real-world vulnerabilities. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 127–146. Springer, 2009.

[55] Y. Han, W. Lu, and S. Xu. Preventive and reactive cyber defense dynamics with ergodic time-dependent parameters is globally attractive. *IEEE TNSE*, 8(3):2517–2532, 2021.

[56] Yujuan Han, Wenlian Lu, and Shouhuai Xu. Characterizing the power of moving target defense via cyber epidemic dynamics. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, pages 1–12, 2014.

[57] Miguel Garcia Tavares Henriques. Diverse intrusion-tolerant systems. 2019.

[58] Matti A Hiltunen, Richard D Schlichting, Carlos A Ugarte, and Gary T Wong. Survivability through customization and adaptability: The cactus approach. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 1, pages 294–307. IEEE, 2000.

[59] David A Holland, Ada T Lim, and Margo I Seltzer. An architecture a day keeps the hacker away. *ACM SIGARCH Computer Architecture News*, 33(1):34–41, 2005.

[60] John Homer, Su Zhang, Xinming Ou, David Schmidt, Yanhui Du, S Raj Rajagopalan, and Anoop Singhal. Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security*, 21(4):561–597, 2013.

[61] Andrei Homescu, Todd Jackson, Stephen Crane, Stefan Brunthaler, Per Larsen, and Michael Franz. Large-scale automated software diversity - program evolution redux. *IEEE Trans. Dependable Sec. Comput.*, 14(2):158–171, 2017.

[62] Chu Huang, Sencun Zhu, and Robert Erbacher. Toward software diversity in heterogeneous networked systems. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 114–129. Springer, 2014.

[63] Y. Huang and A. Ghosh. Introducing diversity and uncertainty to create moving attack surfaces for web services. In *Moving target defense*, pages 131–151. Springer, 2011.

[64] Ray Hunt. Internet/intranet firewall security-policy, architecture and transaction services. *Computer Communications*, 21(13):1107–1123, 1998.

[65] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[66] Todd Jackson, Babak Salamat, Andrei Homescu, Karthikeyan Manivannan, Gregor Wagner, Andreas Gal, Stefan Brunthaler, Christian Wimmer, and Michael Franz. Compiler-generated software diversity. In *Moving Target Defense*, pages 77–98. Springer, 2011.

[67] J. Jafarian, E. Al-Shaer, and Q. Duan. An effective address mutation approach for disrupting reconnaissance attacks. *IEEE TIFS*, 10(12):2562–2577, 2015.

[68] Somesh Jha, Oleg Sheyner, and Jeannette Wing. Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pages 49–63. IEEE, 2002.

[69] J. Just, J. Reynolds, L. Clough, M. Danforth, K. Levitt, R. Maglich, and J. Rowe. Learning unknown attacksâa start. In *Proc. RAID*, pages 158–176. Springer, 2002.

[70] Gaurav S Kc, Angelos D Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280. ACM, 2003.

[71] John C Knight and Nancy G Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on software engineering*, (1):96–109, 1986.

[72] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz. Sok: Automated software diversity. In *IEEE S& P*, pages 276–291, 2014.

[73] Per Larsen, Stefan Brunthaler, and Michael Franz. Security through diversity: Are we there yet? *IEEE Security & Privacy*, 12(2):28–35, 2014.

[74] D. Li, Q. Li, Y. Ye, and S. Xu. Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge. In *AAAI-2019 Workshop on Artificial Intelligence for Cyber Security (AICS'2019)*.

[75] D. Li, Q. Li, Y. Ye, and S. Xu. A frameowrk for enhancing deep neural networks against adversarial malware examples. *IEEE Transactions on Network Science and Engineering (TNSE)*, 8:736–750, 2021.

[76] D. Li, Q. Li, Y. Ye, and S. Xu. Sok: Arms race in adversarial malware detection. *Accepted to ACM Computing Survey*, 2021.

[77] D. Li, T. Qiu, S. Chen, Q. Li, and S. Xu. Can we leverage predictive uncertainty to detect dataset shift and adversarial examples in android malware detection? In *The 2021 Annual Computer Security Application Conference (ACSAC)*, 2021.

[78] Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. *arXiv preprint arXiv:1809.06498*, 2018.

[79] Xiaohu Li, Paul Parker, and Shouhuai Xu. A stochastic model for quantitative security analyses of networked systems. *IEEE Transactions on Dependable and Secure Computing*, 8(1):28–43, 2011.

[80] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu, and H. Jin. Vuldeelocator: A deep learning-based fine-grained vulnerability detector. *IEEE TDSC*, 2021.

[81] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu. Vulpecker: an automated vulnerability detection system based on code similarity analysis. In *Pro. ACSAC'16*, pages 201–213, 2016.

[82] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, S. Wang, and J. Wang. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing (accepted for publication)*, 2021.

[83] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Zhang, Z. Chen, and D. Li. Vuldeelocator: A deep learning-based system for detecting and locating software vulnerabilities. IEEE TDSC (accepted for publication), 2021.

[84] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. In *Proc. NDSS'18*, 2018.

[85] Zhen Li, Jing Tang, Deqing Zou, Qian Chen, Shouhuai Xu, Chao Zhang, Yichen Li, and Hai Jin. Robustness of deep learning-based vulnerability detectors: Attack anddefense. under review, 2021.

[86] Zongzong Lin, Wenlian Lu, and Shouhuai Xu. Unified preventive and reactive cyber defense dynamics is still globally convergent. *IEEE/ACM Trans. Netw.*, 27(3):1098–1111, 2019.

[87] Bev Littlewood, Sarah Brocklehurst, Norman Fenton, Peter Mellor, Stella Page, David Wright, John Dobson, John McDermid, and Dieter Gollmann. Towards operational measures of computer security. *Journal of computer security*, 2(2-3):211–229, 1993.

[88] WT Lord Kelvin. Electrical units of measurement. *Nature*, 28(708):91–92, 1883.

[89] Wenlian Lu, Shouhuai Xu, and Xinlei Yi. Optimizing active cyber defense dynamics. In *Proceedings of the 4th International Conference on Decision and Game Theory for Security (GameSec'13)*, pages 206–225, 2013.

[90] Y. Luo, B. Wang, and G. Cai. Effectiveness of port hopping as a moving target defense. In *Proc. ICCST*, pages 7–10. IEEE, 2014.

[91] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[92] Matteo Magnani and Luca Rossi. The ML-Model for Multi-layer Social Networks. In *ASONAM*, pages 5–12. IEEE Computer Society, 2011.

[93] Mateusz. Windows win32k.sys system call table. `http://j00ru.vexillium.org/syscalls/win32k/32/`.

[94] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 177–187. IEEE, 2000.

[95] Dan McWhorter. Apt1: exposing one of china's cyber espionage units. *Mandiant. com*, 18, 2013.

[96] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6):85–89, 2006.

[97] Microsoft. Msdn library. `https://msdn.microsoft.com/en-us/library/ms310241/`.

[98] J. Mireles, E. Ficke, J. Cho, P. Hurley, and S. Xu. Metrics towards measuring cyber agility. *IEEE T-IFS*, 14(12):3217–3232, 2019.

[99] H. Moniz, N. Neves, M. Correia, and P. Verissimo. Ritas: Services for randomized intrusion tolerance. *IEEE TDSC*, 8(1):122–136, 2008.

[100] Saran Neti, Anil Somayaji, and Michael E Locasto. Software diversity: Security, entropy and game theory. In *HotSec*, 2012.

[101] David Nicol, Bill Sanders, Jonathan Katz, Bill Scherlis, Tudor Dumitra, Laurie Williams, and Munindar P. Singh. The science of security 5 hard problems (august 2015). `http://cps-vo.org/node/21590`.

[102] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Trans. Dependable Sec. Comput.*, 1(1):48–65, 2004.

[103] Steven Noel and Sushil Jajodia. *A Suite of Metrics for Network Attack Graph Analytics*, pages 141–176. Springer International Publishing, Cham, 2017.

[104] Adam J. O'Donnell and Harish Sethu. On achieving software diversity for improved network security using distributed coloring algorithms. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 121–131, 2004.

[105] The Forum of Incident Response and Security Teams FIRST. The common vulnerability scoring system (CVSS), June 2015.

[106] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. A survey on systems security metrics. *ACM Computing Surveys (CSUR)*, 49(4):62, 2017.

[107] Chen Peng, Maochao Xu, Shouhuai Xu, and Taizhong Hu. Modeling and predicting extreme cyber attack rates via marked point processes. *Journal of Applied Statistics*, 44(14):2534–2563, 2017.

[108] Chen Peng, Maochao Xu, Shouhuai Xu, and Taizhong Hu. Modeling multivariate cybersecurity risks. *Journal of Applied Statistics*, 0(0):1–23, 2018.

[109] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[110] M. Platania, D. Obenshain, T. Tantillo, R. Sharma, and Y. Amir. Towards a practical survivable intrusion tolerant replication system. In *Proc. IEEE SRDS*, pages 242–252, 2014.

[111] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, 2012.

[112] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues. Model-based quantitative network security metrics: A survey. *IEEE Communications Surveys Tutorials*, 19(4):2704–2734, 2017.

[113] Alex Ramos, Marcella Lazar, Raimir Holanda Filho, and Joel JPC Rodrigues. Model-based quantitative network security metrics: A survey. *IEEE Communications Surveys & Tutorials*, 19(4):2704–2734, 2017.

[114] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 156–165. IEEE, 2000.

[115] R. Rodrigues, M. Castro, and B. Liskov. Base: Using abstraction to improve fault tolerance. *ACM SIGOPS Operating Systems Review*, 35(5):15–28, 2001.

[116] Tom Roeder and Fred B Schneider. Proactive obfuscation. *ACM Transactions on Computer Systems (TOCS)*, 28(2):1–54, 2010.

[117] Robert Rudd, Richard Skowyra, David Bigelow, Veer Dedhia, Thomas Hobson, Stephen Crane, Christopher Liebchen, Per Larsen, Lucas Davi, Michael Franz, Ahmad-Reza Sadeghi, and Hamed Okhravi. Address oblivious code reuse: On the effectiveness of leakage resilient diversity. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017*, 2017.

[118] Babak Salamat, Todd Jackson, Gregor Wagner, Christian Wimmer, and Michael Franz. Runtime defense against code injection attacks using replicated execution. *IEEE Transactions on Dependable and Secure Computing*, 8(4):588–601, 2011.

[119] National Science and Technology Council. Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program. `https://www.nitrd.gov/SUBCOMMITTEE/csia/Fed_Cybersecurity_RD_Strategic_Plan_2011.pdf`, 2011.

[120] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *Proc. CCS'2004*, pages 298–307, 2004.

[121] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[122] Deb Shinder. Solutionbase: Strengthen network defenses by using a dmz. `https://www.techrepublic.com/article/solutionbase-strengthen-network-defenses-by-using-a-dmz/`.

[123] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *2013 IEEE Symposium on Security and Privacy*, pages 574–588. IEEE, 2013.

[124] Ana Nora Sovarel, David Evans, and Nathanael Paul. Where's the feeb? the effectiveness of instruction set randomization. In *USENIX Security Symposium*, 2005.

[125] Mark Stamp. Risks of monoculture. *Commun. ACM*, 47(3):120–, March 2004.

[126] Raoul Strackx, Yves Younan, Pieter Philippaerts, Frank Piessens, Sven Lachmund, and Thomas Walter. Breaking the memory secrecy assumption. In *Proceedings of the Second European Workshop on System Security*, pages 1–8, 2009.

[127] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[128] Orcun Temizkan, Sungjune Park, and Cem Saydam. Software diversity for improved network security: optimal distribution of software-based shared vulnerabilities. *Information Systems Research*, 28(4):828–849, 2017.

[129] Randall D Tobias et al. An introduction to partial least squares regression. In *Proceedings of the twentieth annual SAS users group international conference*, pages 1250–1257. SAS Institute Cary, NC, 1995.

[130] E. Totel, F. Majorczyk, and L. Me. Cots diversity based intrusion detection and application to web servers. In *Proc. RAID*, pages 43–62. Springer, 2005.

[131] L. Wang, S. Jajodia, and A. Singhal. *Network Security Metrics*. Springer, 2017.

[132] L. Wang, S. Jajodia, A. Singhal, P. Cheng, and S. Noel. k-zero day safety: A network security metric for measuring the risk of unknown vulnerabilities. *IEEE TDSC*, 11(1):30–44, 2014.

[133] Lingyu Wang, Mengyuan Zhang, Sushil Jajodia, Anoop Singhal, and Massimiliano Albanese. Modeling network diversity for evaluating the robustness of networks against zero-day attacks. In *European Symposium on Research in Computer Security*, pages 494–511. Springer, 2014.

[134] Lingyu Wang, Mengyuan Zhang, and Anoop Singhal. Network security metrics: from known vulnerabilities to zero day attacks. In *From Database to Cyber Security*, pages 450–469. Springer, 2018.

[135] Q. Wang, W. Guo, K. Zhang, A. Ororbia II, X. Xing, X. Liu, and C. Giles. Adversary resistant deep neural networks with an application to malware detection. In *ACM KDD*, pages 1145–1153, 2017.

[136] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS'03)*, pages 25–34, 2003.

[137] Yoav Weiss and Elena Gabriela Barrantes. Known/chosen key attacks against software instruction set randomization. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 349–360. IEEE, 2006.

[138] T. Wood, R. Singh, A. Venkataramani, P. Shenoy, and E. Cecchet. Zz and the art of practical bft execution. In *Proceedings of the sixth conference on Computer systems*, pages 123–138, 2011.

[139] Avishai Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6):62–67, 2004.

[140] Jun Xu, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Transparent runtime randomization for security. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 260–269. IEEE, 2003.

[141] L. Xu, Z. Zhan, S. Xu, and K. Ye. Cross-layer detection of malicious websites. In *ACM CODASPY*, pages 141–152, 2013.

[142] L. Xu, Z. Zhan, S. Xu, and K. Ye. An evasion and counter-evasion study in malicious websites detection. In *IEEE CNS*, pages 265–273, 2014.

[143] M. Xu, L. Hua, and S. Xu. A vine copula model for predicting the effectiveness of cyber defense early-warning. *Technometrics*, 59(4):508–520, 2017.

[144] M. Xu and T. Kim. Platpal: Detecting malicious documents with platform diversity. In *Proc. USENIXSecurity*, pages 271–287, 2017.

[145] M. Xu, K. M. Schweitzer, R. M. Bateman, and S. Xu. Modeling and predicting cyber hacking breaches. *IEEE T-IFS*, 13(11):2856–2871, 2018.

[146] M. Xu and S. Xu. An extended stochastic model for quantitative security analysis of networked systems. *Internet Mathematics*, 8(3):288–320, 2012.

[147] Maochao Xu, Gaofeng Da, and Shouhuai Xu. Cyber epidemic models with dependences. *Internet Mathematics*, 11(1):62–92, 2015.

[148] S. Xu. Cybersecurity dynamics: A foundation for the science of cybersecurity. In *Proactive and Dynamic Network Defense*, pages 1–31. 2019.

[149] S. Xu. The cybersecurity dynamics way of thinking and landscape. In *ACM Workshop on Moving Target Defense*, 2020.

[150] S. Xu, W. Lu, L. Xu, and Z. Zhan. Adaptive epidemic dynamics in networks: Thresholds and control. *ACM TAAS*, 8(4), 2014.

[151] S. Xu, W. Lu, and Z. Zhan. A stochastic model of multivirus dynamics. *IEEE Transactions on Dependable and Secure Computing*, 9(1):30–45, 2012.

[152] Shouhuai Xu. Cybersecurity dynamics. In *Proc. HotSoS'14*, pages 14:1–14:2, 2014.

[153] Shouhuai Xu. Emergent behavior in cybersecurity. In *Proceedings of the 2014 Symposium on the Science of Security (HotSoS'14)*, pages 13:1–13:2, 2014.

[154] Shouhuai Xu. SARR: A cybersecurity metrics and quantification framework (keynote). In Wenlian Lu, Kun Sun, Moti Yung, and Feng Liu, editors, *Science of Cyber Security - Third International Conference (SciSec'2021)*, volume 13005 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2021.

[155] Shouhuai Xu, Wenlian Lu, and Hualun Li. A stochastic model of active cyber defense dynamics. *Internet Mathematics*, 11(1):23–61, 2015.

[156] Shouhuai Xu, Wenlian Lu, and Li Xu. Push- and pull-based epidemic spreading in networks: Thresholds and deeper insights. *ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS)*, 7(3):32, 2012.

[157] Shouhuai Xu and Kishor Trivedi. Us nsf satc 2019 pi meeting breakout group report briefing: Cybersecurity metrics: Why is it so hard?, 2019.

[158] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers: A case study on pdf malware classifiers. In *NDSS*, January 2016.

[159] L. Yang, X. Yang, and Y. Tang. A bi-virus competing spreading model with generic infection rates. *IEEE Trans. Netw. Sci. Eng.*, 5(1):2–13, 2018.

[160] Yi Yang, Sencun Zhu, and Guohong Cao. Improving sensor network immunity under worm attacks: a software diversity approach. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 149–158, 2008.

[161] Z. Zhan, M. Xu, and S. Xu. Characterizing honeypot-captured cyber attacks: Statistical framework and case study. *IEEE T-IFS*, 8(11), 2013.

[162] Z. Zhan, M. Xu, and S. Xu. A characterization of cybersecurity posture from network telescope data. In *Proc. InTrust*, pages 105–126, 2014.

[163] Zhenxin Zhan, Maochao Xu, and Shouhuai Xu. Predicting cyber attack rates with extreme values. *IEEE T-IFS*, 10(8):1666–1677, 2015.

[164] Mengyuan Zhang, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Network attack surface: Lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[165] Mengyuan Zhang, Lingyu Wang, Sushil Jajodia, Anoop Singhal, and Massimiliano Albanese. Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. *IEEE Transactions on Information Forensics and Security*, 11(5):1071–1086, 2016.

[166] Yongguang Zhang, Harrick Vin, Lorenzo Alvisi, Wenke Lee, and Son K Dao. Heterogeneous networking: a new survivability paradigm. In *Proceedings of the 2001 workshop on New security paradigms*, pages 33–39, 2001.

[167] Ren Zheng, Wenlian Lu, and Shouhuai Xu. Active cyber defense dynamics exhibiting rich phenomena. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, pages 1–12, 2015.

[168] Ren Zheng, Wenlian Lu, and Shouhuai Xu. Preventive and reactive cyber defense dynamics is globally stable. *IEEE Trans. Network Science and Engineering*, 5(2):156–170, 2018.

[169] Deqing Zou, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin. $\mu$vuldeepecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 01 2019.

[170] Deqing Zou, Yawei Zhu, Shouhuai Xu, Zhen Li, Hai Jin, and Hengkai Ye. Interpreting deep learning-based vulnerability detector predictions based on heuristic searching. *ACM Trans. Softw. Eng. Methodol.*, 30(2), March 2021.

## VITA

Huashan Chen was born in Jiangsu, China. He received the B.S. degree from Shandong University in 2012 and received the M.S. degree from the Institute of Information Engineering, Chinese Academy of Sciences, in 2016. He joined the Department of Computer Science at the University of Texas at San Antonio since 2017. His primary research interests are in cybersecurity, especially security metrics and moving target defense.