CYBER AGILITY FOR ATTACK DETERRENCE AND DECEPTION

by

Jafar Haadi Jafarian

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2017

Approved by:

Dr. Ehab Al-Shaer

Dr. Bill Chu

Dr. Bojan Cukic

Dr. Jinpeng Wei

ABSTRACT

JAFAR HAADI JAFARIAN. Cyber Agility for Attack Deterrence and Deception.
(Under the direction of DR. EHAB AL-SHAER)

In recent years, we have witnessed a rise in quantity and sophistication of cyber attacks. Meanwhile, traditional defense techniques have not been adequate in addressing this status quo. This is because the focus has remained mostly on either identifying and patching exploits, or detecting and filtering them. These techniques are only effective when intrusions are known or detectable. However, unknown (zero-day) vulnerabilities are constantly being discovered, and known vulnerabilities are not often patched promptly. Even worse, while defenders need to patch all vulnerabilities and intrusions paths against unknown malicious entities, the attackers only need to discover only one successful intrusion path in a system that is known and static. These asymmetric advantages have constantly kept attackers one step ahead of defenders.

To reverse this asymmetry in cyber warfare, we aim to propose new proactive defense paradigms that can deter or deceive cyber attackers without relying on intrusion detection and prevention and by offering *cyber agility* as a system property. Cyber agility allows for system configuration to be changed dynamically without jeopardizing operational and mission requirements of the system.

In this thesis, we introduce two novel cyber agility techniques based on two paradigms of *cyber deterrence* and *cyber deception*. Cyber deterrence techniques aim to deter cyber threats by changing system configurations randomly and frequently. In contrast,

cyber deception techniques aim to deflect attacks to fake targets by misrepresenting system configurations strategically and adaptively.

In the first part of this dissertation, we propose a multi-strategy, multi-parameter and multi-dimensional host identity mutation technique for deterring reconnaissance attacks. This deterrence is achieved by mutating IP addresses and anonymizing fingerprints of network hosts both proactively and adaptively. Through simulation and analytical investigation, we show that our approach significantly increases the attack cost for coordinated scanning worms, advanced network reconnaissance techniques, and multi-stage APT attacks.

In the second part, we propose a formal framework to construct active cyber deception plans that are goal-oriented and dynamic. Our framework introduces a deception logic that models consistencies and conflicts among various deception strategies (e.g., lies) and quantifies the benefit and cost of potential deception plans.

In the third part, we demonstrate and evaluate our deception planning framework by constructing an effective deception plan against multi-stage attacks. Through our experimentation, we show that the generated deception plans are effective and economical, and outperform existing or random deception plans.

ACKNOWLEDGMENTS

I am thankful to my advisor, Prof. Ehab Al-Shaer, for his professional and personal mentorship. I am also grateful to Prof. Lisa Russel-Pinson for her valuable insights and aspiring guidance. Thanks to Prof. Cukic, Prof. Chu, and Prof. Wei for their feedback and contribution. Also, I would like to express my sincere gratitude to Prof. Maher for her support and advice. I would like to thank my parents, my wonderful sister, and my lovely brothers for their love during these stressful years. Thanks to Zohreh for standing by my side during all the hard moments. And last, but not least, I would like to thank my beloved friends at UNCC, especially Amirreza, Pedram, Sina, Kaveh, Reza, Qasim, Fida and Ashutosh without whom this long journey would have been even harder.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Motivation

With ever-increasing prevalence and presence of the Internet in our lives, cyber security has turned into to a matter of national and international concern. Meanwhile, over the past few decades, the quantity and severity of sophisticated cyber attacks have increased significantly [53, 63]. In recent years, we have witnessed a variety of advanced and persistent cyber attacks by well-resourced and highly sophisticated attackers [63,68] that targeted highly sensitive economic, political, or national security information [53, 99].

Meanwhile, traditional approaches to cyber defense are hardly adequate to defend against these emerging advanced cyber attacks [53, 63]. This is because the cyber is asymmetric as attackers have more advantage over defenders regarding information gathering and damage. For example, while attackers need just a single undetected exploit to successfully compromise the system, defenders need to exhaustively block attacks to all potential vulnerabilities. Even worse, static configuration of cyber systems enables attackers to learn the cyber system properties, while defenders have fundamental challenges to detect attackers' reconnaissance activities or to discover their motive. These asymmetries enable attackers to be many steps ahead of defenders.

To change this asymmetric status quo in cyber warfare, new proactive paradigms are needed to deter unknown attackers without relying on detection and prevention techniques. Recognition of this dire need has resulted in techniques that incorporate *cyber agility* as a property into the system. Cyber agility is a system property that allows for dynamic change of the underlying system configuration. The goal is to enable the cyber to proactively defend against unknown threats by dynamically changing the system parameters and defense strategies in a timely and economical fashion.

As a result of this proactive defense, effective cyber deterrence is enabled to reverse this asymmetry in cyber warfare. Cyber deception is another cyber agility technique that enables deflecting attackers to invalid target by misguiding them through fake assets.

In this thesis, we introduce two novel cyber agility techniques that enable proactive defense, called *cyber deterrence* and *cyber deception*. Both techniques focus on defeating advanced cyber threats in their reconnaissance or information gathering stage. Defeating reconnaissance would have a huge impact on advanced cyber attacks because reconnaissance is the precursory step of such attacks (see cyber kill-chain [53]) and its objective is to collect information to identify potential intrusion points into the targeted system.

Cyber deterrence techniques such as moving target defense (MTD) approaches aims to deter cyber threats by changing the system parameters randomly and frequently. Examples of MTD techniques for cyber deterrence include instruction set randomization [69, 92], memory address randomization [111], and compiler-generated software

diversity [55] to avoid attacks such as buffer overflows and worms [111]. Our focus in this thesis is on cyber deterrence techniques that target network reconnaissance, such as IP address mutation techniques.

Several MTD-based cyber deterrence techniques against reconnaissance have been proposed in the literature [1,2,6,9,33,51,67,70,130,131]. However, existing approaches provide mutations that are slow and predictable. Moreover, the provided mutations are usually only proactive, temporal, and often non-transparent to end-hosts or network protocols [6, 9, 70]. A collection of these weaknesses limits the agility of existing techniques, thus limiting their effectiveness to automated naive reconnaissance models such as hitlist scanners [6] or automated network worms [1, 56].

In this thesis, we propose a cyber deterrence approach that provides high agility (high unpredictability and high mutation rate), is transparent to cyber systems (do not require changes in end-hosts and legacy protocols) and incurs very low overhead (can be deployed with reasonable cost and without breaking network sessions). The agility is enabled across various dimensions and composes various mutation strategies, to maximize effectiveness, disallow evasion, and minimize overhead.

The second approach focuses on cyber deception as another technique to offer cyber agility. Cyber deception is a misrepresentation of system configurations for the sake of misguiding attackers. The primary goal of this misguiding usually goes beyond attack deterrence to a characterization of attackers' techniques and motive. Although deception technologies such as honeypots [96] have been present in cyber defense for almost three decades [26], the focus has been primarily on devising passive and static deception traps (tools). This architecture lacks the dynamic agility that is necessary

to adapt to attackers' behavior and allow for wide deployment. In this thesis, we propose a framework for the cyber deception that provides adaptive, automated, and goal-oriented deception plans against various cyber attackers. Using this framework, we model and propose an adaptive, and cost-effective cyber deception plan to defeat multi-stage cyber attacks.

1.2 Background

In this section, we overview some concepts and technologies that are essential to understanding the contribution of this work.

1.2.1 Cyber Intrusion Kill-Chain

A kill chain is a discrete description of a systematic process to identify, target and engage an adversary to create desired effects. *Intrusion kill chain* is a new chain model that describes cyber intrusions. The cyber kill chain focuses on advanced and persistent threats where the attacker must do reconnaissance to identify the targets, and develop suitable payloads to compromise and bypass a trusted perimeter. Once inside, the attacker would take actions toward the objective, by laterally moving inside the environment. At every new location, the attacker may repeat this process to identify new potential targets, compromise them, and expand her intrusion inside the environment [53].

According to this APT model, the intrusion kill-chain is broken down into the following steps: reconnaissance, weaponization, delivery, exploitation, installation, command and control (C2), and actions on objectives.

- Reconnaissance: a process of discovering and selection of targets, using a va-

riety of techniques, ranging from active scanning and passive probing to social engineering techniques.

- Weaponization: developing an exploit and incorporating it in a deliverable payload, such a PDF document or a service (HTTP) request.
- Delivery: sending the weaponized payload to the targeted environment, using network, USB removable media, email attachments, or drive-by downloads.
- Exploitation: After the weaponized payload is delivered to the target, its execution launches the attack code that would try to exploit a vulnerability in the operation system, or services. The execution may be automated via a vulnerability or may simply occur by luring the user to do so.
- Installation: installation of a backdoor on the exploited target in order to maintain persistence inside the environment.
- Command and Control: in many cases, compromised targets establish outbound connections for command and control to the adversary, that allows the adversary to have remote presence inside the target environment, thus turning the attacker into an insider.
- Actions on Objectives: after achieving a privileged and insider access, the attacker can take actions toward their primary objectives. This objective could be data exfiltration, or even sabotaging integrity and availability of the target. Alternatively, the attacker may use the new compromised target as a point for compromising additional systems and moving laterally deeper inside the environment. According to the ATT&CK Matrix by MITRE [85], the actions include persistence, privilege escalation, defense evasion, credential access, dis-

covery, lateral movement, execution, collection, and further command and control. Among these actions, the most important and also relevant to this thesis is lateral movement. Lateral movement consists of techniques that enable an adversary to access and control remote systems on a network, either using an installed backdoor or using conventional communication tools using the escalated privilege on that target.

A special and important class of reconnaissance includes network reconnaissance, which refers to an active or passive process through which potential attackers collect information about the target network, including its active ranges, active and reachable systems, OS and services of these systems, and potential vulnerabilities or misconfiguration on those services or other components of the network. Based on technique, network reconnaissance is categorized into two classes: active and passive. Active network reconnaissance is done via probing scans generated by a custom tool or using off-the-shelf tools such as Nmap [78] or Nessus [75], or it could be passive sniffing or eavesdropping on target networks' traffic, or inspecting system caches or open connections [35]. The reconnaissance of a target network could also be done externally or internally. This is because the network perimeters are isolated from the Internet, using firewalls and other security devices, and therefore the majority of target resources are not accessible from outside. However, once an attacker is inside the target network (after successful exploitation of a public system and laterally moving to it), then they will have much fewer security restrictions in probing and identifying internal systems [77]. External network reconnaissance occurs when an attacker is probing the public or demilitarized zone (DMZ) of the target network, while internal

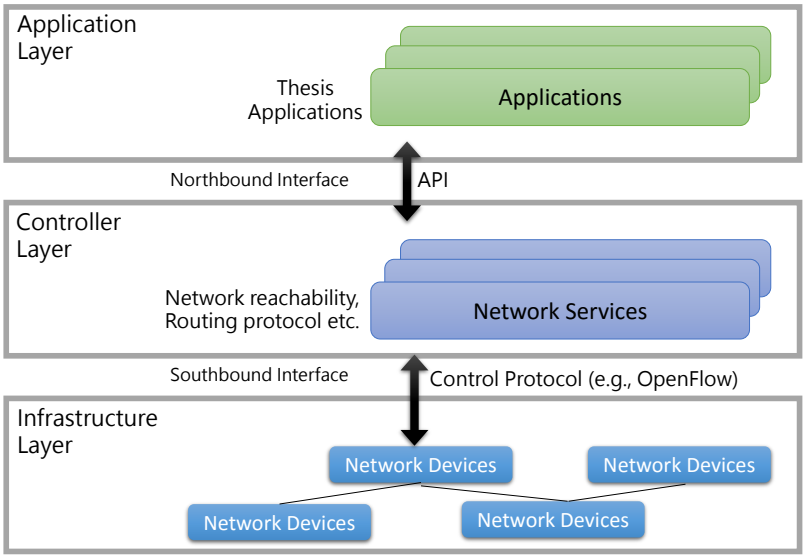

Figure 1: SDN Architecture

reconnaissance refers to reconnaissance launched by an insider.

1.2.2 Software-Defined Networks (SDN)

SDN is a novel networking model in which the networking is controlled and implemented by a software [82]. The main goal of SDN is to centralize control of the traffic by migrating the control logic to centralized computer resources. A network controller monitors and controls the entire network from a *central* vantage point via an interface, such as OpenFlow [82], which allows researchers and administrators to develop their customized networking model. Communication with SDN usually occurs via southbound and northbound APIs, as illustrated in Figure 1 [42].

The control plan consists of three main components: controller(s) that can access and manage all resources from a central location using OpenFlow (OF) protocol; OpenFlow agents on switches that are able to communicate with the controller through OF messages, and finally forwarding plane which is responsible for exchange

OF messages between the controller and switches [42].

While hardware SDN switches cost thousands of dollars, SDN emulation tools such as Mininet [74] provide a cheap and scalable emulation framework for testing. Mininet is a network emulator [74] that allows emulation and orchestration of a network consisting of a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight software virtualization to make a single process emulate a high-fidelity network host with a real Linux core that can accept *ssh* or run the Linux applications.

1.2.3 Satisfiability Modulo Theories and Microsoft Z3 SMT Solver

The Satisfiability modulo theories (SMT) problem is a decision problem for logical formulas concerning combinations of background theories expressed in classical first-order logic with equality. Formally speaking, an SMT instance is a formula in first-order logic, where some function and predicate symbols have additional interpretations, and SMT is the problem of determining whether such a formula is satisfiable.

SMT formulas provide a much richer modeling language than is possible with Boolean SAT formulas. Although satisfiability problems are NP-complete in general, recent advances in SMT solvers have made them scalable to problems with millions of variables [87].

SMT solvers have found applications across a wide variety of domains, including verification, proving the correctness of programs, and software testing based on symbolic execution. Although satisfiability problems are NP-complete in general, recent advances in SMT solvers have made them scalable to problems with millions of vari-

ables [87].

Z3 is a state-of-the-art theorem prover from Microsoft Research [84, 98], which is freely available for academic research. It has built-in support for integer and real constants, which are mathematical integers and reals, not machine integers. It also includes support for bit vectors, uninterpreted functions, extensional arrays, and quantifiers.

Z3 is a low-level tool and is best used as a component in the context of other tools that require solving logical formulas. Z3 includes a number of APIs for *C++* and *.NET* to simplify its embedding into other applications; in fact, there are no standalone editors or user-centric services for interacting with Z3. The language syntax used in the front ends favor simplicity in contrast to linguistic convenience.

1.3 Characteristics of Advanced Cyber Attacks

While cyber threats have always been a phenomenon, in recent years we have witnessed an increase in sophistication of cyber threats [99]. These threats are usually launched by well-resourced and trained adversaries [63] that conduct multi-year intrusion campaigns targeting highly sensitive economic, proprietary, or national security information [53]. These adversaries accomplish their goals using advanced tools and techniques designed to defeat existing defense mechanisms. In recent years, we have observed multiple examples of such advanced intrusions that have spanned over a significant period, evaded firewall and anti-virus capabilities, and enabled adversaries to harvest sensitive information [53].

Traditional approaches to cyber defense are not able to counter these emerging

advanced and persistent cyber threats. These techniques are only able to detect known attacks, because they rely on detecting a signature or behavior of the attack that is known to be malicious; however, advanced and persistent attacks are usually launched by elite attackers who use low-and-slow stealthy reconnaissance as well as unknown or zero-day exploits in order to evade these signature-based or behavioral intrusion detection systems.

The APT attacks usually have the following properties:

- Persistent and stealthy presence of attackers: the potential attackers are silently present in the network for a long while probing and attacking network hosts and services to find a path toward their intended target.
- Advanced (stealthy and unknown) tactics and techniques: while certain attack behavior (e.g., Nmap [78] probes) could be detected by intrusion detection systems, skilled attackers use some stealthy and unknown techniques to bypass these countermeasures. These techniques could be learned during an attacker's presence in the network.
- Multi-staged: due to zoning and deployment of defense-in-depth strategies in networks, such threats are usually initiated by compromising a publicly accessible host. They gradually continue to compromise hosts and move from one zone to another until they finally discover a path to the target [39]. At each zone, an attacker needs to probe the address space and discover reachable hosts until the target hosts are identified.

Defeating such advanced, persistent, and stealthy threat models requires new defense paradigms beyond intrusion detection and prevention. In the next section, we

explain the limitations of state-of-the-art defense paradigms against such advanced cyber attacks.

1.4 Limitations of the State-of-the-art against Advanced Cyber Attacks

Traditional reactive approaches to cyber defense, such as intrusion detection, are not sufficient to address emerging advanced and persistent threats because they only provide countermeasures against an attack only after it happens; and also they only detect an attack if its behavior or signature is known [53].

A major defense against cyber threats has been focused on discovering, categorizing and patching vulnerabilities. However, vendors are usually slow in patching discovered vulnerabilities, and in many cases, very severe vulnerabilities have remained unpatched for long periods of time [63]. In fact, many patches are often published after the vulnerabilities are known and have been exploited, in some cases after months and years [62, 63].

Even when patches are published, enterprises are very slow in applying those patches in their systems, usually due to lack of resources and a deficiency in realizing the potential threat of successful compromises. A 2015 report by Verizon showed that 99.9% of vulnerabilities are exploited over a year after their CVE is published [115].

But the real problem is that even if all the patches are published and deployed promptly, not all vulnerabilities are known or immediately patchable by vendors (e.g., zero-day vulnerabilities). For example, Stuxnet, a worm targeting industrial control systems in 2010 [37], used four zero-day vulnerabilities for exploitation. These exploits were not known to the cyber security community, thus giving Stuxnet significant

advantage in stealthy propagation.

Even worse, an attacker needs to find only a single exploitable vulnerability to infiltrate; meanwhile, the defender must exhaustively ensure none exists [62]. But finally, one uncovered intrusion path is sufficient for a persistent attacker to compromise and infiltrate the system.

Another major defense paradigm is detection and prevention of attacks. However, traditional defense tools such as intrusion detection systems, firewalls, and malware detectors can counter known attack techniques. Therefore, if the attack goes undetected, they provide zero resistance against it. Even worse, thwarting one or a few intrusion attempts does not necessarily terminate the attack. On the contrary, it allows persistent adversaries to learn properties of the cyber system until they discover an uncovered path for an intrusion.

Another factor that significantly contributes to this situation is the static nature of cyber systems. This means that attackers have a static target to study and find vulnerabilities and then a window of exposure to exploit the vulnerability to gain privileged access on other machines and networks until the exploit is noticed, the vulnerability found, patch released, and then applied widely.

As a result of these problems, in the cyber warfare between attackers and defenders, attackers have a significant advantage over defenders both in collecting information about the other party and also in inflicting damage on them. The attacker has plenty of time to investigate these static targets, and during this process, the system provides little or no resistance against such adversarial reconnaissance. Even worse, most reactive defense paradigms rely on distinguishing attack signatures or behavior,

which has zero effectiveness against zero-day or even stealthy attack techniques. These asymmetries would always keep attackers one step ahead of defenders.

1.5 Cyber Agility against Advanced Cyber Attacks

Changing these asymmetries requires approaches that can provide some resistance against such advanced attacks, even when all or part of attack techniques, tactics or procedures are zero-day, unknown or undetectable. These techniques must not rely on reactive detection of attackers' behaviors. These new paradigms must provide two capabilities. First, they should be able to provide resistance against the attack, even when the attack is unknown or zero-day; i.e., it is not necessarily recognizable by the defender. This resistance must deter the attack by increasing the time or effort required to complete the attack. Second, to change the asymmetry, they must enable characterization of the attack regarding its goals, objectives, and techniques.

Cyber agility is a novel class of active cyber defense that has a high-level goal of providing resistance against the most complicated and stealthy cyber threats. Cyber agility is a system property that enables the cyber to proactively defend against unknown threats by dynamically changing the system parameters and defense strategies in a timely and economical fashion. Agility provides robustness and resilience for the system to defend against completely or partially unknown attacks. By strategically establishing dynamics into the system, cyber agility provides a proactive defense to deter many attacks including worms, botnets, DoS and reconnaissance attacks with the presence of uncertainties of the timing and types of attacks.

In this thesis, we explore and develop paradigms for two novel approaches in cyber

agility, called *cyber deterrence* and *cyber deception*. Cyber deterrence focuses on proactive randomization or mutation of system configuration parameters to incur uncertainty on the attackers, thus deterring the attack. This deterrence occurs either by slowing down attack completion or by increasing the cost of attack, thus raising the bar for cyber attackers. For example, *mutation* of IP addresses frequently over time makes the information gathered about a host on a certain IP address obsolete, thus forcing the attacker to recollect information about that IP or host again and again. Cyber deception, on the other hand, focuses on lying about the real value of system configuration parameters. These lies are crafted and implanted in the system proactively, and their goal is to persuade attackers to take a course of action that is in defenders' favor. For example, by luring attackers toward decoy machines (honeypots) in a network, defenders can deflect attackers from critical hosts. Also, engagement with decoy machines makes attackers' techniques and goals visible to defenders.

Cyber agility aims to provide proactive resistance against attacks by adding agility to the cyber infrastructure. However, cyber deterrence provides this agility differently from cyber deception. Specifically, for cyber deterrence, configuration parameters values are mutated frequently, the goal of which is to introduce new unaccounted-for challenges for attackers. However, cyber deception provides this agility, not by agile changing of system parameter values, but by showing fake values for these parameters. The goal of this deception is to persuade attackers to take wrong actions in the defender's favor.

In this sense, there is a fundamental difference between cyber deterrence and cyber deception. Cyber deterrence is not necessarily required to be invisible to attackers;

rather, it achieves its objectives by frequent mutation of system configurations. For example, mutation of hosts' addresses is observable by attackers. Cyber deception, in contrast, is only achievable through stealthiness; in other words, deception by nature must not be visible (detectable) to attackers. For example, if attackers know about the decoy nature of a host, they will avoid probing or attacking that host.

While different regarding methodology, both deterrence and deception pursue the same objectives:

- Goal A - Attack Deterrence: deterring the attack occurrence by increasing cost and uncertainty in its planning and execution.
- Goal B - Attack Deception and Characterization: increase the chance of detecting and characterizing the attack.

However, while cyber deterrence techniques primarily focus on achieving goal A, cyber deception techniques primarily aim to achieve goal B. For example, IP mutation primarily aims to deter network reconnaissance through frequent changing of IP addresses; however, as a secondary goal, this mutation increases the probability that an attacker issues a dark scan (to a non-existing IP or port), thus increasing attack detectability. In contrast, deploying a group of honeypots in the address space primarily aims to engage attackers and increase the chance of their detectability and characterization; however, as a secondary goal, this engagement with decoys would deter attack progression by deflecting attackers from critical network hosts.

A variety of cyber agility techniques for deterrence and deception of different threat models have been proposed in cyber security. Cyber deterrence has been primarily used to defeat advanced threat models through randomization: instruction set ran-

domization [69, 92] to defeat code injection attacks, address space layout randomization (ASLR) [111] to prevent exploitation of memory corruption vulnerabilities, compiler-generated software diversity [55] to defeat buffer overflows and code manipulation attacks [111], and route mutation [34, 57] to defeat eavesdropping and denial-of-service attacks. Several cyber deterrence approaches have been proposed for defeating network reconnaissance [6, 15, 51, 70, 130]. These approaches have done some preliminary efforts in mutating IP addresses of network hosts; IP mutation refers to randomization of IP addresses of network hosts, usually over time, to expire the validity of information collected about network hosts. However, none of the previous techniques provide a deployable transparent mechanism for address randomization that exploits the full potentials of cyber deterrence, especially regarding achieving high unpredictability and fast mutation rate. This low unpredictability and constrained mutation speed make these models only effective against trivial threat models, such as hitlist worms [6]. These techniques are not sufficiently agile to provide deterrence against more advanced threat models, such as automated random scanners (e.g., network worms) or reconnaissance for intrusion attacks. Moreover, they require changes in existing network protocols and devices and break active network sessions. This makes their agility severely intrusive to the operation and integrity of the system.

Cyber deception techniques and paradigms for defending cyber systems have been proposed in the literature; however, the focus has been mostly on devising honey things; from honeypots [72, 96] and honeyfarms [126] to honey router [41] and honeyclient [3]. The primary objective of these defensive deception techniques has been

to disrupt attacks in their reconnaissance stage, by providing misleading information to attackers. However, the popularity and evolution of honey techniques have rather diminished in recent years, mostly because the focus has remained on devising isolated honey-thing systems and techniques. In fact, existing cyber deception techniques are static and isolated, often configured and deployed individually, and usually very susceptible to exposure [7]. However, *random* use of one or a few isolated and static deception techniques have limited effectiveness, especially against sophisticated reconnaissance attackers. Instead, effective deception of such attackers requires a strategic and dynamic composition of these deception systems in a manner that their enterprise-wide deception plan provides a good incentive to attackers for engagement and minimum likelihood of exposure. While several formal deception planning frameworks have been proposed in the literature [25, 26, 36, 50, 107, 108], the majority of existing models are not designed (and therefore not suitable) for cyber deception planning; moreover, they all consider a very limited number of deceptive actions in their planning. Therefore, they do not provide very the scalability and real-time agility that is required for cyber deception.

In the next sections, we overview the state-of-the-art for both cyber deterrence and cyber deception paradigms and techniques against network reconnaissance attacks, followed by a detailed discussion of their shortcomings and limitations.

1.6 Overview and Limitation of IP Mutation for Cyber Deterrence

Mutation techniques [56, 61, 93] for deterrence of network reconnaissance focus on changing the static nature of cyber systems and establish dynamics in them by ran-

domly and frequently mutating certain parameters of the system.

The main trend in this regard focuses on mutating configuration of network hosts, especially their IP addresses, to thwart adversarial reconnaissance, network scanners and worms [59]. The APOD (Applications that Participate in their Own Defense) scheme uses *hopping tunnels* based on address and port randomization to disguise the identity of end parties from sniffers [9]. DyNAT provides a transparent approach for IP hopping by translating the IP addresses before packets enter the core or public network to hide the IP address from man-in-the-middle sniffing attacks [70]. NASR proposes an IP hopping LAN-level network address randomization scheme based on DHCP to defend against hitlist worms [6].

Several approaches have used IP mutation as a technique against honeypot mapping attacks [15, 131]. Another approach proposes an address mutation technique for protecting MANET nodes against attackers' reconnaissance efforts [2]. SDNA proposes an agility approach at the hypervisor level, to make host appearance dynamic to observers while retaining transparency to OS, applications, and end-users [130]. Several approaches have suggested techniques for mutation of IPv6 addresses [130]. While IP scanning attacks are not possible in IPv6 due to the abundance of addresses, sniffing attacks are still viable. To protect a specific flow between two endpoints against sniffing attacks, MTD6 rotates addresses of the sender and the receiver mid-session without dropping or renegotiating sessions [67]. DHCP6D uses DHCP for address mutation to avoid the cost of address generation and verification [33]. Another approach referred to as MOB6D protects mobile devices against sniffing and reconnaissance attacks by randomizing the prefixes of IPv6 addresses [51].

Existing approaches each suffer from one or several of the following shortcomings, thus confining their effectiveness especially against network reconnaissance attacks:

- **Limited and slow mutation**: existing models are only able to defeat random scanners due to limited agility; that is, slow mutation rate. This limited mutation rate is not sufficient to deter network scanners; in fact, we show that to achieve a maximum effectiveness against scanners, the mutation rate must be equal to the scanning rate. Existing approaches are not able to achieve such high mutation rates, thus making them only effective against mapping attack models, such as hitlist worms [6] or naive honeypot mapping techniques [15,131]. Moreover, existing approaches only mutate addresses at fixed intervals, and do not consider adversarial behavior or scanning rates in determining their mutation strategy or rate. This significantly constrains their effectiveness against intelligent scanning strategies, such as cooperative or local-preference [59].
- **No effectiveness against fingerprint-based identification**: existing techniques only mutate one parameter, which is IP addresses of network hosts. Mutation of IP addresses over time breaks association that the attacker has established between information collected about a host and its IP address. However, if a host has a unique fingerprint, an elite attacker would be able to identify a host using this fingerprint [60], thus bypassing this IP mutation.
- **Intrusive and non-transparent mutation**: existing techniques are not transparent to network protocols or devices; i.e., their deployment requires costly changes in existing infrastructure. Even worse, they break active network sessions, thus making their agility highly disruptive to the integrity of network

operation.

- **Non-adaptive and inefficient mutation**: the proposed techniques in the literature only mutate addresses of network hosts uniformly and with a fixed rate. This has two repercussions. First, when there is no reconnaissance (scanning) activity, this results in useless mutations of addresses, which incurs unnecessary costs on the system. Second, as mentioned above, when an attacker is scanning the network with a high rate, this fixed rate would result in lower deterrence [1], as compared to a higher mutation rate.
- **Limited effectiveness against APT and collaborative scanners**: existing approaches mutate host-to-address bindings globally [6, 9, 70, 131]. This means that information collected on a host could be used on another host, at least for a short interval. This allows clients to share their reconnaissance information with each other or reuse information collected on a previous host after a successful lateral movement.

1.7 Overview and Limitation of Cyber Deception Planning Frameworks

Most works on cyber deception focus on devising new decoy systems that can deceive a specific threat model [54, 88, 96, 114]. In addition to these decoy systems which provide isolated and static deception, a few works have focused on providing a framework for defining, modeling, and reasoning on deception.

A number of these approaches provide a formal definition of deception, using modal logic [16, 20, 31, 102, 107, 108]. Existing theoretical works on modeling deception focus on formal modeling of deceptive communication without providing means to combine

and synthesize various deceptive actions. They do not provide metrics or methodologies for measuring deception benefit and cost in a cyber context, which is required for identifying an optimal cyber deception plan. Several other works offer a game-theoretic modeling of specific cyber deception scenarios [17,36,40,133] that can identify the optimal cyber deception plan for a specific threat model. However, these models can only consider simple deception problems that only consist of a few number of deceptive actions.

Another class of works has used simple probabilistic models to make choices between alternative deception plans [101, 102]. While these models are more flexible than previous ones, none provide a generic framework for formal modeling of arbitrary cyber deception problems.

The existing works on cyber deception suffer from several major shortcomings:

- **Ineffectiveness of cyber static and isolated deception against elite attackers**: most works on cyber deception focus on offering isolated and static decoy systems [11, 96, 114]; these static and uncoordinated deployments have enabled advanced attackers to easily identify and evade these systems [60]. Defeating advanced attackers requires a coordinated, goal-oriented, and cost-effective combination of these decoy systems to provide a consistent and believable system-wide deception plan.
- **Limited effectiveness of existing deception planning models due to limited agility.** Existing frameworks address specific deception problems that include one or a few deceptive actions [17, 17, 36, 40, 90, 90]. Even worse, their planning is not automated and real-time. So, they fail to provide sufficient

agility and therefore high effectiveness.

- **Lack of a scalable and expressive framework for modeling generic cyber deception problems.** While existing frameworks have done some preliminary effort in cyber deception planning [16, 20, 31, 102, 107, 108], they fail to provide a logical foundation for defining cyber deception, modeling different types of deception, and providing metrics for measuring cost-benefit trade-off of different deception plans in an automated and adaptive manner. In fact, scientific and quantitative reasoning on deception requires a framework that can incorporate and model any potential deceptive action in the cyber context, as well as modeling the benefit-cost trade-off for various compositions of these actions. Most importantly, it must be able to formally model an attacker's thinking process and how these different compositions would affect attacker's beliefs and potential actions.
- **Lack of a comprehensive deception model against advanced reconnaissance.** The general objective of decoy technologies is to mislead attacks in their reconnaissance stage. This false reconnaissance could mislead attackers in next stages of kill-chain [53] by engaging them with exploiting decoys and deflecting them from exploiting real hosts. However, while each decoy technology could be even individually effective, a strategic and goal-oriented combination of these systems could generate a synergistic effect by satisfying expectations of even an elite reconnaissance attacker and providing tempting incentives for the attacker to probe and engage with a decoy. This goal-oriented composition requires a planning model that must consider configuration and vulnerability

of network hosts, different network zones and their access control rules, potential intrusion paths into the network, and also risk assessment information for network hosts.

1.8 Work Objectives

In this dissertation, we have developed cyber agility techniques and frameworks to defend against advanced reconnaissance attacks. Our techniques are built based on two primary paradigms: cyber deterrence and cyber deception. Our work objectives in this dissertation are as follows:

- **Developing techniques and metrics for proactive and adaptive cyber deterrence and deception against advanced network reconnaissance**. The static and truthful nature of cyber systems, in addition to the inability of existing reactive defense paradigms, enables cyber attackers to perform successful network reconnaissance with little or no resistance. We plan to offer novel techniques for incorporating cyber agility into the cyber systems to complicate and disrupt naive and advanced classes of network reconnaissance. These techniques must be able to provide deterrence and deception techniques against network reconnaissance in a proactive manner. Additionally, they must provide mechanisms and paradigms for characterizing attackers' behavior and adapting the agility to this behavior. Also, we develop metrics for quantifying benefit and cost of cyber deterrence and deception techniques for optimal planning and comparison.
- **Developing an adaptive, robust, and transparent cyber deterrence**

approach against network reconnaissance. Existing cyber deterrence techniques do not provide sufficient agility to defeat advanced scanning and network reconnaissance attacks. In this thesis, we present a cyber deterrence technique that provides high agility by enabling fast and unpredictable mutation of IP addresses. Our approach mutates host IP addresses in different dimensions (temporal and spatial) and based on a variety of strategies (proactive, adaptive, reactive), to prevent elite attackers from bypassing this mutation. Our mutation also anonymizes host fingerprints. This multi-dimensional and multi-strategy agility can deter a range of network reconnaissance attacks, from naive hitlist, honeypot mapping, and random scanners, to advanced network reconnaissance in multi-stage APT attacks.

- **Presenting a real-time adaptive deception planning framework for cyber defense**. We develop a formal framework, based on satisfiability modulo theories (SMT) [30] and a many-valued fuzzy logic system called Gödel logic [10], for defining and modeling deception. The framework provides logical foundations for modeling how deceptive actions and their different combination would affect the thinking process of potential adversaries. The model can capture conflicts and synergies among these deceptive actions. It also provides mechanisms for adaptation of the generated deception plans to new information about attackers or cyber systems.
- **Developing an effective, high-agility, and mission-oriented deception plan against advanced intrusion attacks**. In this dissertation, we present a high-agility deception plan that consists of existing decoy technologies to defeat

advanced intrusion attacks launched by elite attackers. This must consider potential intrusion paths into a given network, different zones of the network and their reachability policies, vulnerability levels of network hosts, in addition to the mission statement which determines defense priorities (i.e., critical hosts that must be protected). The generated plan must determine what combination of these decoy technologies must be deployed, with what configuration, and at which locations of the network, to maximally deceive and deter advanced intrusion attacks in the network. This optimal deception plan is modeled and determined using the aforementioned cyber deception framework.

1.9 Research Challenges

Toward fulfilling our work objectives, there are many research challenges that we need to address. The key challenges are as follow:

- **Constraint satisfaction problems for agility planning**. Achieving high agility is a complex problem that requires satisfaction of conflicting constraints because agility requirements often contradict with integrity, operational and budgetary constraints. For example, in cyber deception planning introduced in chapter 3, there is a trade-off between the effectiveness of deception and its cost, which makes this planning problem NP-hard. In general, agility planning requires solving hard and multi-constraint problems to achieve targeted protection level with affordable cost and overhead, which is a challenging task.
- **Selection and orchestration of agility parameters**. Defeating network reconnaissance as a threat model requires selection of a correct and complete set

of agility parameters to make this agility resistant to evasion techniques. This selection requires a systematic investigation of potential and available agility parameters and their disruptive effect on the given threat model, which is a challenging task. Also, these parameters must be orchestrated to avoid anomalies and conflicts in their operation, and also enable the synergistic composition to achieve a higher benefit. This orchestration requires discovering these conflicts and synergies, and using a framework to model such conflicts and synergies. Both of these tasks are challenging. The former task requires an in-depth understanding of the practical and technical repercussions of manipulating these parameters on the system, while the latter requires a formal understanding and modeling of the problem.

- **Quantifying benefit and cost of agility and their trade-off**. Quantifying the benefit and cost of an individual agility parameter or a combination of several agility parameters requires devising new metrics and quantification methodologies. In many cases, a combination of theoretical, simulation, and emulation methodologies must be used to quantify benefit and cost. Designing such metrics and methodologies are one of the focal challenges of this dissertation. This quantification is also important because the cyber agility measures must be fast enough to mitigate attack promptly and slow enough to limit the agility cost. This is because changing agility parameters incurs costs on the system, and therefore unnecessary mutations increases cost without increasing benefit. Discovering a balanced trade-off between this benefit and cost is one of the challenges of this dissertation.

- **Modeling adversarial behavior**. Cyber attackers are often advanced and adaptive. Effective agility requires developing techniques and methods for fast and accurate characterization of attacks as well as methodologies for belief generation and prediction. Given the complexities and unpredictability of human decision-making process and their ability to learn and adapt dynamically, such modeling and characterization are one of the most important challenges of this dissertation.
- **Maintaining service continuity and operational integrity**. Cyber agility requires continuous changing and adaptation of network configuration. Ensuring that these changes are not disruptive to integrity and continuity of services is an important challenge. For example, for IP mutation we need to ensure that these mutations do not violate end-to-end reachability of the network. As another example, for cyber deception, we must ensure that normal operations of the network are not affected by deceptive actions.
- **Scalability**. The agility solutions must be solved for networks with thousands of devices. Unscalable solutions are not practical for real-world scenarios, and provided solutions must be scalable; solving constraint satisfaction problems and agility models of such large size is a challenging task.

1.10 Contributions

Our contribution in this dissertation is as follows:

- A multi-dimensional host identity hiding technique. Our approach achieves high agility by fast, adaptive, and unpredictable mutation of IP address; it achieves

low overhead by making mutations and fingerprint anonymizations transparent to network protocols, users, and devices in a manner that is non-intrusive to active network sessions. Our cyber deterrence technique anonymizes host identities in a multi-dimensional, multi-parameter, and multi-strategy manner. The multi-dimensional mutation changes address temporally and spatially to defeat information gathering aggregation both over time and location. The approach also uses multiple novel strategies for mutation of addresses: proactive mutation when there is no characterization of attackers' strategies, the adaptive mutation when certain patterns are observed in attackers' strategies, and reactive mutation against malicious scans. We also offer a novel framework for characterization of attackers' scanning strategies for adaptive mutation.

- We present a deception planning framework for cyber defense that provides logic, interfaces, and mechanisms for modeling and constructing deception plans for cyber threats, and for adapting the plan based on the system feedback. The framework presents interfaces for defining a *deception model* for a given cyber domain, using a *deception logic*. The deception modeling logic is an abstraction over satisfiability modulo theories (SMT) [12] extended by many-valued logic [10], and it can model perlocutionary aspects of deception (e.g., belief, cause-and-effect, and intention), as well as its quantitative traits (belief certainties, budget, and risk). The framework includes necessary mechanisms for validating and solving the model to generate a sound deception plan, which then will be deployed in the given system.
- We develop an effective and orchestrated deception plan against advanced APT

attacks. This deception plan composes a variety of decoy technologies, based on networks' configuration and potential intrusion paths, defense priorities, and zoning and access control policies of the system. The deception plan is modeled and crafted using our deception planning framework. The deception plan achieves high agility by considering a large and diverse group of potential decoy systems. We show that the deception plan outperforms other competing plans by achieving a higher effectiveness with a lower budgetary overhead.

1.11 Technical Approach Overview

In this section, we provide a technical overview of our proposed cyber agility techniques in this dissertation.

1.11.1 Multi-dimensional, Multi-parameter, and Multi-strategy Host Identity Anonymization

In this work, we propose a cyber deterrence technique for hiding host identities that can deter advanced reconnaissance techniques. This approach, called M-RHM, achieves this by full and multi-parameter anonymization of host identities, mutating host IP addresses over temporal and spatial dimensions, and also a careful composition of various mutation strategies.

M-RHM mutates both internal and public addresses of network hosts. The public addresses are mutated to deter *external* reconnaissance on the DMZ, while internal addresses are mutated to deter insider attacks and multi-stage intrusions.

In our preliminary work, we present a cyber deterrence technique called Random Host Mutation (or RHM) that mutates addresses of network hosts over time. In M-

RHM, in addition to mutating IP addresses only over time, referred to as *temporal* mutation dimension, we also mutate them based on the IP address of the source; i.e., two distinct sources would reach the same host with different IP addresses. This is referred to as *spatial* mutation and means that each host has its unique view of the network, which is different from other hosts. Therefore, information collected on a host could not be used on another host. This is especially effective against reconnaissance sharing and lateral movements in the network.

In M-RHM, we adopt a multi-strategy mutation approach, as a combination of proactive, adaptive, and reactive strategies. The proactive mutation strategy is adopted when no specific scanning pattern is observed in the network, to defeat unknown external and internal scanners. In contrast, adaptive mutation strategy is adopted when the characterization of attackers' scans shows a pattern that suggests which addresses are more probable to be scanned next. The adaptive mutation enhances effectiveness and reduces cost by tuning mutation rate and distribution by considering the attacker's behavior. Finally, the reactive strategy is adopted for scans that are identified to be malicious, which are scans that are not preceded by appropriate DNS query. This strategy defeats internal scanners.

Moreover, M-RHM offers a multi-parameter mutation technique. Specifically, M-RHM mutates hosts' IP address, MAC address, and domain name (for reverse-DNS queries), and more importantly, host fingerprints. To randomize host fingerprints, a group of shadow decoys is dispersed in the address space, based on the paradigm of k-anonymity, to anonymize host fingerprints both in spatial and temporal dimensions. A shadow decoy of a host has the same network-level and application-level fingerprint

as of that host.

Achieving fingerprint anonymization through k-anonymity necessarily means that for each real host in the network, $k-1$ shadow decoys are dispersed in the address space. In M-RHM, a host only mutates its addresses with its $k-1$ shadow decoys, and since all these k machines have the same fingerprint, the status of network address space seems static from outside, and therefore mutations are invisible to attackers.

M-RHM mutates host IP addresses by a conflict-free and synergistic composition of the following mutation strategies.

- *Proactive temporal* [1, 56]: if no identifiable scanning pattern is observed in the network, then addresses are proactively mutated at regular intervals. The proactive temporal is, in fact, the same strategy as that of RHM [1] and OF-RHM [56] in address mutation. These temporal mutations occur both for internal and external address space. This strategy is especially important in defense against external-active and internal-passive scans. However, contrary to our preliminary work, a host only mutates its address with addresses of its shadows.
- *Adaptive temporal* [58]: if a scanning pattern is discovered in the sequence of scans that are observed in the public address space, then M-RHM adapts mutation intervals as a function of the observed scanning rate. This approach is a novel research contribution of M-RHM. Furthermore, M-RHM uses statistical hypothesis testing to learn potential patterns in attackers' scanning. The adaptive temporal strategy is first presented in A-RHM [59], and it increases deterrence and deception against strategic external scanners such as cooperative or local-preference.

- *Proactive Spatial [61]*: the spatial mutation aims to defeat (deter and deceive) information sharing among internal network hosts, by giving them a disjointed set of addresses to reach other internal hosts. Therefore, proactive spatial mutation aims to defeat *internal active and passive* scans by assigning addresses in a manner that when attackers potentially move from a host i to a host j, the information collected at i is only used at j to reach shadows. Spatial mutation is especially effective against advanced multi-stage attacks.

Through rigorous theoretical analysis and simulation, we show that M-RHM can prevent propagation of network worms, provide high resistance against external scanners, deter advanced and collaborative reconnaissance, and provide high resistance against APT attacks.

1.11.2 A Formal Framework for Active Cyber Deception Planning

In this work, we present a formal framework for cyber deception planning. Contrary to passive deception, active cyber deception aims to proactively design and deploy a deception plan, consisting of many small lies to fabricate a convincing *big lie* that is consistent, believable and tempting, even to skilled attackers. We show that such planning needs a framework that allows formal modeling and quantitative reasoning on deception, and there exists no logic or framework that allows one to model how various lies and their different combinations would prohibit an attacker from achieving the desired goal in a certain domain. We show that development of such framework is necessary because the number and quality of lies are large and different; lies have different costs and are often contradictory and therefore selecting a set of lies that

would provide a consistent image of the domain with affordable cost and a high temptation is not possible without modeling their interactions, effects, costs, and conflicts.

The presented deception planning framework provides logic and mechanisms for constructing deception plans for a given domain, and for interacting with the domain to deploy the plan, as well as updating the plan based on domain feedback. The framework will have interfaces for defining a *deception model* for a given domain, using a *deception logic*. The logic is an abstraction over satisfiability modulo theories (SMT) [12] extended by many-valued fuzzy logic [10, 48], which can model perlocutionary aspects of deception (e.g., belief, cause-and-effect, and intention), as well as its quantitative traits (belief certainties, budget, and risk). The framework includes necessary mechanisms for validating and solving the model to generate a sound deception plan, which then will be deployed in the given domain.

The deception logic models a system, using the following components: facts, belief, actions, causality relationship between facts, beliefs, and actions, and also intention (or goal) of deception. This modeling shows how providing various deceptive information pieces to attackers can mislead them into wrong beliefs on the facts. The deception goal is misleading attackers to beliefs that are most beneficial for defense. Therefore, in addition to these generic components of deception, the deception logic includes the following cyber-specific components: benefit/cost quantification for alternative plans, belief certainties, deception cost. These components allow the framework to quantify benefit and cost of alternative deception plans and select the deception plan that has a higher certainty of satisfying the deception goal. The framework

provides components for modeling adversary types and distribution. This allows us to devise deception plans that are effective against various types of attackers with different goals, knowledge and sophistication levels. Finally, the framework provides mechanisms for revising the plan based on new observations about attackers.

1.11.3 Deception Planning against Multi-Stage APT Attacks

To defeat multi-stage intrusion attacks using decoy services, we model the problem of decoy planning against intrusion attacks, where the problem is to determine what decoy systems and each with what configuration must be placed in each network subnet (zone), based on our defense priorities and budget, in order to maximally deflect such attacks from important assets and enable its characterization. This deception plan targets remote network reconnaissance and intrusion attacks.

We consider a variety of decoys in our planning and model how various alternative configurations and placement of these decoys would manipulate attackers' information gathering process and mislead them about which network services must be targeted in next stages of the kill-chain. This is achieved by misleading attackers in evaluating the criticality of various real network services, thus reducing the certainty that the attacker selects them as a target. The resulting plan ensures that such misleading can happen with a high certainty. Through comparison with alternative honeypot planning paradigms, we show that the generated deception plan is effective and scalable.

1.12 Organization

The rest of the thesis is organized as follows: In Chapter 2, we propose *M-RHM*, a multi-parameter, multi-strategy, and multi-dimension technique for hiding host identities from adversarial reconnaissance. The goal is to defeat advanced and persistent attacks by deterring attackers' progress in the network, and increasing their detectability by making their behavior more tractable. We propose metrics for evaluating the model and provide rigorous theoretical and experimental evaluations to manifest the benefit and cost of the technique.

In Chapter 3, we propose a cyber deception planning framework that provides logic, interfaces, and mechanisms for constructing optimal deception plans for a given system, and for interacting with the system to deploy the plan, as well as updating the plan based on the system feedback.

In Chapter 4, we use this deception planning framework to craft a deception plan consisting of a variety of decoy services against multi-stage intrusion attacks. In this model, we show that the framework is expressive and effective and provides plans that are intuitively reasonable.

In Chapter 5, we present a summary of the dissertation, in addition to our plan for future works.

CHAPTER 2: MULTI-DIMENSIONAL RANDOM HOST IDENTITY HIDING (M-RHM)

2.1 Problem Statement

Cyber deterrence aims to establish cyber agility into systems by mutation of system parameters [56, 61, 92]. The basic idea is to change the static nature of a system and establish dynamics in it by randomly and frequently mutating (altering) configurations of the system and its components, such that (1) attackers' premises and assumptions about the system fails, and (2) their collected reconnaissance information regarding the system is constantly deprecated. The main objective is taking away attacker's advantage of being able to study the target system off-line and find vulnerabilities that can be exploited at the attack time.

Examples of mutation for attack deterrence include instruction set randomization [69, 92], memory address randomization [111], and compiler-generated software diversity [55] to avoid attacks such as buffer overflows and worms [111].

Our focus in this dissertation is on disrupting advanced and persistent intrusion attacks, by deterring their reconnaissance phase through cyber agility. The role and significance of cyber deterrence techniques against adversarial reconnaissance in networks have been partially investigated in the literature. For example, to disrupt the reconnaissance activities that exploit the static nature of MAC addresses in wireless networks, mutation of these addresses has been recently incorporated into iOS [91].

Accordingly, to counter scanning and reconnaissance activities that rely on the static assignment of host IP addresses, several IP address randomization techniques, for example, based on DHCP [6] or NAT [70], have been proposed. However, these techniques suffer from limited unpredictability and non-transparent protocols, which confines their effectiveness to naive threat models such as hitlist worms [6] and makes their deployment unjustifiably costly.

To address these limitations, in our preliminary investigation of this problem, we introduced an IP randomization technique, called *Random Host Address Mutation (RHM)* [1,56], that allows one to achieve unpredictable address mutation with transparent deployment in networks. RHM aims to establish dynamics into static networks by reconfiguring the network unpredictably and in an affordable manner.

RHM mutates host addresses proactively, and with a fixed rate. This has two repercussions. First, when there is no reconnaissance (scanning) activity, this results in unnecessary mutations of addresses, which incurs unjustified costs on the system. Second, when an attacker is scanning the network with a high rate, this fixed rate would result in lower deterrence [1], as compared to a higher mutation rate. To address this limitation, we require a mutation technique that can learn an attacker's scanning rate and adapt mutation rates to that.

Also, RHM only mutates addresses uniformly. Uniform mutation is the optimal strategy when no information is available about an attacker's scanning strategy. However, if mutations are adapted to how an attacker is scanning the address space (e.g., sending most scans to certain address ranges and less to others), much more deterrence could be gained [58]. Achieving this requires a mutation technique that can

identify certain patterns in a potential scanning and adapt address mutation to that.

To address these two limitations, in another preliminary work, we proposed an adaptive IP randomization technique, called attacker-aware RHM or A-RHM [58]. A-RHM introduces two techniques for identifying patterns in attackers' scanning. It also introduces a technique for learning attackers' scanning rate and adapting the mutation rate based on that. By adapting mutation strategy and rate to attackers' behavior, A-RHM can provide higher deterrence against scanners.

In spite of their effectiveness against scanners, both RHM and A-RHM suffer from several shortcomings. Firstly, both techniques only mutate IP addresses temporally. This is only effective against less advanced and usually automated threat models such as scanners but fails to defeat more advanced threat models such as intrusion attacks, where an elite human attacker compromises a chain of hosts and laterally moves in a stepping-stone manner until it reaches a target. Defeating such threat model require a mutation technique that can disrupt these lateral movements inside our network. Such technique must mutate host IP addresses spatially, such that each host has its unique view of the network (i.e., its own set of addresses to reach other hosts). Moreover, these views must be created in coordination and concerning an attacker's potential or observed behavior such that maximum deterrence and deception is incurred on the infiltrating attacker.

Secondly, mutations of IP addresses in both RHM and A-RHM are only effective against less advanced and usually automated scanners, but fails to defeat reconnaissance launched by elite human attackers; this is because merely mutating IP addresses of hosts is not enough to hide them from an elite attacker, because such an attacker

would use other attributes of a host, such as its fingerprint [60], to identify it later. Defeating such attackers require a mutation technique that can anonymize other identifying attributes of a host, especially its network fingerprint.

In this chapter, we propose a multi-dimensional, multi-parameter, and multi-strategy host identity hiding or anonymization technique, called M-RHM, that addresses these limitations by combining a group of proactive, adaptive, and reactive mutation strategies to deter advanced reconnaissance. M-RHM integrates previous strategies of RHM and A-RHM with some novel mutation strategies and dimensions in a synergistic and consistent manner.

Before introducing M-RHM, we provide an overview of existing work on IP mutation in the literature, followed by our preliminary works RHM [1,56] and A-RHM [58].

2.2 Related Work

Mutation, also known as randomization or hopping, is a common technique for security. Examples of such techniques include using the one-time virtual number instead of the real credit number for online transactions; instruction set randomization [69, 92], memory address randomization [111], compiler-generated software diversity [55], and end-to-end software diversification of Internet services [23]. However, our focus in this dissertation is on cyber deterrence techniques against network reconnaissance, which are mainly focused on IP mutation.

2.2.1 Literature Review of IP Mutation

Mutation of IP addresses (also called address shuffling or IP hopping) to disrupt threat models such as sniffing and man-in-the-middle attacks, scanners and worms,

as well as network reconnaissance has been investigated in the literature.

The APOD (Applications That Participate in Their Own Defense) scheme [9] uses *hopping tunnels* based on address and port randomization to disguise the identity of end parties from sniffers. However, this approach is not transparent as it requires the cooperation of both client and server hosts during the IP mutation process.

DyNAT [70] provides a transparent approach for IP hopping by translating the IP addresses before packets enter the core or public network to hide the IP address from man-in-the-middle sniffing attacks. Although this technique will make network discovery infeasible for sniffers, it does not work for scanners who rely on probe responses for discovering the end-hosts.

NASR [6] offer an IP hopping approach that can defend against hitlist worms. NASR is a LAN-level network address randomization scheme based on DHCP update. It is not transparent to the end-hosts because DHCP changes are applied to the end-host itself which results in disruption of active connections during address mutation. Moreover, it requires changes to the end-host operating system which makes its deployment very costly. Also, NASR provides very limited unpredictability and mutation speed because its IP mutation is limited on the LAN address space and will require DHCP and host to be reconfigured for this purpose.

Yegneswaran et al. [131] and Cai et al. [15] present techniques for defending honeynets from systematic mappings that aim at differentiating live IPs from monitored ones and blacklisting monitored IPs for efficient target selection. Their approach is based on shuffling monitored IP addresses with live IPs after the number of probes received in the honeynet has exceeded a threshold.

Several approaches have suggested techniques for mutation of IPv6 addresses [130]. While IP scanning attacks are not possible in IPv6 due to the abundance of addresses, sniffing attacks are still viable. To protect a specific flow between two endpoints against sniffing attacks, MTD6 [67] rotates addresses of the sender and the receiver mid-session without dropping or renegotiating sessions. The design takes an advantage of IPv6 networks allowing nodes to seamlessly bind new IPv6 addresses. MT6D creates dynamic Interface IDentifier (IID) obscuration to create dynamic IP addresses. In contrast, DHCP6D [33] use DHCP for address mutation to avoid the cost of address generation and verification. Another approach called MOB6D [51] protects mobile devices against sniffing and reconnaissance attacks by randomizing the prefixes of IPv6 addresses.

In a similar work, Albanese et al. [2] propose a mechanism for periodically changing the virtual identity of nodes in a MANET to defeat the attacker's reconnaissance efforts. The approaches are transparent to end-hosts but do not consider spatial mutation which restricts their applicability against coordinated attack models.

SDNA [130] inserts a hypervisor within each network node to make host appearance dynamic to observers while at the same time retaining transparency to OS, applications, and end-users. The main shortcoming of SDNA architecture emanates from the costly reconfiguration of each network node to include hypervisor technology. Furthermore, SDNA is only designed for IPv6 technology.

Several other works have focused on providing a general theory [134], framework [19, 130], and comparison/evaluation [86, 129] methodologies for moving target defense, using IP randomization as a mutation vector.

IP Mutation has also been used by botnets [52] to hide phishing and malware delivery sites behind an ever-changing network of compromised hosts acting as proxies. The basic idea behind Fast flux is to have numerous IP addresses associated with a single fully qualified domain name, where the IP addresses are swapped in and out with extremely high frequency, through changing DNS records. The simplest type of fast flux named "single-flux", is characterized by multiple individual nodes within the network registering and de-registering their addresses as part of the DNS A (address) record list for a single DNS name [14]. This combines round robin DNS with very short - usually less than three minutes (180s) [14] - TTL (time to live) values to create a constantly changing list of destination addresses for that single DNS name. A more sophisticated type of fast flux referred to as "double-flux", is characterized by multiple nodes within the network registering and de-registering their addresses as part of the DNS Name Server record list for the DNS zone [14]. This provides an additional layer of redundancy and survivability within the malware network. Existence and widespread success of such botnets demonstrate the feasibility and effectiveness of mutating addresses of network hosts via DNS.

2.2.2 Proactive RHM

In [1,56], we propose a technique called Random Host Mutation or RHM that offers a two-level IP mutation approach (low-frequency and high-frequency) to maximize unpredictability. The approach is transparent to end-hosts, but the mutation is only performed uniformly, and without considering the adversarial behavior. In [56], we present a variation of RHM for software-defined networks and introduce a technique

called OF-RHM (OpenFlow Random Host Mutation). In this section, we provide a brief overview of this preliminary work.

2.2.2.1 Methodology Overview

In Section 2.2, we introduced several IP mutation works in the literature. However, these works suffer from several shortcomings, including low agility (limited unpredictability and slow mutation), and high cost (lack of transparency to network hosts and network sessions). These limitations make them only effective against naive threat models. Compared to previous approaches in the literature, RHM offers a scheme that achieves substantially high unpredictability, minimizes reconfiguration overheads, and is transparent to all network hosts and sessions.

To achieve these goals, RHM mutates IP addresses of network hosts randomly and frequently, to make them untraceable. To minimize overhead on network hosts, RHM keeps the actual IP addresses (referred to as *rIP*) of hosts unchanged. Instead, it creates routable short-lived ephemeral IP addresses (eIP) that are chosen from the unused ranges of the network. The eIP addresses will be used for routing and are automatically translated into the rIPs and vice versa at the network edges (subnet) close to the destination. The new eIP addresses are provided to clients via DNS, similar to fast-flux techniques [14] that are used extensively by botnets for stealthiness.

Maximum unpredictability is achieved when each eIP address is chosen (I) *uniformly* and (II) from the *largest* possible unused address space. The largest possible mutation space consists of all unused ranges of the network. However, it is practically impossible to sample each eIP from all unused ranges at once, because each unused

address range can only be routed to one physical subnet at any given time. Therefore, maximizing unpredictability is not practically possible. However, to achieve substantially high unpredictability while addressing this practical limitation, RHM uses a two-level mutation scheme; namely low frequency mutation (LFM) that changes the set of unused ranges assigned to each host, and high frequency mutation (HFM) that changes the eIP address associated with each host by sampling a new address from these unused ranges. To achieve uniform sampling, ranges must be assigned to hosts (for each LFM interval) such that the distribution based on which eIPs are chosen has low divergence from the uniform distribution in a long interval.

To address another objective, i.e., minimizing reconfiguration overheads, note that when new ranges are assigned to each host (for each LFM interval), routing tables must be updated accordingly. Assigning ranges to hosts in a completely random manner enhances unpredictability, but may substantially increase the routing table sizes and affect their performance. Thus, to bound the size of the routing tables, RHM preferably assigns adjacent ranges to same physical subnets, so that the supernetting allows us to decrease routing table sizes. Therefore, RHM determines the new range to host assignment for each LFM interval by considering the trade-offs among unpredictability, adaptability, and performance. This problem is formally modeled as a constraint satisfaction problem using Satisfiability Modulo Theories (SMT) [12].

For deployment, we propose plug-and-play architectures and communication protocols for seamless and incremental deployment of RHM on the legacy as well as software-defined networks, without requiring any changes in the end-host or legacy network protocols. This seamless integration allows for separating network adminis-

tration from mutation management, making mutation transparent to administrators and end-host configurations. Moreover, it allows RHM to achieve very fast mutation rates, since transparency to end-host means that old TCP/UDP sessions are not disrupted by new mutations. Therefore, mutating a host eIP only affects forthcoming sessions, while old sessions can continue communicating using their old addresses.

2.2.2.2 Architecture and Protocols

Deployment of RHM in legacy networks requires a set of components and protocols to handle mutation planning and translation. We deployed RHM on two network types: legacy TCP/IP networks [1], and software-defined networks (SDN) [56]. While deployment of RHM on legacy networks is straightforward, software-defined networking (SDN) provides a more flexible infrastructure for developing and managing random host mutation efficiently and with minimal operational overhead.

Deployment of RHM in a network requires components to handle three distinctive responsibilities: (1) *LFM planning* to determine the mutation space of each host, and (2) *HFM planning* to determine eIP of hosts, and (3) *eIP-rIP translation* to translate rIPs to/from eIPs during communication. Depending on the underlying network infrastructure, these responsibilities will be carried out by different components of the network.

In legacy TCP/IP networks, two types of components are added to the existing architecture [56]: a central controller, called *MC* and a set of distributed gateways, called *MG*, which are located at the boundaries of physical subnets.

Figure 2 shows the architecture of these components in the network. Within this

architecture, the aforementioned responsibilities are handled in the following manner: (1) *LFM planning* is performed by MC. The new ranges are announced to MGs which are located at the boundary of subnets (between subnet switch and the core). MGs are then responsible for broadcasting new ranges to the routers and updating the routing tables of the network. To communicate with network routers, MGs must use the interior gateway protocol (IGP) of the network; (2) *HFM planning* is performed by MGs for all the hosts in their corresponding subnets, and (3) *translation* is performed by MGs for all inbound and outbound connections. For translation, MG must perform two distinctive tasks: updating source and destination IP addresses of packets, and updating DNS replies issued by the authoritative DNS of the network to reflect the new eIP associations. In addition to these responsibilities, MC is also responsible for authorization of rIP-based flows.

For scalability, this architecture (as well as the SDN architecture) could be extended to include several controllers, each managing a segment of the network. Each controller has full autonomy in the management of its designated network segment because no information needs to be distributed among controllers. The only constraint is that the mutation space must be divided between these controllers. This can hurt unpredictability but substantially improves scalability.

Moreover, although it is preferable to assign one MG to every physical subnet, it might not be cost-effective. In this case, it is possible to assign an MG to handle responsibilities for several subnets, with the drawback that network traffic would be more exposed to passive reconnaissance attacks such as sniffing because eIPs are translated to rIPs several hops before the packets reach the final subnet.

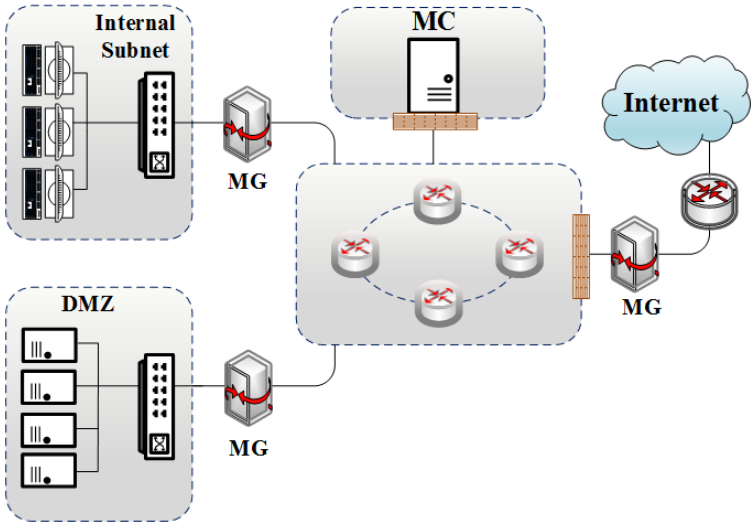

Figure 2: RHM architecture on legacy networks

In SDN, the network controller (e.g., POX [81]) monitors and controls the entire network from a *central* vantage point via an interface, such as OpenFlow [83], and defines the forwarding and address translation behavior of switches distributed in the network accurately and synchronously. Accordingly, the aforementioned responsibilities are performed in the following manner: (1) *LFM and HFM planning* are both performed by the SDN controller. After HFM planning, the controller installs appropriate flows in the switches along the way to handle necessary eIP-rIP translations; (2) *translation* is performed by SDN switches of the network and according to the flow actions determined by the controller.

Since the eIP addresses of network hosts are periodically mutated, a host in RHM network must be reached via its name and through DNS. Figure 3 depicts the general outline of RHM protocol for communication via name in an RHM-enabled legacy network. For SDN, the protocol is the same, but translations are performed by OF-switches of physical subnets.

To communicate with a server using its name, a DNS query is sent to resolve the name of the server. The DNS response is intercepted by the responsible entity (i.e., MG in conventional networks and controller in SDN), and the rIP of the server is replaced with its corresponding eIP (steps 1-3). Moreover, the TTL of the DNS response is updated based on to the HFM interval duration. As a result, clients will receive the eIP of the destination host and initiate their connections accordingly (steps 3-4). The packets are routed to this eIP as the destination address (step 5). Finally, when the packet arrives at the destination subnet, destination eIP is replaced with its corresponding rIP (step 6). This translation is performed for as long as the flow continues. More importantly, future mutations do not affect a previously established flow. The packets of the flow will be updated and forwarded until the session is terminated (FIN for TCP) or expired (long inactive time for both TCP and UDP). Therefore, new mutations do not affect previously established sessions.

A host can still be reached via its rIP, but only by authorized users. In fact, unnamed network hosts can only be reached via rIP. In this case, the access request must be authorized by the controller. If access is granted, the rIP of the destination is translated to its corresponding eIP. The authorization is handled based on the access control policy of the network.

For SDN, the controller is responsible for handling communications via name and rIP. The general algorithm of POX controller for communication management is presented in [56]. OF-switches are configured to encapsulate unmatched packets (that have no matching flows in flow tables) and send them to the controller. The controller determines the type of connection (i.e., via rIP or eIP) and installs necessary flows in

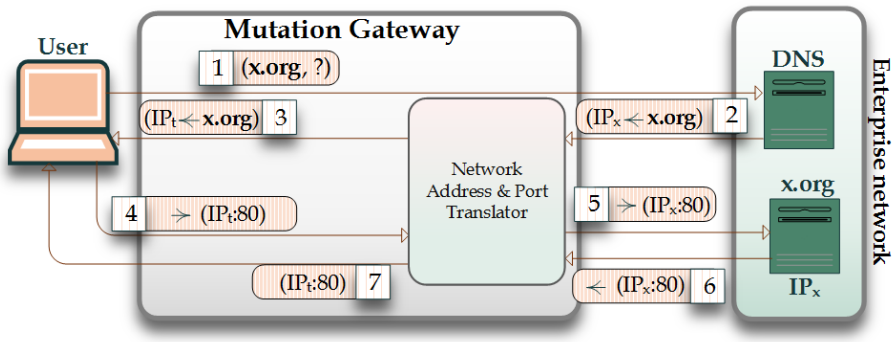

Figure 3: Communication protocol using host name

all OF-switches in the path. Each connection must be associated with a unique flow because the rIP-eIP translation changes for each connection. This property guarantees the end-to-end reachability of hosts because the rIP-eIP translation for a specific connection remains unchanged regardless of subsequent mutations.

When the LFM interval expires, some eIPs might not be still released. RHM uses the longest prefix match policy of routers to temporarily exclude these eIPs from their corresponding ranges until they are released. Specifically, specific entries are added to the routing tables of the network for these unreleased eIPs. As a result, packets destined to an unreleased eIP will be routed to its current host, while packets destined to the rest of addresses in the range are available to be assigned and routed to another host. Without this scheme, an attacker will be able to deplete the pool of available ranges by establishing many long-lived connections.

2.2.2.3 Evaluation of Proactive RHM

Scanners and worms (or any other automated malware) use various scanning techniques to identify network hosts. From RHM perspective, scanning techniques can be classified into two main categories: (1) uniform scanning, and (2) strategic scanning.

In uniform scanning techniques, the scanner uniformly selects addresses from the address space and probes them. However, uniform scanning generates tremendous noise which makes it highly susceptible to detection [135]. Strategic scanning techniques, such as cooperative or local-preference scanning take advantage of network dynamics to improve their effectiveness while reducing the volume of generated traffic [135]. Therefore, contrary to uniform scanning, strategic scanning techniques have a higher possibility of evading detection.

In this section, we investigate and quantify the effectiveness of our approach to scanning techniques. In order to determine optimal scanning strategy, we can model the interaction between a scanning attacker and an RHM-enabled network as a static game, where optimality is defined as follows: a scanning strategy is designated as optimal (by the attacker), if for a fixed number of probes it results in the discovery of a higher number of hosts in the network. This game-theoretic analysis, the details of which are omitted from here for brevity, leads to the following conclusions. Firstly, for an RHM network, the optimal scanning strategy for an attacker during a mutation interval is *uniform sampling without replacement*, which is known as cooperative scanning [22, 135]. However, after each mutation interval, the attacker is forced to restart his scan. Using any other scanning strategy does not increase attacker's payoff. Moreover, in the case where an attacker deviates from the uniform strategy by using a biased scanning algorithm such as local-preference, the defender has the opportunity to potentially increase her deterrence.

Secondly, for a static network, cooperative scanning is the most effective scanning strategy, because a static network is analogous to an RHM network with an infinite

mutation interval.

Also, the game-theoretic analysis shows that the optimal mutation distribution for eIP selection is uniform. This necessarily means that deployment of RHM in a network will force a rational attacker to commit to uniform scanning, which is highly susceptible to detection [119]. In the next section, we theoretically and experimentally evaluate the effectiveness of RHM against well-known non-uniform scanning strategies: (1) cooperative scanning as the optimal scanning technique for static networks, and (2) local-preference scanning as non-uniform scanning techniques, which are shown to be more effective than the uniform scanning in static networks.

Cooperative Scanning. In this section, we aim to quantify the deterrence that is exerted by RHM on the most optimal scanning strategy for static networks, which is the cooperative scanning. Sophisticated scanners usually aim to minimize connection failures by avoiding repeated probing of the same IP address [135] because in a static network the status of an address does not frequently change in a relatively long interval.

Assume a group of $k \geq 1$ scanners are scanning the address range of our network with m addresses and n moving hosts cooperatively, and the mutation interval of all hosts is λ.

For a designated target, the expected number of scans to hit the target is $(m-1)/2$, while for a uniform scanner the expected number of scans to hit the target is the mean of a Bernoulli trial, which is m.

Assume η denotes the scanning rate of each scanner. We define $\Gamma = \eta\lambda$ as the ratio of attacker scanning rate on defender mutation rate, where mutation rate is $1/\lambda$. If

the scanning rate is the same as the mutation rate (i.e., $\Gamma = 1$), for every scan the scanners will miss the target with probability $(1 - \frac{1}{m})$. For m scans, the scanner will miss each target with probability:

$$P_{miss} = \left(1 - \frac{1}{m}\right)^m \approx e^{-1} = 0.37$$

If scanners only make m' ($m' \leq m$) scans, the miss probability is:

$$P_{miss} = \left(1 - \frac{1}{m}\right)^{m'} \approx e^{-m'/m} \tag{1}$$

This value is greater than e^{-1} if $m' < m$. For example, if scanners only scan half of the address space, then $e^{-\frac{1}{2}}$ ratio of hosts will be missed.

Suppose the scanning rate (η) is higher than mutation rate $1/\lambda$; i.e., scanning is faster than mutation ($\Gamma > 1$). Assuming that scanners make a total of m' probes, the defender on average mutates after every Γ scans. Therefore, the scanning duration could be divided into $\lceil m'/\Gamma \rceil$ intervals. The first probe will miss the target with probability $1-1/m$. However, the second probe will miss with probability $1-1/(m-1)$, and so forth. In general, during each interval scanners will miss the target with the following probability:

$$P_0 = \left(1 - \frac{1}{m}\right)\left(1 - \frac{1}{m-1}\right)\ldots\left(1 - \frac{1}{m-\Gamma+1}\right)$$

Therefore, after m' probes the scanner will miss the target with the following probability:

$$P_{miss} = P_0^{\lceil \frac{m'}{\Gamma} \rceil} \tag{2}$$

When $m' >> \Gamma$, i.e., the rate ratio is far smaller than the number of probes, $(m-k) \simeq m$ and therefore $P_{miss} \simeq e^{-m'/m}$. On the contrary, when $m' \simeq \Gamma$, the mutation has no effect on scanning since the hosts are stationary during an interval, and therefore P_{miss} approaches $1 - m'/m$.

Figure 4 shows the theoretical and experimental mutation success probability of RHM with $m = 2^{20}$, 2^{21} and 2^{22} and different Γ values. The x-axis shows the logarithm of Γ (scanning rate over mutation rate) to base 2. The scanner makes a total of $m' = 2^{20}$ scans. We can see that the experimental result is roughly consistent with the theoretical analysis. Moreover, note that the mutation can save 40% to 80% of network hosts from infection. Also note that when $\Gamma << m'$, the mutation success probability remains close to $e^{-m'/m}$. However, as Γ approaches m', the mutation success probability drops to $1 - m'/m$. In fact, when $m' = m = \Gamma$ the mutation success probability becomes 0.

Our analysis up to here only focuses on scanners; i.e., the number of scanners does not change. However, in case of a cooperative worm, each newly-infected machine starts acting as a scanner. To analyze the effect on cooperative worms, we use mathematical propagation modeling [45]. Assume $S(t)$ and $I(t)$ represent the susceptible and infected populations at time t, respectively. Parameter η denotes the scanning rate of each scanner. In a static network, it is straightforward to see that [1]:

$$\frac{dI(t)}{dt} = \frac{n}{m}\eta I(t). \tag{3}$$

For RHM network, assume $Q(t)$ represents the quarantined population at time t; i.e., susceptible hosts that are not accessible to scanners as a result of mutation.

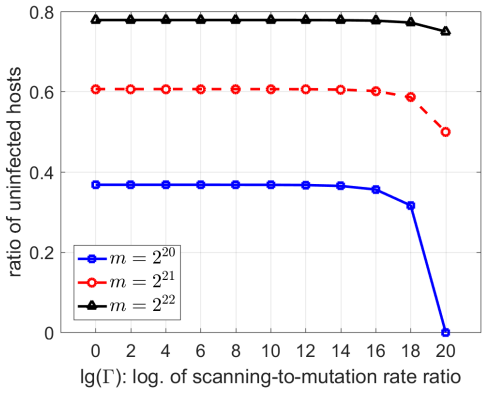

Figure 4: ratio of uninfected hosts against cooperative worms

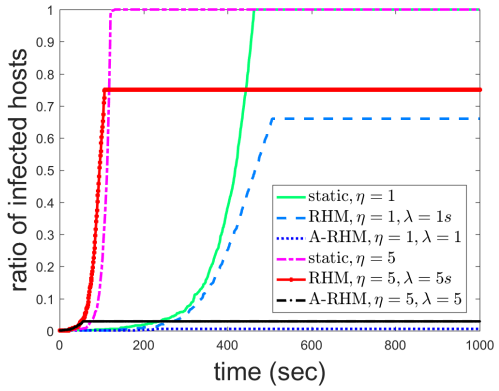

Figure 5: propagation of cooperative worm in various network types

With respect to the infected population, the probability that a probe targets a host is $\frac{((n-Q(t)))^2}{m}$ and the probability of probe success is $(n - Q(t))$. Therefore:

$$\frac{dI(t)}{dt} = \frac{(n - Q(t))}{m}\eta I(t) \tag{4}$$

On average every $(n - I(t))$ unsuccessful scans move one host to quarantined state with probability $\frac{((n-I(t))^2}{m}$. The probability emanates from the fact that some of the hosts may already be infected, and so only the portion of hosts which are not infected must be considered. Therefore, the propagation rate of quarantined hosts is:

$$\frac{dQ(t)}{dt} = \frac{(n - I(t))}{m}\eta I(t) \tag{5}$$

Assuming that the scanning stops once the whole address space is probed, numerical analysis of these equations (using Matlab Symbolic Math Toolbox) shows that as t grows $I(t)$ approaches $(1-e^{-1})n$, while $Q(t)$ approaches ne^{-1} which is the same result that is achieved for non-propagating scanners.

Figure 5 shows the simulated propagation of a cooperative worm in a static network, an RHM network, and a network with adaptive mutation (A-RHM). First, note that

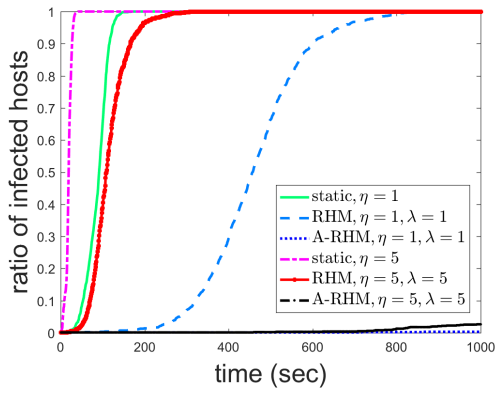

Figure 6: propagation of local-preference worms

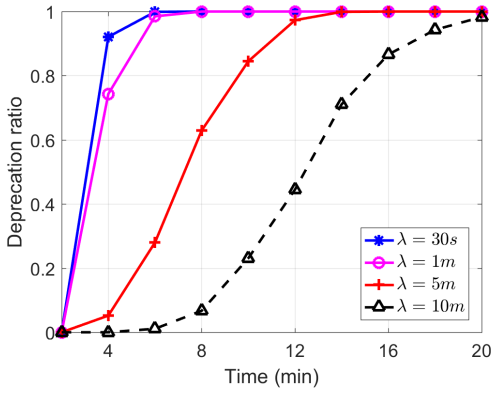

Figure 7: deprecation ratio for various mutation rates

the propagation of a cooperative worm in a static network is very fast, even when the worm is scanning the network with a low rate of 1 scan per second. Secondly, note that when $\lambda = 1$ and the scanning speed of the worm is $\eta = 1$ scan per second, RHM can still rescue 40% of network hosts from infection. However, when the worm is scanning the network with $\eta = 5$ scans/sec and the $\lambda = 5$ seconds (one mutation every 5 seconds), only 12% of network hosts evade infection in an RHM network. However, note that proactive temporal achieves far less deterrence than the adaptive temporal strategy, as will be discussed in the next section.

Local-Preference Scanning. This scanning strategy aims to increase propagation speed by considering the distribution of hosts in the network. When vulnerable hosts are not uniformly distributed in a worm's scanning space, local-preference scan increases its propagation speed [135].

The local-preference scanning works as follows [119]: a probe issued by an infected host is targeted toward a local subnet $a.b.c.d/k$ with probability p; otherwise, with probability $1 - p$ the probe is targeted toward a random address that is uniformly

chosen from the whole address space.

In static networks, hosts' addresses are not uniformly distributed in the address space. Rather, hosts in the same subnet have the same network address. In a typical scenario, assume our network consists of /22 subnets, and network hosts' rIPs are concentrated in a few subnets. The local-preference worm scans the local /22 subnet with probability p.

Figure 6 shows the propagation of a local-preference worm in various network settings with $m = 2^{20}$, and $n = 2^{16}$. Firstly, note that in static networks the local-preference worm propagates very fast because hosts are concentrated in certain ranges of the network and host address distribution is biased. Secondly, note that RHM can decrease the worm's propagation speed by uniformly distributing hosts in the address space. However, the effectiveness of proactive temporal is less than that of adaptive temporal strategy, as will be discussed in the next section.

Network Reconnaissance. RHM disrupts reconnaissance by fast deprecation of network addresses. Changing the addresses of network hosts invalidates existing mappings of network hosts, thus forcing potential attackers to squander their resources on re-discovery of these mappings.

The disruption of reconnaissance information has significant repercussions for many attack vectors. For example, *Hitlist scanners/worms* scan a pre-generated list of addresses to minimize connection failures [6]. By affording very fast mutations, RHM maximizes disruption that is exerted on hitlist scanners. *Honeynet mapping* techniques aim to systematically identify and blacklist the addresses that are monitored [15, 131]. RHM completely thwarts honeynet mapping, because the addresses

are transient and the mutation can be very fast.

Network reconnaissance, in essence, could be disrupted using a naive DHCP-based randomization, such as NASR [6]. Two factors differentiate RHM from previous approaches. Firstly, RHM allows significantly faster mutations than other approaches. For example, while a DHCP-based randomization can only mutate a host address every $10 - 15$ minutes [6], RHM can achieve a mutation rate of 10 seconds. This will considerably raise the bar for attackers since they only have a very short interval to update their reconnaissance information and launch the attack. Moreover, contrary to DHCP-based mutations, address mutation in RHM does not disrupt previous (TCP/UDP) sessions.

Figure 7 shows the deprecation of reconnaissance information from an attacker's perspective over time. The deprecation ratio is defined as the proportion of addresses that are no longer active, as compared to a ground truth (preliminary scanning). To quantify deprecation ratio, we used our RHM-enabled SDN implementation. The address space is a class B and includes $n = 1,000$ hosts. We scanned the whole address space every 2 minutes and over 20 minutes using Nmap [78] and calculated the ratio of deprecated addresses by comparing the active addresses to those active addresses observed during the first scan. In the figure, λ denotes the mutation interval of network hosts. Note that the smaller the interval λ, the faster the deprecation rate of the information.

Secondly, and more importantly, RHM achieves a substantially high unpredictability. It is intuitive that the robustness of a defensive scheme against reconnaissance depends on its unpredictability: the smaller the mutation space from which addresses

of a host are chosen, the easier for the attacker to identify the host. The two-level mutation of RHM allows us to change mutation ranges as well as host addresses. Therefore, in a long interval, the addresses of all hosts are being chosen from the whole address space, thus maximizing the unpredictability.

However, RHM only mutates IP addresses of network hosts. While this is the first step toward anonymizing the identity of network hosts over time, to prevent an attacker to recognize a host from previous reconnaissance, it does not stop the attacker from using other potentially identifying attributes of a host to distinguish it later. For example, if only one host in the address space is running *Apache Tomcat* Web server, an attacker could use this unique feature to identify that host later, even after its IP address has been mutated.

2.2.3 Adaptive RHM

RHM mutates address of network hosts proactively. This means that new addresses are uniformly selected from the address space regardless of whether any scanning activity is observed or not, and also addresses are mutated at fixed, regular intervals.

In the previous section, we showed that when we have no information about what scanning strategy (distribution) an attacker may adopt, uniform mutation is the optimal strategy for RHM. However, what if we can observe some patterns in the sequence of scans that are observed in the address space? Then, a better strategy is to adapt our eIP selection strategy for mutation to these patterns.

Also, having fixed mutation intervals is not very effective, because (1) when there is no scanning activity on the address space, this results in unnecessary mutations thus

increased cost; and (2) when there is a significant number of scans observed on the address space, more deterrence could be achieved by increasing the mutation interval. In the previous section, we evaluated the inter-relationship between an attacker's scanning rate and the mutation rate and showed that the optimal mutation rate is when it is equal to the attacker's scanning rate.

To address these limitations, in [59] we introduce an adaptive RHM technique called A-RHM that can adapt both mutation eIP selection strategy and mutation rate based on observed scanning activities on the dark address space of the network. A-RHM uses the same architecture and communication protocol as RHM (Sec. 2.2.2) and we do not mention the details again for brevity.

2.2.3.1 Methodology Overview

In A-RHM, observable adversarial behavior such as scans to unused addresses and ports are used as a mutation parameter that determines the rate and strategy for mutating hosts' addresses. In this work, we introduce two different paradigms for adaptive mutation planning, based on statistical hypothesis testing [58]. To achieve a fast and accurate characterization of adversarial scanning strategies, we observe the sequence of unsuccessful scans that are generated by network hosts and estimate their distribution using hypothesis testing. We introduce two hypotheses. The first hypothesis, *non-uniformity test*, analyzes these scans to determine whether they are skewed toward certain ranges of the network. If the hypothesis is accepted, hosts are evacuated from these ranges probabilistically, thus reducing the probability that a host is discovered by the attacker. The second hypothesis, *non-repetition test*,

analyzes the probes to determine whether a potential attacker is scanning the network while avoiding or minimizing repeated probing of the same address. If this hypothesis is accepted, network hosts are evacuated from addresses that have a high probability of being scanned shortly.

Non-Uniformity Test. In various classes of scanning, for example, local-preference, sequential, and divide-and-conquer, the majority of sampled IP addresses by each scanner are chosen from certain regions of the network. In fact, it has been shown that when hosts are not uniformly distributed in the address space, biased scanning techniques such as local-preference scan increases a worms propagation speed [135]. Moreover, local scanning reduces the possibility of a probe being investigated by a security device such as ID, and thus enhances evasion probability.

Quick identification of highly-scanned ranges allows us to probabilistically move hosts out of these ranges thus decreasing the number of successful probes. This will slow down scanning and worm propagation. To determine whether a potential attacker is scanning uniformly or not, we define a non-uniformity test. The objective of this non-uniformity test is to investigate the one-tailed null hypothesis that the distribution of destination addresses of failed scans is *not* uniform.

With this aim, A-RHM records the addresses that are probed and *failed*, i.e., the scanned address is not assigned to any host. The reason we only take failed connections into consideration is that the number of failed connections generated by a random scanner is significant enough to characterize its distribution. Moreover, legitimate connection requests might be generated by legitimate services running on the hosts or might be generated by the attacker for evasion [119]. So ignoring them

allows us to characterize the scanning behavior of attackers more accurately. The reason we only consider probes to internal addresses is that we only need to determine the weights of our network ranges.

The mutation space is denoted as $\Omega = \{r_1, \ldots, r_M\}$, which are fixed-sized unused ranges of the network, each with size L. After observing a sequence of failed connections, c_k denotes the number of failed connections to range $r_k \in \Omega$. To test the null hypothesis stating that the frequency distribution of failed connection events (over ranges) is consistent with uniform distribution, we use Pearson's χ-squared test. This test defines a measure of goodness of fit as the sum of differences between observed and expected outcome frequencies.

Assume χ^2_ℓ denotes the goodness of fit value after observing ℓ events. For uniform distribution, each range receives ℓ/M scans on average, where M is the number of ranges.

$$\chi^2_\ell = \sum_{k=1}^{M} \frac{(c_k - \ell/M)^2}{\ell/M} \tag{6}$$

It is straightforward to see that the degrees of freedom χ^2 is $M - 1$. The resulting value is compared to the chi-squared distribution with $M - 1$ degree of freedom to determine the goodness of fit, with a significance level β ($\beta \geq 0.05$). Specifically, a χ-squared probability (p-value) of less than or equal to β necessarily means that the sample's deviation from uniform distribution is very significant:

$$P(\chi^2_\ell) = \begin{cases} \leq 0.05 & \text{accept the hypothesis} \\ > 0.05 & \text{reject the hypothesis} \end{cases} \tag{7}$$

Note that χ^2_ℓ can be calculated sequentially and upon arrival of each new event. Assume $(\ell+1)^{th}$ event is a probe destined to r_k, and c_k denotes the number of probes to r_k among the last ℓ events. $\chi^2_{\ell+1}$ can be calculated in $O(1)$ using the following equation:

$$\chi^2_{\ell+1} = \chi^2_\ell + 2(c_k - \ell/M) + 1 \tag{8}$$

$$c_k = c_k + 1 \tag{9}$$

Rejection of the null hypothesis necessarily means that the probes of a potential scanner are mostly concentrated in certain ranges of the network. Therefore, by moving hosts to those ranges with lower probabilities, the defender can (1) increase the probability of failed connections, thus enhancing an attacker's detectability, and (2) reduce probing success rate, thus slowing down the attack. Therefore, a variation of Chauvenet's criterion is used to update the weights that are associated with ranges:

$$w_k = \begin{cases} 0 & c_k \leq \ell/M \\ \frac{1}{\xi}\min(\frac{\pi_k - \ell/M}{M^2/12}, \xi) & \text{otherwise} \end{cases} \tag{10}$$

where ξ denotes the minimum deviation for a range to be have maximum weight. In our experiments, we assume that $\xi = 2.0$. Basically, if c_k for a range has more than $\xi \cdot \sigma$ distance from the average, it will have $w_j = 1$. If a range has negative or zero deviation from ℓ/M, it will have $w_j = 0$.

The probability distribution function π_1 denotes the scan probability characterization over address space based on this test. Given L as the size of each range, and w_j as the weight of range r_j, and W as the summation of all weights, every address

$x_k \in r_j$ in that range has a probability of $w_j/(L \cdot W)$ to be scanned.

$$\pi_1(x_k) = \frac{w_j}{L. \sum_{i=1}^{M} w_i}; x_k \in r_j \tag{11}$$

Such adaptability may allow an attacker to evacuate network hosts from certain network ranges. When the number of these hazardous ranges becomes high, the attacker has a higher probability of finding hosts in non-hazardous ranges. Without loss of generality, assume the attacker intends to exclude at least *half* $(M/2)$ of ranges from mutation space. To this aim, suppose half of the ranges receive c_1 failed probes, and the attacker sends c_2 failed scans to the other half. The total number of scans is denoted as $l = \frac{M}{2}c_1 + \frac{M}{2}c_2$. For a range to be excluded from mutation space (with $\xi = 2.0$), the following condition must hold:

$$\frac{\pi_1 - l/M}{M^2/12} \geq 2 \rightarrow c_1 \geq c_2 + M^2/3 \tag{12}$$

Therefore, the attacker has to generate $\frac{M}{2}c_2 + \frac{M^3}{6}$ scans in order to exclude half of ranges from mutation space. This value is significantly high, even for small networks. For example, for $c_2 = 64$ and $M = 2^{10}$, the attacker has to generate almost 180 million failed scans. For excluding $M - 1$ ranges from mutation space, he has to generate almost $M^4/6$ failed scans, which for the aforementioned example adds up to almost 183 billion scans! This shows that manipulation of A-RHM non-uniformity test requires an impractically high number of failed scans.

Non-repetition Test. The objective of this test is to investigate the hypothesis that a potential attacker is sampling (probing) the address space *without replacement* or with limited replacement. This means that the attacker tends to avoid or minimize

multiple probes to a single address. To test this hypothesis, we calculate the deviation from the average number of probes to each *scanned* address. If this deviation is close to 0, it necessarily means that addresses have been probed an almost equal number of times, and the scanning is without replacement (or it is very limited).

Assume μ_k denotes the number of probes to the k^{th} address, ℓ denotes the total number of failed probes, and m denotes the number of unused addresses in the network; i.e., $m = ML$. If the non-uniformity hypothesis is accepted, then we only consider addresses that belong to hazardous regions. If it is accepted, we consider all addresses in the address space.

Also, assume $\overline{\mu_\ell}$ denotes average number of scans to each address, after observing ℓ scans. Upon receiving the $(\ell+1)^{th}$ probe to the k^{th} IP address, we have $\mu_k = \mu_k + 1$. Assuming $M_{2,\ell} = \sum_{i=1}^{m}(\mu_i - \overline{\mu_\ell})^2$, the unbiased variance of the sample is recursively calculated in the following manner:

$$M_{2,\ell+1} = M_{2,\ell} + 2 \cdot \mu_k + 1 - \overline{\mu_\ell} - \overline{\mu_{\ell+1}} \tag{13}$$

$$S^2_{\ell+1} = \frac{M_{2,\ell+1}}{m-1} \tag{14}$$

For each ℓ events, we define the coefficient of variation (normalized deviation) as:

$$\hat{o^\ell} = \frac{S_n}{\overline{\mu_\ell}} \tag{15}$$

The one-tailed null hypothesis is that the attacker is scanning the address space with

no or limited repetition:

$$\hat{o^{\ell}} = \begin{cases} \leq 0.001 & \text{accept the non-repetition hypothesis} \\ > 0.001 & \text{reject the non-repetition hypothesis} \end{cases} \tag{16}$$

Accepting the null hypothesis means that the addresses that have been probed fewer times than the average ($\overline{\mu^{\ell}}$) are more probable to be scanned shortly.

Therefore address x_k is marked as hazardous if the following condition holds:

$$\frac{\overline{\mu_{\ell}} - \mu_k}{S_n} > \tau \tag{17}$$

In other words, an address x_k is hazardous, if $\mu_k < \overline{\mu_{\ell}} - \tau \cdot \sigma$ (e.g., $\tau = 2$). The weight for x_k is defined as follows:

$$w_k = \begin{cases} 1 & \mu_k < \overline{\mu_{\ell}} - \tau \cdot \sigma \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Accordingly, the scan probability of x_k is computed as follows:

$$\pi_2(x_k) = \frac{w_k}{\sum_{i=1}^{m} w_i} \tag{19}$$

In this work, we assume that at any time only one of these tests is verified. Therefore, the final characterization, denoted as π, is either π_1 (non-uniformity is verified) or π_2 (non-repetition is verified) or uniform distribution in case none of the hypotheses are verified.

Adaptive Mutation Rate. Another fundamental question that needs to be addressed is the duration of mutation interval. Although this duration can always be

set to the minimum practical value, as our analysis in [58] shows, unnecessarily small intervals will result in a generation of unneeded DNS queries and increases the overhead. Therefore, the duration must be decreased only when necessary. Setting the duration to longer intervals, on the other hand, can result in lower effectiveness. Hence, this duration must be adapted to the scanning rate of a potential attacker in the network. It is intuitive that the highest effectiveness is achieved when host addresses are mutated with the same rate with which the attacker is scanning the network. Assume λ^{min} denotes the minimum TTL value that is practically achievable and λ^{max} denotes the proactive mutation interval, which is the maximum interval that the addresses remain unchanged.

The j^{th} interval duration is denoted as λ_j. To calculate the length of the j^th interval, assume we have observed ℓ new failed scans during the last randomization interval of length λ_{j-1}. Therefore, during the last interval, a potential attacker has been scanning the network with an average rate of $\ell/(\lambda_{j-1})$. The length of the jth interval is determined based on the estimated average scanning rate of the $(j-1)^th$ interval and also previous intervals, using a low-pass filter to cut fast variations:

$$\lambda_j = \min(\max(\alpha \cdot \frac{\lambda_{j-1}}{\ell} + (1-\alpha) \cdot \lambda_{j-1}, \lambda^{min}), \lambda^{max}) \tag{20}$$

where $0 < \alpha < 1$. Our analysis shows that $\alpha = 0.75$ provides a smoothed estimator which avoids fast variations but it is still fast enough to react to sudden increases in the number of failed probes which could hint to new scanning activities in the network. However, note that λ_j could never be smaller than the practically minimum interval length.

2.2.3.2 Evaluation of Adaptive RHM

In this section, we evaluate the effectiveness of A-RHM as compared to proactive RHM. We again analyze the effectiveness of the model against two threat models: scanners and network reconnaissance. For scanners, we evaluate the effectiveness of A-RHM against cooperative and local-preference scanners and compare our results with those achieved for proactive RHM.

Cooperative Scanning. In A-RHM [58], adaptive mutation allows us to characterize the behavior of a potential attacker by analyzing the distribution of destination addresses of failed probes. This increases the effectiveness of address mutation against cooperative worms.

Figure 5 shows the simulated propagation of a cooperative worm in a static network, an RHM network, and an A-RHM network with the adaptive mutation. First, note that while for the first scenario, with RHM can only save 40% of network hosts from infection, with adaptive mutation less than 1% of network hosts are infected. For the second scenario, the higher scanning rate allows the worm to propagate initially before the non-repetition hypothesis is accepted. This allows the worm to infect almost 5% of network hosts. However, the worm propagation is contained afterward and 95% of network hosts evade infection completely.

Local-preference Scanning. Similar results are achieved for local-preference scanning as presented in Figures 6. The figure shows the propagation of a local-preference worm in various network settings with $m = 2^{20}$, and $n = 2^{16}$. Note that the adaptive mutation can severely slow down the propagation of the worm. In the first scenario

where $\eta = 1$ and $\lambda = 1s$, our approach can contain the propagation of the worm almost in its entirety. The second scenario, for a faster worm and a slower mutation rate ($\eta = 5$, $\lambda = 5s$), the propagation is still significantly slower than that of RHM. In general, the adaptive mutation is far more effective than the proactive one against local-preference scanning.

Network Reconnaissance. While A-RHM achieves significant improvement over RHM against scanners, its effectiveness against network reconnaissance is the same as that of RHM, because a human attacker would immediately discover the mutating nature of network addresses and avoid local or cooperative scanning strategies [60].

Moreover, A-RHM is also susceptible to advanced host identification attacks that would use other identifying attributes of a host such as its fingerprint, instead of IP address, to identify a host from previous reconnaissance [60].

2.3 Multi-dimensional Host Identity Hiding (M-RHM)

In this section, we introduce our cyber deterrence technique, called M-RHM. We first describe the intended threat model and then focus on introducing various strategies and dimensions of M-RHM.

2.3.1 Threat Model

In M-RHM, our goal is to defeat advanced and persistent multi-stage attacks on enterprise networks that aim to infiltrate deep into the network to compromise and take over critical internal servers. Due to defense-in-depth and zoning strategies, these attacks usually begin by one or a group of elite attackers on the Internet. The initial stage of the attack is reconnaissance of the public address space of the enterprise to

determine the configuration of active hosts, including their IP addresses, fingerprints (OS and services) and vulnerabilities.

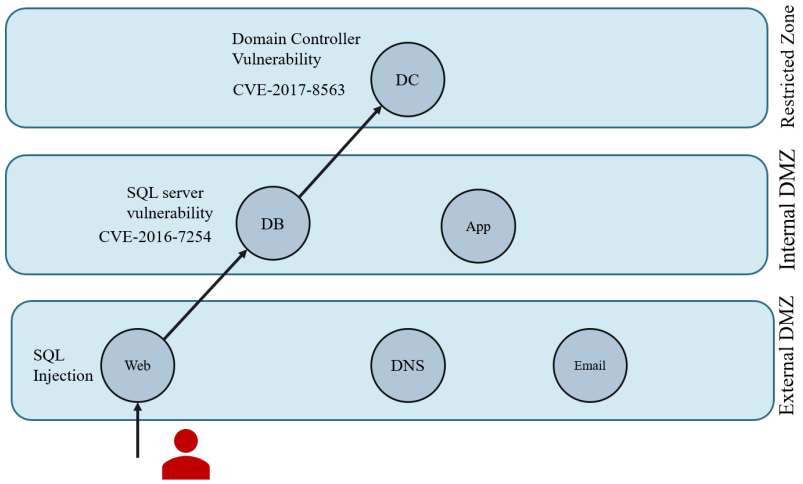

Figure 8: An example of multi-Stage APT intrusion

After identifying these network hosts and their configuration, the attacker tries to compromise a vulnerable service on an active host in the external DMZ. Once the attacker compromises this public host, she laterally moves to this host. Then the attacker would go on to take other actions, including escalating her privilege to the desired level.

However, this compromised public host is not usually the final objective of the attacker; but the attacker uses this public host as a *stepping stone* to reach internal hosts that are not accessible from outside due to boundary defense.

Therefore, the attacker again performs reconnaissance but this time on the internal address space to enumerate reachable internal hosts from the compromised host, along with their configurations. After this internal reconnaissance, the attacker again tries to compromises a vulnerable service on a discovered internal host.

These recurring steps are continued until the attacker reaches the desired host, or overtakes all possible hosts in the network. The goal of M-RHM is to defeat such multi-stage intrusions, by anonymizing identity of both public and internal network hosts through address mutation and fingerprint anonymization, and also by combining several mutation strategies over time and space. The goal is to maximally deter such attack progression and also maximize the potential for deception by maximizing the probability that the intrusion ends up in the decoys.

Figure 8 shows an example of such an attack in a small network with three zones. In the example network, the attacker first performs a reconnaissance of public address space and discovers three servers in the external DMZ. Then she identifies a public Web server that is vulnerable to SQL injection. By compromising the SQL injection vulnerability, the attacker compromises the Web server. Next, the attacker laterally moves to the Web server and uses internal reconnaissance to identify the two servers in the internal DMZ. Then, she discovers and exploits a vulnerability on the Database server and laterally moves to it. The attacker again does another internal reconnaissance from the DB server and identifies the domain an controller server. Finally, the attacker compromises a vulnerability on the domain controller and laterally moves to this server which is the goal of the attack.

In Chapter 1 we introduced the cyber intrusion kill chain which describes the steps taken by an advanced and persistent attacker during an intrusion. The Att&CK matrix introduced by MITRE [85] expands the last three stages of the kill chain (control, execute, maintain) with a ten tactic (e.g., persistence, privilege escalation, defense evasion, lateral movement, etc.). In this expanded cyber intrusion kill chain, two

steps are critical to defeating multi-stage attacks. The first one is the reconnaissance stage, which at network level could be described as the recurring process of identifying, enumerating and fingerprinting network hosts [46, 59]. The second one is *lateral movement*, which refers to the attacker's movement from one host to another in the network [85]. Figure 9 describes different cyber kill chain stages of the described multi-stage attack on the example network of Figure 8.

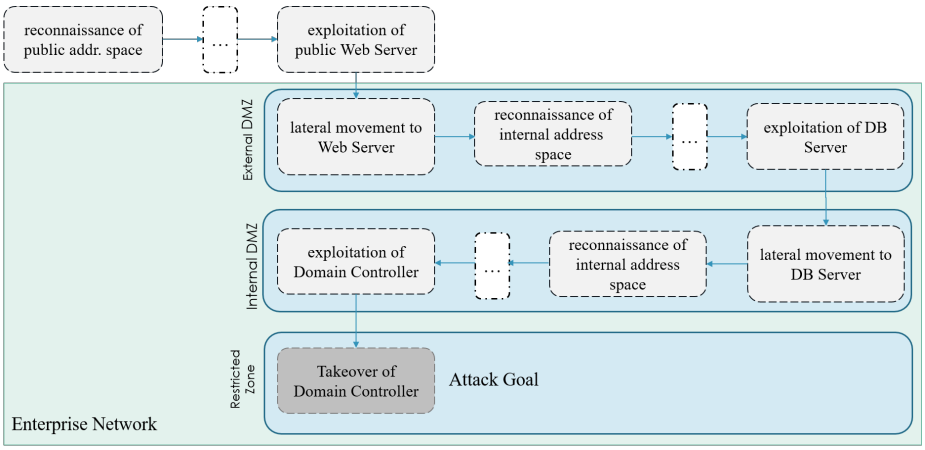

Figure 9: An example of cyber kill chain steps for multi-stage intrusion attacks

In addition to the location from where the reconnaissance is performed (either internal or external), the network reconnaissance could be classified into two groups, based on the reconnaissance technique: active or passive reconnaissance. Passive reconnaissance is an attempt to gain information about targeted hosts without actively engaging with them [46]. In the active reconnaissance, in contrast, the attacker engages with the target system, typically conducting a port scan to determine active hosts and their open ports [46].

Therefore, four types of reconnaissance are identifiable in a multi-stage intrusion attack:

- External and active: active scans to public address space by external attackers.
- External and passive: passive reconnaissance by external attackers.
- Internal and active: active scans to local or public address space by internal attackers.
- Internal and passive: passive reconnaissance by internal attackers.

Defeating multi-stage attacks requires an approach that provides countermeasures against all these four different classes of reconnaissance.

2.3.2 M-RHM Overview

Figures 10 and 11 show how various components of M-RHM are combined and what classes of reconnaissance each one is targeting. To introduce these strategies and their goal, we first provide an overview of the mutation dimensions and parameters that are included in M-RHM. M-RHM mutates two properties of network hosts. First, like RHM and A-RHM, M-RHM mutates host IP addresses. This mutation occurs by assigning ephemeral IP addresses to network hosts as described in Section 2.2.2. Second, M-RHM deploys *fingerprint anonymization* to disallow elite attackers from identifying a host identity based on its fingerprints (instead of its IP address). This is achieved by combining k-anonymity with address mutation for two fundamental reasons: (1) to anonymize host identities against elite attackers who use fingerprints for identification (de-anonymization) of hosts, and (2) to mutate addresses of a host only with its shadow decoys to make address mutation invisible to attackers. To achieve k-anonymity, for every network hosts, $k - 1$ shadow decoys (decoys with identical fingerprints) are placed in the address space.

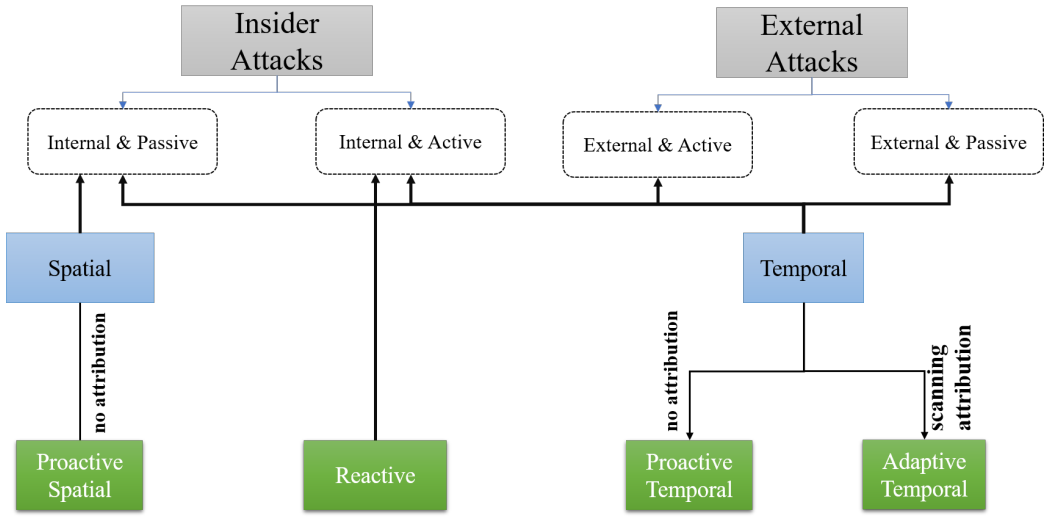

Figure 10: Various vectors of M-RHM and their corresponding threat models

M-RHM mutates addresses in two dimensions: temporal, i.e., at regular intervals, and spatial, i.e., based on source location.

Temporal Mutation: addresses are mutated over time, either at regular intervals (proactive) or tailored to observed scan rate (adaptive). The former is called proactive temporal mutation, while the latter is referred to as adaptive temporal mutation. This mutation is done both for external (public) and internal (local) address spaces. Temporal mutation aims to defeat both active and passive reconnaissance, and both for external and internal attackers. However, the adaptive temporal is only used against external reconnaissance.

Spatial Mutation: addresses are mutated based on source identity such that each host has a unique vision of the network. In other words, to reach host k, host i and j must use different IP addresses. Spatial mutation is only done for internal hosts, because for internal communications, the same host that queries the authoritative DNS must be the one using that queried address. While this is the case for internal

hosts, assuming this for external hosts is problematic due to existence of public DNS resolvers. The main goal of spatial mutation is to defeat information sharing among internal hosts, and more importantly, deter lateral movements in multi-stage attacks by not letting attackers use information collected on a host in any other internal host in the network.

In general, M-RHM combines three different strategies of mutation; namely *proactive*, and *adaptive* and *reactive* to provide resistance against a variety of threat models.

Proactive Strategy: when there is no information and characterization about existence or potential strategies of an attacker, M-RHM adopts a proactive mutation strategy, in both temporal and dimensions, as will be explained later. These two strategies are called proactive temporal and proactive spatial mutations strategies, respectively. The goal of proactive temporal mutation is to deter both external and internal reconnaissance when attackers' scanning patterns or rate is not identifiable. In comparison, the goal of proactive spatial mutation is to deter multi-stage internal attacks the by minimizing the potential of information reuse.

Adaptive Strategy: when some characterization of the strategies of potential attackers is identifiable, M-RHM adopts the adaptive mutation strategy by adapting address mutation based those thee characterizations. Note that this characterization occurs based on analyses of suspicious activities (e.g., probes to dark addresses and decoys) in the network. The adaptive strategy could be adopted on both temporal and spatial dimensions. But in this work, we only discuss the adaptive temporal strategy and leave the adaptive spatial strategy to future work.

The adaptive temporal mutation strategy follows two goals. First, it aims to en-

hance deterrence against external scanners by adapting mutation to attackers' scanning strategy and rate. Second, it aims to minimize unnecessary mutations when there is no or little potential of scanning activity on the public address space.

Reactive Strategy: if a flow to an active IP of an internal host i is identified as malicious, M-RHM adopts a reactive strategy against it by remapping that IP to a shadow decoy of i for only this source (identified attacker). This defense strategy of M-RHM is completely novel. We will discuss how we identify *malicious* flows for M-RHM in Section 2.3.7. The reactive strategy is enabled by the temporal mutation and as a result of short-lived nature of hosts' IP addresses. However, it defeats active internal scanning by identifying and redirecting these scans to shadow decoys.

In comparison to the adaptive strategy, if an activity is identifiably malicious then M-RHM adopts reactive mutation strategy. But adaptive strategy focuses on analyses of suspicious activities in order to learn about patterns or scanning rate of a potential attacker and then adapts mutations to this learned behavior accordingly.

The combination of these dimensions and strategies results in four different mutation strategies, which are shown in Figures 10 and 11. Figure 11 shows what information is needed for each strategy, what type of reconnaissance it is targeting, and what threat models would be affected by each of these four strategies.

2.3.3 Architecture

Figure 12 shows the architecture of M-RHM. Like RHM, M-RHM mutates attributes of network hosts after relatively short durations (e.g., 5-30 minutes). This duration is called *mutation interval*. This mutation includes externally-observable

Defense Strategy	Dimension	Knowledge about (Attribution)	Scan Source?	Active/ Passive?	Scan Type?	Threat Model
Reactive	-	Malicious scans identified	Internal	Active	Malicious (not preceded by DNS query)	Internal attackers/scanners
Proactive	Temporal	No knowledge	External/ Internal	Both	Suspicious (to dark address or ports)	External scanners
Proactive	Spatial	No knowledge	Internal	Both	Suspicious	Internal lateral movements (Multi-stage APT)
Adaptive	Temporal	Attacker's scanning strategy attributed	External	Active	Suspicious	External scanners

Figure 11: Various dimension and strategies of M-RHM and their threat models

parameters of a host, including IP and MAC addresses associated with its network interfaces and domain name (that is provided to reverse-DNS queries).

Figure 12 depicts the architecture for deploying M-RHM in a TCP/IP enterprise network. In this architecture, like RHM, a central entity called *Mutation Controller (MC)* is responsible for determining new mutations for network hosts, and announcing them to *Mutation Gateways (MG)*, which are distributed entities located at the boundaries of physical subnets (between subnet switch and default router). These gateways act similar to NAPT (network address and port translation) devices, translating address/port pairs based on new mutations received from the MC.

Honeycloud is a consolidated subnet of high-end machines, each hosting a number of virtual machines. These VMs are running the same platforms and services of the production hosts. This cloud is maintained by a cloud controller that uses an underlying centralized virtualization framework, such as Vagrant [32], for unified and automated management of these virtual machines. The cloud controller uses

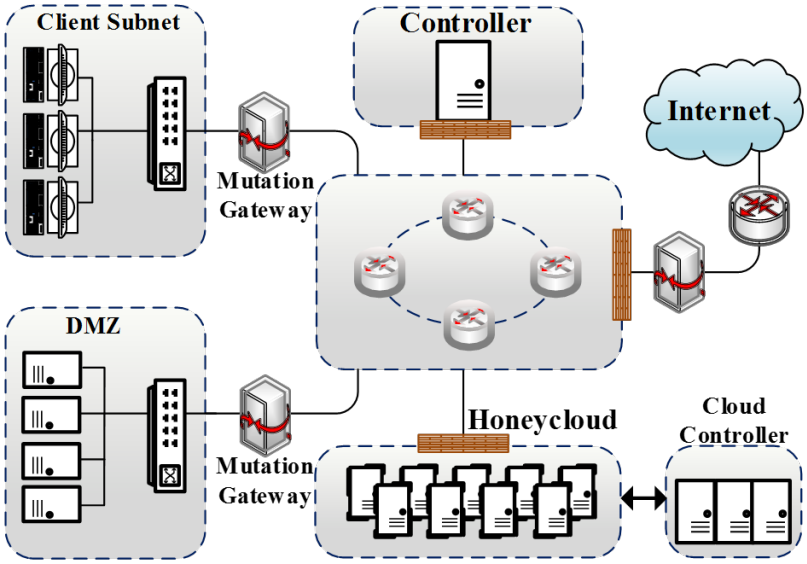

Figure 12: Architecture of the M-RHM

VM introspection techniques (e.g., VMScope [64], NFM [122]) to detect and handle infected VMs.

2.3.4 Mutation Parameters and Communication Protocols

In this section, we provide an overview of how mutations of different parameters are realized in M-RHM.

Mutation of IP Addresses. M-RHM mutates IP addresses of network hosts, both for public and internal hosts. Mutation of addresses is carried out as explained for RHM in Section 2.2.2. Regular communication with network hosts occurs via their domain names. When a user queries an authoritative DNS for IP address of a host, the DNS reply provides the temporary IP address of that host to the user. Figure 3 provides a step-by-step description of how communication with a host via name occurs in our network. The details are mentioned in Section 2.2.2 and are omitted here for brevity.

Fingerprint Anonymization. Mutation of IP addresses is not sufficient to defeat advanced reconnaissance. This is because the fingerprint of a host could also be used by an attacker to identify a host. For example, if only one host in the network is running a Web server, then attackers could use this to identify the host after its addresses are mutated [60]. In data privacy, such attributes are known as quasi-identifiers [123]; i.e., an attribute or a collection of attributes that could unintentionally identify an entity.

To address the problem of quasi-identifiers in data privacy, the concept of k-anonymity [123] is defined and enforced on the data. A release of data is said to have the k-anonymity property if the information for each entity contained in the release cannot be distinguished from at least $k-1$ entities whose information also appears in the release. For our problem, the k-anonymity means that the fingerprint of a host must be the same as at least $k-1$ other hosts at any point. This idea also coincides with the concept of *shadow* decoys [5] which has empirically shown to be effective for attack slowdown.

So, for each host $k-1$ shadow decoys with fingerprints identical to that host are located in the address space. Since honeycloud includes one instance of every real service in the network, traffic to all these $k-1$ shadows are received and handled by a one VM that has the same fingerprint as its host. The addresses for these shadow decoys are mutated the same as production hosts, to make them indistinguishable. These $k-1$ shadow decoys are created by redirection to services in the honeycloud.

Figure 13 shows how this redirection occurs. Assume an attacker, as part of her reconnaissance randomly selects an IP address IP_{rand} and probes it to discover if it is

running a Web application on port 80. If IP_{rand} is not assigned to a Web server, which is very probable even in a small address space, the NAPT module redirects them to the predetermined Web application in the honeycloud, which resides on port 8080 of a VM called Web honeypot. Note that no changes are required for any legacy protocol or device, and the mutations are transparent to end-hosts, users, and network devices.

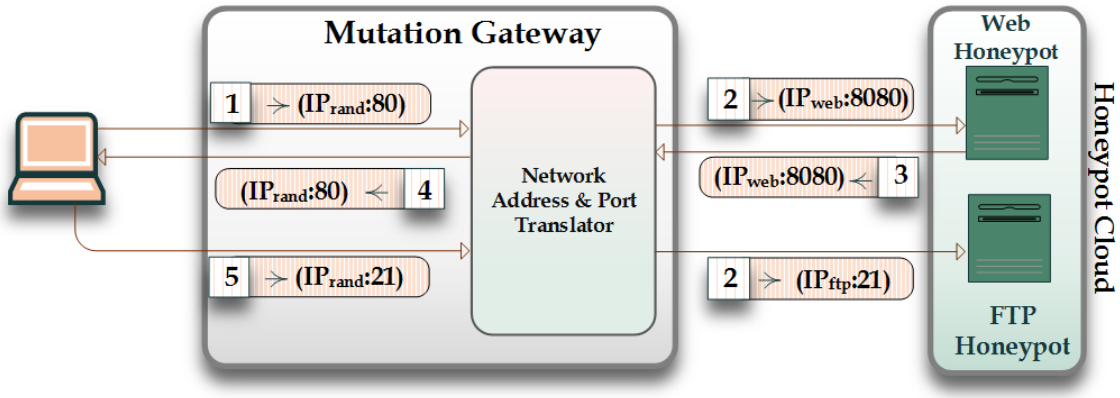

Figure 13: redirection of flows (destined to inactive address-port pairs) to honeycloud for generating shadow decoys

Mutation of Host Names against rDNS. Also, to address mutation, RHM mutates other attributes of network hosts to enhance its unpredictability and resistance against reconnaissance and scanning. Reverse DNS (rDNS) is a useful IP-to-name translation feature that an authoritative DNS may provide for domains in its zones. The reason rDNS mutation is important is that the attacker may maliciously send a rDNS query for a randomly-discovered currently legitimate eIP to discover its corresponding name and record it for future attacks. To thwart this, when a host sends a rDNS query for i, a temporally-mutated ephemeral name is returned. Ephemeral names are generated and distributed by the controller using a standard symmetric encryption algorithm such as AES. The key is generated based on time and source

host identity. This ensures that network hosts are still able to utilize rDNS responses for administrative logging, while the name is ephemeral and cannot be used for information gathering and hitlist attacks. Moreover, we only need to respond to rDNS queries which are issued by necessary services and legitimate clients.

Reverse DNS is represented by PTR records and stored in a special zone called *.in-addr.arpa* [47]. For example, the zone for the PTR record of address range 152.15/24 would be *152.15.in-addr.arpa*. This zone is administered by the owner of the address range.

Reverse-DNS mutation provides benefits of the rDNS while preventing its misuse by potential attackers. One of the main uses of reverse-DNS on the Internet is to verify that the sending server is not a malicious spammer. This is done by doing reverse-DNS lookups for the IP of the sending server to ensure that there is a rDNS record associated with it. If not, the receiving mail server considers the email as spam. Without rDNS mutation, since we are constantly mutating the IP address of our email server, we can not have a rDNS record for a specific IP address. Meanwhile disabling rDNS makes Internet mail servers to consider our mail server as a spammer.

In M-RHM, we include a PTR record for every unused range of the public address space. Now, once a receiving mail server issues a rDNS query, it will be delegated to our DNS server. If the queried IP is currently assigned to an active host, we generate an ephemeral name based on time and IP address of the querying source and send it to the querying mail server.

Another use of reverse-DNS is in logging for administrative purposes. For example, assume an internal Web server (for example Apache) in an enterprise network is

logging incoming Web traffic, and the Web server has a tool for doing a reverse-DNS lookup to discover the domain name of the source IP addresses in logs (*l*ogresolve in Apache [8]). If rDNS is disabled, the web server will not be able to identify the name of its clients, especially since IP addresses are constantly mutated. When rDNS mutation is implemented, the reverse-DNS lookup of an eIP provides the ephemeral name of that host. However, since each log is associated with a timestamp, an administrator can recover the original names by recreating the key from the timestamp and IP address of the source and decrypting the ephemeral name into its original real name. This implementation allows the administrators to use rDNS for internal purposes and also provides support for rDNS lookups for spam detection on the Internet. Meanwhile, it does not allow an attacker to do a reverse-DNS on an active eIP of a host and use the name later to discover the new IP from that name.

Mutation of MAC Addresses. M-RHM also mutates MAC addresses. Hosts in a physical subnet use ARP protocol for resolution of network layer addresses into link layer addresses. The objective of MAC mutation approach is to mutate MAC addresses of hosts in a physical subnet, to disrupt local scanning and reconnaissance threats. MAC addresses are mutated into ephemeral MAC addresses using naming standards of EUI-48/EUI-64 similar to the random mutation technique. However, to perform MAC mutation in legacy networks, a gateway must be located behind subnet switch. For SDN, such gateway is not required, and the MAC mutation is handled by the controller.

Randomizing MAC addresses to thwart reconnaissance in wireless networks has recently been adopted by major operating systems such as iOS 8 [91]. However, we be-

lieve that managing MAC address mutation at network level will make it transparent to end-hosts and thus increases its manageability and effectiveness, while significantly minimizing deployment cost. Next, we will investigate our four mutation strategies.

2.3.5 Proactive Temporal Mutation with Deception

Proactive temporal mutation mutates addresses of network hosts at regular intervals. This includes both IP addresses of public hosts for Internet sources and IP addresses of internal hosts for internal sources. While for public hosts, mutations are only temporal (no spatial mutation), for internal hosts these mutations are also spatially done based on the identity of the source: two different hosts i and j use different addresses to reach a destination host t. Here, we focus our discussion on mutation of private IP addresses for internal hosts. Mutation of public IP addresses is a special case of this where all Internet hosts are considered as one source.

Assume an enterprise network with m unused internal IP addresses, represented by A. To determine A, we encode used ranges as Boolean expressions using Binary Decision Diagram (BDD) [21] operations. The total mutation space, A, can be generated by subtracting used ranges (e.g., used for real IPs), denoted as $R_1, \ldots, R_u$, from the whole address space, R:

$$A \leftarrow R \wedge \neg(R_1 \vee \ldots \vee R_u) \tag{21}$$

Each host t has an anonymity number, denoted as k_t, where $k_t - 1$ represents the number of shadow decoys that are in the address to achieve k_t-anonymity for host t. Given k_t for a host t, k_t IP addresses are randomly chosen from A and assigned to

t. The set of these random addresses is denoted as A_t and represents the mutation space for host t.

The IP addresses to reach destination host t from every source host i are mutated after every *mutation interval*. The current mutation interval duration is represented by λ. The duration of this interval is initially set to a default number, λ^{max} which shows the maximum interval before IP addresses of a destination host i is mutated for all source hosts j. For external hosts, λ could be reduced by the adaptive temporal mutation strategy (Section 2.3.6) when more scanning activity is observed in the network.

Assume intervals are numbered from 1. At k^{th} interval, the IP address that a source host i must use to reach a destination t is denoted as $IP^k_{i\to t}$. At every mutation interval, this IP is randomly set to a different IP address from the set A_t. Assume the chosen IP for the host j at this interval is x. All other addresses in $A_t - \{x\}$ are assigned to shadow decoys of t for the source host j.

Mutation Rule 1 (proactive temporal mutation with deception): mutate IP addresses to reach t and its decoys for all source hosts j at every interval. Given n as number of network hosts and A_t as the mutation space of host t, we can formally define mutation rule 1 as follows:

$$\forall i, t, k : IP^k_{i\to t} \in A_t; i \neq t \tag{22}$$

$$\forall i, t, k, x : (IP^k_{i\to t} = x) \Rightarrow (IP^{k+1}_{i\to t} \neq x); i \neq t \tag{23}$$

This necessarily means that a host will only swap its address with its shadow decoys over time; i.e., at any point in time one and only one address from A_t is assigned

to reach destination t and the rest are assigned to its decoys. This also means that from the perspective of users, the address space configuration remains the same over time; in other words, the same fingerprints are observed on the same addresses, and this never changes over time. Therefore, the mutations are invisible to users; this is a fundamental difference between previous approaches and M-RHM.

Also, note that the addresses are mutated every λ^{max} seconds. We show that higher DNS TTL values, which are determined based on the length of mutation intervals, would result in less load on DNS servers. However, they would also slow down the rate with which previous scanning results are deprecated, and thus lead to less effective defense. A trade-off between the two can be achieved by keeping the mutation interval long, but if suspicious scanning activities are observed, this interval is adaptively reduced. This is the adaptive mutation strategy that will be discussed in the next section.

2.3.6 Adaptive Temporal Mutation with Deception

When attackers' scanning strategy (distribution) and the rate is characterizable, M-RHM mutates addresses according to these characterizations. The adaptive temporal is only applicable to public address space because internal scans are redirected to decoys by reactive mutation strategy.

In Section 2.2.3, we introduced two hypotheses for characterizing patterns in scans to the public address space. The first one tests whether scans are skewed toward certain ranges of the network. If this hypothesis is verified, higher scan probabilities are given to addresses in those ranges, based on Eq. 10. The second one tests whether

scans are non-repetitive. If so, higher scan probabilities are given to addresses that have received a lower number of scans. The result of these hypotheses is a scan probability distribution over address space, denoted by $\pi = \{\pi_1, \ldots, \pi_m\}$.

Given π, M-RHM mutates host addresses based on the following rule:

Mutation Rule 2-A (adaptive temporal mutation with deception): in mutating IP addresses to reach public host t and its decoys for all external sources at every interval, swap the address to reach t with a decoy that has a lower probability of being scanned.

Given n as number of public hosts, m as the number of unused public addresses addresses, A_t as the mutation space of host t, and π as characterized scanning probabilities of public addresses, we can formally define mutation rule 2 as follows:

$$\forall k, t, y, x : (IP^{k}_{ext \to t} = x) \wedge (y \in A_t - \{x\} \wedge \pi_y < \pi_x) \Rightarrow (IP^{k+1}_{ext \to t} = y) \qquad (24)$$

where ext denotes any external client. In other words, if a decoy of public host t has a lower probability of being scanned, we swap their addresses in the next interval. This means that hosts are moved to fewer addresses that are less probable to be scanned while decoys are moved to addresses that have a higher probability.

As noted in mutation rule 1, host addresses are temporally mutated every λ^{max} seconds. However, if some notable scanning activity is observed in public address space, mutation interval duration is adapted to this rate. This is because, as shown through evaluation in Section 2.2.2, when the scanning rate is high, faster mutation is required for better deterrence against it. However, unnecessarily small intervals will result in the generation of unneeded DNS queries and increase the overhead [59].

Therefore, the duration must be decreased only when necessary.

Mutation Rule 2-B (adaptive mutation rate): the next mutation interval duration for a public host t must be adapted to the current scanning rate observed on public address space. Eq. 20 defines our technique for adapting mutation rates to attackers' scanning rate. Note that the adapted interval duration can never exceed λ^{max}, but it can never become less than λ^{min} due to practical reasons.

2.3.7 Reactive Mutation against Internal Scans

The temporal mutation mutates addresses of network hosts periodically. The TTL value of DNS replies is set to short values such that it expires after each mutation. Therefore, to establish new connections to a host, user machines need to re-query the authoritative DNS for acquiring the new IP address. For internal hosts, the host itself will query the authoritative DNS, while for external hosts a local DNS resolver or even a public DNS resolver could be querying on behalf of the host.

For internal hosts, when host i queries for the address of host t, the mutation gateway can record the time of the query. Now, when flow from i to t is observed by its gateway, the gateway checks to see if there is a DNS query from i for the address of t since its last mutation. If not, then this flow is a scan because there exists no valid DNS query issued by i for t in the last mutation interval. Therefore, the flow is issued by a malicious entity that has been fortunate to probe the right address (that is currently assigned to t). Note that the longer the mutation interval λ^{max}, the higher the number of DNS queries that we must record for an interval.

Mutation Rule 3 (reactive mutation): if a malicious scan is issued from source

host i to destination host t (i.e., a flow not preceded by a DNS query), remap the current IP for t to a shadow decoy of t for only source host i.

Reactive mutation is especially effective against *active internal* reconnaissance because any malicious scan from a host is redirected to decoys. A fundamental result of this is those threat models that works based on active scanning, especially network worms, are not able to propagate inside an M-RHM network. In other words, using active scanning, a worm would only be able to infect public (DMZ) hosts, but any scan to local address space would be detected and redirected to decoys. Note that the temporal mutation plays a key role in making reactive mutation practical because by mutating addresses in short intervals, MG needs to keep track of a smaller number of DNS queries. Evaluation of the effect of shorter intervals on the overhead of reactive mutation is left to future works.

One advantage of reactive mutation is that it can also be used against botnets. This is especially the case where the IP addresses of bots are required to be known, such as cases where bots are used by botmaster as redirector proxies in fast-ux networks [100], or peer-to-peer botnets such as TDL-4 [44].

However, the reactive mutation may affect regular operations of legitimate peer-to-peer applications, such as Skype, in the network, because in these application IP addresses are attained from mechanisms other than DNS. Such applications are whitelisted, and reactive mutation is not applied to them.

2.3.8 Proactive Spatial Mutation

The static one-to-one binding of hosts to IP addresses allows adversaries to conduct thorough reconnaissance to discover network hosts. Specifically, this fixed address mapping allows distributed network scanners to aggregate information gathered at multiple locations over different times to construct an accurate and persistent view of the network. This enables adversaries to collaboratively share and reuse their collected reconnaissance information in various stages of attack planning and execution.

Spatial mutation [61] presents a novel moving target defense strategy which enables host-to-IP binding of each destination host to vary randomly across the network based on the source identity (spatial mutation). This spatial mutation will distort attackers' view of the network by causing the collected reconnaissance information to expire as adversaries transition from one host to another or share their information with other attackers. Consequently, adversaries are forced to re-scan the network frequently after each lateral movement. These recurring probings significantly raise the bar for the adversaries by slowing down the attack progress, while improving its detectability.

The spatial strategies address several limitations of the temporal strategies. Firstly, while temporal mutation forces adversaries to redo their reconnaissance over time, it does not provide any resistance against distributed reconnaissance [61]. Attackers may use distributed reconnaissance to reduce the amount and rate of noise generated by one specific host, to prevent detectors such as TRW [66] to detect them. It does not also provide any resistance against information sharing between hosts. This allows attackers to reuse reconnaissance information (such as hitlist [6]) collected at host

i on another host j. The spatial dimension would counter these threat models by providing a unique set of host-to-IP mappings to each network host.

Secondly, by providing a unique view of the network to each host, the spatial mutation would significantly complicate advanced persistent threat which relies on multi-stage intrusion. As described in our threat model, these adversaries are usually external entities who initiate their attack by compromising a publicly accessible host. Then, they gradually continue compromising hosts and laterally moving from one host to another until they finally discover a path to the target [39]. In legacy networks, as the attacker gathers more information and attacks new hosts, the search space shrinks and her knowledge about network enhances, which allows her to gradually move toward the target. However, with spatial mutation all gathered information is invalidated by moving from one host to another. This invalidation obfuscates an attacker's view of the network, making her to blindly attack already attacked hosts. Even if the attacker is informed of spatial mutation, this invalidation forces them to recollect information again and on every host, thus deterring their progress and increasing their efforts and detectability.

In legacy networks, a name is usually mapped to one static addresses which can be used globally (by any host) to reach the target. In spatial approach, name-to-address mapping occurs dynamically and as a function of source (requester) identity. Specifically, each client must use a different IP address to reach a destination t.

Assume $\mathbf{R}$, $\mathbf{E}$, and $\mathbf{N}$ denote the set of rIPs, eIPs, and names of the network respectively. The DNS functionality in static networks could be described as the

following:

$$DNS : \mathbf{N} \rightarrow \mathbf{R}, DNS(n_d) \mapsto i_d \tag{25}$$

Z where n_d and i_d denote the name and rIP address of the queried host respectively.

In spatial mutation, the mapping function is changed to reflect the spatial IP address assignment:

$$DNS' : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{E}, DNS'(n_d, n_s) \mapsto i_d \tag{26}$$

where n_s denote the name of the querying host (client). Note that the source identity is known to the authoritative DNS server for intra-enterprise DNS queries. For external clients, although there are proposals to extend DNS resolvers to pass in part of the client's IP address in the DNS message [13], this information is not currently available in DNS queries.

Mutation Rule 4 (proactive spatial mutation): for every pair of hosts i and j, the address for reaching host t from i must be used on j to reach one of t's shadow decoys.

$$\forall k, i, j, t : (IP^k_{i \rightarrow t} = x) \Rightarrow (IP^k_{j \rightarrow t} \neq x); j \neq i \neq t \tag{27}$$

This means that every pair of hosts must use two different IP addresses to reach the same destination host. Therefore, if attacker moves from i to j (or vice versa) or shares her information with an attacker on j, then using the IP of i to reach t would only lead the attacker to a decoy of t. Moreover, since all addresses for t and its decoys are chosen from A_t, and they all have the same fingerprints (based on k-anonymity),

spatial mutations are invisible to attackers as they laterally move from one host to another.

Assume a network has n internal hosts. From mutation rule 1 (proactive temporal with deception), we know that the set of mutating addresses for host t is denoted as A_t and must include k_t IP addresses to achieve k_t-anonymity. Therefore, to satisfy mutation rule 4, A_t must include $n-1$ addresses to achieve n-anonymity.

Assume m denotes the number of addresses that could be used for mutation. In order to achieve n-anonymity for all n hosts, we need $n(n-1)$ IP addresses. For example, for $n = 1000$ we need almost 1 million unused IP addresses, which may not be satisfiable in some networks. When $n(n-1) > m$, the number of available addresses is less than what is required for perfect spatial mutation, we can not achieve it. This limitation of addresses may prohibit achievement of perfect spatial mutation. Investigation of how addresses are distributed and assigned when we have limited number of addresses is left to future work.

Figure 14 shows an exemplary attack graph of a network with a critical host t as final target. Our strategy in proactive spatial mutation is to provide maximum deterrence and deception against a multi-stage attacker who may start from the initial hosts (e.g., Web) and compromises a chain of hosts to take over T. Assume $A_T = \{x_1, x_2, x_3\}$. The maximum deterrence and deception is achieved when Web, DB and DC all use different IPs from A_T to reach T.

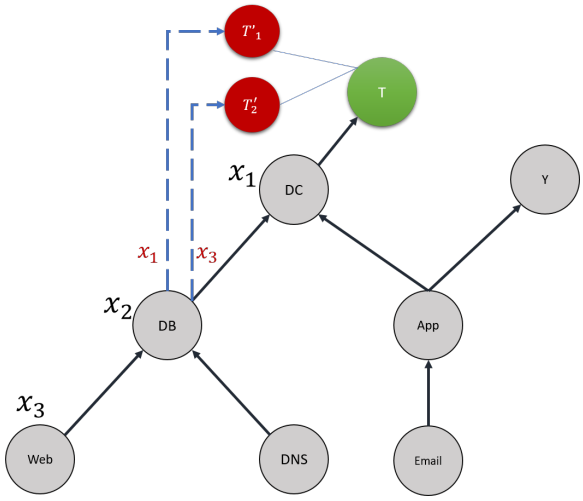

Figure 14: An example of spatial groups

2.3.9 Mutation Algorithms

In this section, we present an algorithm for M-RHM mutation. These algorithms combine various mutation strategies in a consistent manner. The main algorithm of M-RHM is defined in Alg. 1 and defines the inputs and strategies of the mutation. The inputs of this algorithm include

- Number of external unused addresses and hosts: m^e and n^e
- Number of internal unused addresses and hosts: m^i and n^i
- Minimum and maximum mutation interval values: λ^{min} and λ^{max}

Given these inputs, the mutation algorithm first initializes anonymity numbers for both internal and public hosts. Then, using this anonymity number, it determines the internal and public mutation spaces of internal and public hosts. After initialization, the mutation algorithm enters an infinite loop in which it repeatedly fetches the collected logs from the IDS and firewalls. Then, it updates data structures for flow statistics using the new logs. Then, three different functions are called. The first

function, called *mutate_proactive_adaptive_temporal* performs proactive and adaptive temporal mutation for external hosts.

The second function, called *mutate_proactive_spatial_temporal* performs spatial and temporal mutation for fixed mutation intervals for all internal hosts. Finally, the third function called *mutate_reactive* performs reactive mutation for malicious internal communications based on the update flow statistics. Alg. 2 to 7 define all auxiliary algorithms that are used by the main algorithm for consistent and synergistic mutation of both internal and public hosts.

To understand how Alg. 1 works in practice, what conflicts may arise among different strategies and how we address them, note that the mutation algorithm 1 mutates both public addresses and local addresses. For public address space, only two strategies are adopted: proactive and adaptive temporal. The proactive temporal strategy is used only when no characterization of scans is possible, or no scanning activity is observed in the address space. In contrast, the adaptive temporal strategy is only applied when such characterization exists. Since one of these strategies is applied at an interval, no conflicts arise between them. Alg. 3 called *mutate_proactive_adaptive_temporal* describes the algorithm for consistent mutation of public addresses based on proactive and adaptive temporal strategies. Alg. 4 called *characterize_temporal* performs characterization of an attacker's strategy and mutation rate on the public address space.

For local addresses, the proactive spatial, proactive temporal and reactive strategies are used in the following manner: At fixed mutation intervals (proactive temporal), we change the mappings based on constraints of the spatial mutation. Therefore,

temporal constraints define how addresses change from one interval to another, while spatial constraints define how addresses must be assigned within one specific interval. This makes the constraints for temporal and spatial strategies independent, and therefore no conflict arises between the temporal and spatial strategies.

The reactive strategy is done after spatial assignments and can override some of the assignments for individual flows. This is because if a flow from host i to j is not preceded by a DNS query from i for IP of j, we know that this flow is certainly malicious. Therefore, we ignore previous the assignment and redirect it to one of the decoys of j, but only for source i and only for this individual flow This real-time redirection is only limited to the current flow from i to j and has no effect on assignments and therefore does not add to the constraints for mutation planning over several intervals (temporal) or within one interval (spatial).

2.3.10 Evaluation

In this section, we evaluate the effectiveness of M-RHM against our two main threat models: external scanners, and multi-stage attacks. To quantify this effectiveness, we introduce two metrics.

Deterrence Ratio: by deprecating an attacker's information about the network, M-RHM forces the attacker to frequently redo his reconnaissance activities to regain the lost information, thus delaying the completion of the attack. In M-RHM, this loss occurs over time (temporal) and also over location (spatial). *Deterrence* in a network is defined as the amount of time needed to finish the attack successfully. To quantify deterrence magnitude against a specific threat model, we compare deterrence

Algorithm 1 mutate(m^i, n^i, m^e, n^e, λ^{min}, λ^{max})

Inputs: m^i as num. of unused internal addresses, n^i as num. of internal hosts, m^e as num. of unused public addresses, n^e as num. of public hosts, λ^{min} as minimum feasible interval duration, λ^{max} as maximum interval duration

{Initialize Parameters}
for all public hosts t **do**
  randomly select $n^e - 1$ addresses $\{x_1, \ldots, x_{n^e-1}\}$ from m^e
  $A_t \leftarrow \{x_1, \ldots, x_{n^e-1}\}$
end for
$\Gamma^e \leftarrow \{A_1, \ldots, A_{n^e}\}$
for all internal hosts t **do**
  randomly select $n^i - 1$ addresses $\{ip_1, \ldots, x_{n^i-1}\}$ from m^i
  $A_t \leftarrow \{x_1, \ldots, x_{n^i-1}\}$
end for
$\Gamma^i \leftarrow \{A_1, \ldots, A_{n^i}\}$
while true do
  $L =\leftarrow$ collect_logs
  $(S, F, D) \leftarrow$ update_flow_stats(L)
  mutate_proactive_adaptive_temporal(Γ^e, λ^{min}, λ^{max}, S)
  mutate_proactive_spatial_temporal(Γ^i, λ^{max})
  mutate_reactive(F, D, Γ)
end while

Algorithm 2 $(S, F, D) \leftarrow$ update_flow_stats(L)

Inputs: $L = \{(s_1, d_1, prot_1), \ldots\}$ as logged flows
for all internal host i **do**
  for all internal hosts t **do**
    $f_{i,t} = false$
    if there is a flow from i to t in L **then**
      $f_{i,t} = true$
    end if
    $q_{i,t} = false$
    if there is a DNS query from i for t **then**
      $q_{i,t} = true$
    end if
  end for
end for
for all public addresses x **do**
  $S_x \leftarrow$ num. of flows with destination address x in L
end for

Outputs: $S = \{s_1, \ldots, s_{m^e}\}$ as num. of scans to public addresses, $F = \{f_{i,j}\}$ as set of observed flows, $D = \{q_{i,j}\}$ as set of observed DNS queries

Algorithm 3 mutate_proactive_adaptive_temporal(Γ^e, λ^{min}, λ^{max}, S)

Inputs: $\Gamma^e = \{A_1, \ldots, A_{n^e}\}$ as mutation spaces of public hosts, λ^{min} as minimum feasible interval duration, λ^{max} as maximum interval duration, $S = \{s_1, \ldots, s_{m^e}\}$ as num. of scans to all public addresses

for all destination hosts t **do**
  if λ seconds passed since last mutation for t **then**
    $(\pi, \lambda) \leftarrow$ characterize_temporal(λ^{min}, λ^{max}, λ, S)
    $y \leftarrow IP_{ext \rightarrow t}$
    randomly select $x \in A_t - \{y\}$ from addresses with minimum probability in π
    $IP_{ext \rightarrow t} \leftarrow x$
  end if
end for

Algorithm 4 $(\pi, \lambda) \leftarrow$ characterize_temporal(λ^{min}, λ^{max}, λ, S)

Inputs: λ^{min} as minimum feasible interval duration, λ^{max} as maximum interval duration, λ is previous mutation interval duration, $S = \{s_1, \ldots, s_{m^e}\}$ as num. of scans to all public addresses

$\pi_1 \leftarrow$ non-uniformity_test(S)
$\pi_2 \leftarrow$ non-repetition_test(S)
$\pi \leftarrow \pi_1 \oplus \pi_2$
$c \leftarrow \sum_{i=1}^{S_{m^e}} S_i$
$\lambda \leftarrow \alpha \cdot \frac{c}{\lambda} + (1 - \alpha) \cdot \lambda$
$\lambda \leftarrow \min(\max(\lambda, \lambda^{min}), \lambda^{max})$

Outputs: π as characterized scan probability distribution, λ as new mutation interval duration

Algorithm 5 mutate_proactive_spatial_temporal(Γ^i, λ^{max})

Inputs: $\Gamma^i = \{A_1, \ldots, A_{n^i}\}$ as mutation spaces of internal hosts, λ^{max} as maximum mutation interval duration

for all destination host t **do**
  if λ^{max} sec passed since last mutation of t **then**
    $I_t \leftarrow$ assign_spatial(t, A_t)
  end if
end for

Algorithm 6 $I_t \leftarrow$ assign_spatial(t, A_t)

Inputs: t as target host, A_t as mutation space of t

for all source hosts $j \in n^i - \{t\}$ **do**
  $y \leftarrow IP_{j \to t}$
  randomly select $x \in A_t - \{y\}$
  $IP_{j \to t} \leftarrow x$
  $A_t \leftarrow A_t - \{x\}$
end for

Outputs: $I_t = \{IP_{1 \to t}, \ldots, IP_{n^i \to t}\}$ as new addresses to reach destination t

Algorithm 7 mutate_reactive(Γ^i, F, D)

Inputs: $\Gamma^i = \{A_1, \ldots, A_{n^i}\}$ as mutation spaces of internal hosts, $F = \{f_{i,j}\}$ as set of observed flows, $D = \{q_{i,j}\}$ as set of observed DNS queries

for all hosts i and j s.t. $f_{i,j}$ **do**
  if $\neg q_{i,j}$ **then**
    $x \leftarrow IP_{i,j}$
    randomly select $y \in A_j - \{x\}$
    $IP_{i,j} \leftarrow y$
  end if
end for

in an M-RHM versus that of a static network. Assume T_{MRHM} and T_{static} denote the deterrence of M-RHM and static networks respectively. The *deterrence ratio* is defined as follows and shows the magnitude of the delay exerted on the attack before its completion.

$$detRatio = \frac{T_{MRHM}}{T_{static}} \tag{28}$$

While in Section 2.2.2 we showed that RHM (and A-RHM) exert deterrence against scanners and worms, here we show that M-RHM effectiveness goes beyond such automated attacks and deter sophisticated multi-stage attacks that are launched by stealthy and elite human attackers.

Deception Ratio: by invalidating an attacker's assumptions about the network,

M-RHM increases the probability that attackers' malicious or suspicious connections hit decoys thus increasing the deception exerted on attackers and making the attack more detectable and characterizable. *Deception* in a network is defined as the number of probes to decoys before an attack is completed successfully. To quantify the increase in deception, we compare deception in M-RHM to that of a static network. Specifically, assume C_{MRHM} and C_{static} denote the deception in M-RHM and a static network. The *deception ratio* is defined as:

$$decRatio = \frac{C_{MRHM}}{C_{static}} \tag{29}$$

2.3.10.1 Effectiveness against External Scanners and Worms

Scanners and worms (or any other automated malware) use various scanning techniques to identify network hosts. These scanning techniques can be classified into two main categories: (1) uniform scanning, and (2) strategic scanning. In uniform scanning techniques, the scanner uniformly selects addresses from the intended address space and probes them. However, uniform scanning generates a high volume of noise which makes it highly susceptible to detection. Strategic scanning techniques, such as cooperative or local-preference scanning, take advantage of network dynamics to improve their effectiveness while reducing the volume of generated traffic. Therefore, contrary to uniform scanning, strategic scanning techniques have a higher chance of evading detection.

There is one major difference between RHM (also A-RHM) and M-RHM regarding worm propagation. For worm propagation to infect internal network hosts, it must

propagate inside our network using *internal and active* scans. However, the reactive mutation would identify active scans and remap them to decoys, because they are not preceded by appropriate DNS queries. Therefore, M-RHM does not allow worms to propagate inside the network. This means that a worm is only able to scan and potentially infect only public hosts. In the rest of this section, we investigate the effectiveness of M-RHM against scanners.

Cooperative Scanning. Sophisticated scanners usually aim to minimize connection failures by avoiding repeated probing of the same IP address [135], because in a static network the status of an address does not frequently change in a relatively long interval.

Figure 15 compares the ratio of infected hosts in a static network vs. our M-RHM network. Note that M-RHM is highly effective against such cooperative scanners. Moreover, after a few initial infections, the number of infected hosts does not increase after a certain time, because M-RHM constantly mutates host addresses to safe zones (addresses that are less likely to be scanned).

Figure 16 compares the deception against such cooperative scanners in static vs. M-RHM network. Note that the deception has a slight increase; this is due to the mutation invisibility property of M-RHM, which means the space in which a host mutates is limited to a small subset of the address space. This is contrary to RHM and A-RHM, where a host could take any available unused address. This limits the effect of the mutation on deception ratio.

Figure 17 shows the deterrence and deception ratios against cooperative worms, for various network sizes, m, and anonymity numbers, k. First note that the deterrence

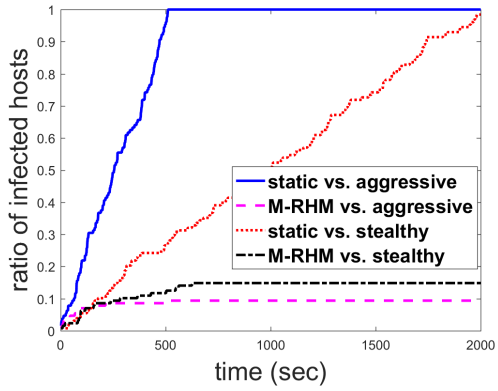

Figure 15: Deterrence against cooperative scanners

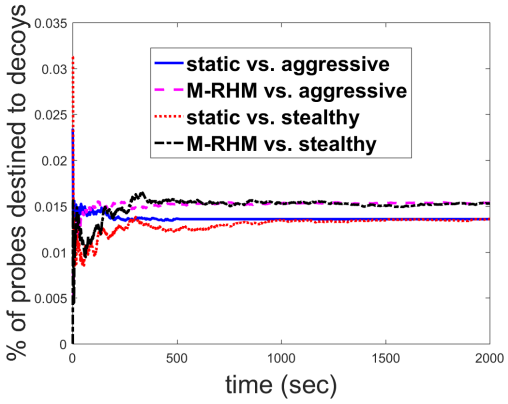

Figure 16: Deception against cooperative scanners

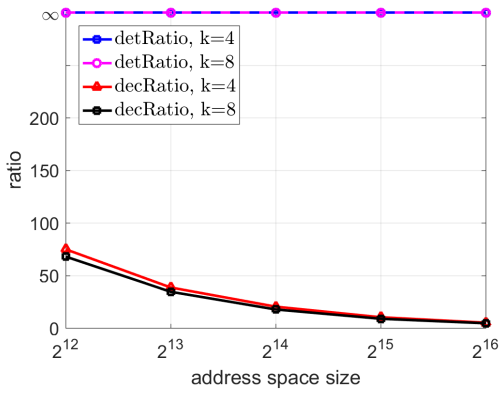

Figure 17: Deterrence and deception ratios against cooperative scanners for various network settings

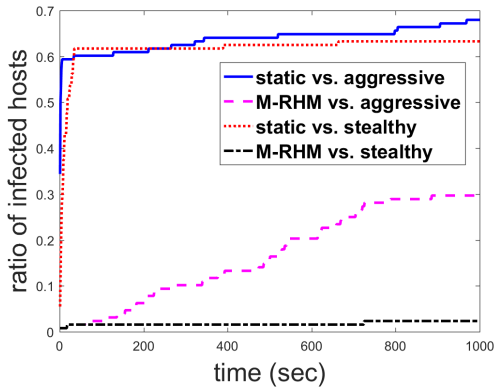

Figure 18: Deterrence against local-preference scanners

is ∞ because a cooperative scanner in M-RHM would never be able to discover all hosts. Secondly, note that the deception ratio is very high. However, the deception ratio decreases for larger network sizes. Also, k has low effect on deception ratio.

Local-Preference Scanning. Local-preference scanning aims to increase propagation speed by considering the distribution of hosts in the network. When vulnerable hosts are not uniformly distributed in a worm's scanning space, local-preference scan increases a worm's propagation speed [135].

We evaluate our approach against two types of local-preference scanners; a stealthy

scanner with a very low scanning rate of 8 scans per second, while the aggressive scanner scans 128 addresses per second.

Figure 18 compares the ratio of infected hosts (i.e., a host that is hit by the scanner) by a local-preference scanner. The network hast $n = 2^7$ hosts and $m = 2^{16}$, and $k = 8$; i.e., for each host there are 7 decoys in the address space. The local-preference scanner is assumed to scan some ranges with higher probability than others; in our simulation, we assume a scanner is scanning a random range of size 2^7 in public address space with probability 0.8; with probability 0.2 the target address for scanning is selected uniformly. Note that for both static networks, the scanner success rate over time is very high, and all network hosts would be discovered in less than 50 seconds. However, for both stealthy and aggressive scanners, the M-RHM network significantly deters the scanner.

Secondly, note that M-RHM adaptive temporal mutation is more effective against stealthy scanners as compared to aggressive ones because the delay caused by the stealthy scanning gives M-RHM sufficient time to discover the localized scanning pattern and adapt to it.

Moreover, using mutation rule 2-A (adaptive temporal), M-RHM constantly moves decoys into addresses that are more likely to be scanned. Therefore, the deception that is exerted on the attacker is increased. Figure 19 compares the percentage of probes that are destined to decoys, in static networks with the same number of decoys to that of M-RHM network. Note that while for static networks the decoy hit rate stays relatively low, this rate is almost 35% for our M-RHM network.

Figure 20 shows the deterrence ratio and deception ratio for various network sizes

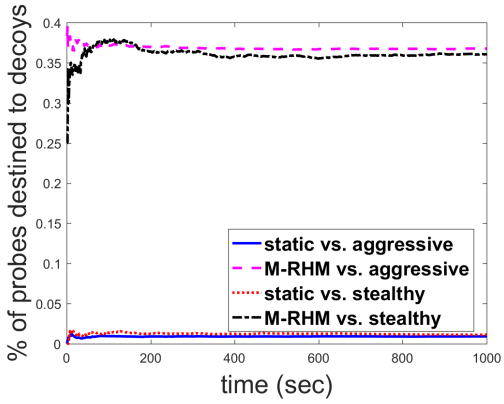

Figure 19: Deception against local-preference scanners

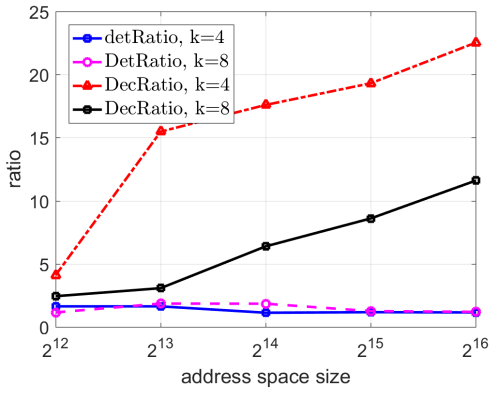

Figure 20: Deterrence and deception ratios against local-preference scanners

and anonymity numbers, k. Firstly, note that *detRatio* is almost independent of network size. In contrast, note that *decRatio* increases with network size; because the larger the network, the larger the deception space. However, note that with larger k, the *decRatio* is decreased; but this does not mean that with larger k, we have lower deception, but it means that with larger k the deception ratio is lower.

2.3.10.2 Effectiveness against Multi-Stage APT Intrusion

In legacy networks, as the attacker gathers more information and attacks new hosts, the search space shrinks and her knowledge about network enhances, which allows her to gradually move toward the target. However, in our M-RHM network, all gathered information is invalidated by moving from one host to another or if the attacker stays in a host for a long time.

Assume a network with n public and internal hosts, and 1 critical host. The network has z zones, where each zone has n/z hosts, and the critical host is in zone z which is the last zone. A host in zone k can only access hosts in zones k and $k+1$ and zone 1 can be accessed externally.

For simplicity, assume that in the attack graph of the network, an average attack path is of length r. In other words, starting from zone 1 the attacker must compromise a chain of r hosts to reach the critical host. We assume that all hosts have anonymity number k. With perfect spatial mutation, we could have $k = n - 1$, but we evaluate the approach for a general k.

We compare our approach with a static network with d decoys, to understand the benefit of each component and their integration against the above threat model.

Before continuing our evaluation, we introduce the effect of anonymized host identities on the reconnaissance. Assume from every host an attacker has to probe n hosts to exploit 1 of them as the target, which is the next host in the chain. When host identities are not anonymized, the attacker does not need to probe a host twice, and the sampling is uniform without replacement. However, when identities are anonymized, the attacker must select (sample) hosts with replacements. When each host is selected based on uniform sampling with replacement, the probability that the attacker hits the host in each try is $1/n$. The probability that the attacker hits in k tries follows geometric distribution $f(k) = (1 - 1/n)^{k-1}1/n$ where $1 \leq k$. The average number of tries before hitting the target is $E(f(k)) = n$.

However, if the attacker is sampling without replacement, the probability function that the attacker fails in $k - 1$ first tries is $(\frac{n-1}{n}\frac{n-2}{n-1}\ldots\frac{n-k-2}{n-k-1})$. The probability that he succeeds in the k^{th} try is $\frac{1}{n-k-1}$, because he has tried $k - 1$ hosts out of n hosts. Therefore the probability that he succeeds after k tries is $g(k) = 1/n$ where $1 \leq k \leq n$.

In this case, the average number of tries before hitting a target is:

$$E(g(k)) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \ldots + n \cdot \frac{1}{n} \Rightarrow$$
$$E(g(k)) = \frac{n+1}{2} \tag{30}$$

For simplicity of calculations, we estimate $E(g(k)) \sim n/2$ since n is usually large (order of thousands). At each host in the path, the attacker scans the address space to fingerprint network hosts and then attempts to exploit each host. In APT attacks, as described by cyber intrusion kill-chain [53], the reconnaissance (scanning) and exploitation stages occur separately. For the reconnaissance stage, attacker relies on tools such as Nmap [78] and Nessus [75] to identify what services are running and what vulnerabilities exist. In the exploitation stage, the attacker uses tools such as Metasploit [80] to potentially exploit a host. According to this threat model, an attacker first scans the address space once and collects fingerprints of all reachable hosts, and then attempts to exploit them in a uniform distribution, where this can be without replacement or with replacement [46, 53]. In the former case, the attacker needs to check $n/2$ hosts on average, while in the latter he needs to exploit n hosts on average.

The time for fingerprinting a host is denoted by t_f and the time for attempting to exploit (whether successful or not) is denoted as t_x. The deterrence for a network, which is the duration of attack completion, is calculated as:

$$T = f * t_f + x * t_x \tag{31}$$

where f and x show the number of fingerprinted and attempted hosts respectively.

Every zone includes $(n+d)/z$ hosts. In a static network with n hosts and d decoys, the attacker needs to fingerprint on average $(n+d)/z$ hosts every time he laterally moves to a new zone. However, if he moves laterally within the zone, fingerprinting is not needed. Therefore, in a chain of length r, the attacker only needs to fingerprint z times, and the number of fingerprinting for a static network is as denoted by f_{st}. For every zone, the attacker has to scan on average $(n+d)/z$ new hosts and with z zones, we have $f_{st} = (n+d)$.

The average number of attempted exploits at each host in the chain is $(n+d)/2z$, because in a static network attacker selects hosts based on uniform sampling without replacement and every zone includes $(n+d)/z$ hosts. The average number of attempted exploits is denoted as x_{st} and for z zones, we have $x_{st} = z \cdot \frac{n+d}{2z} = \frac{n+d}{2}$.

Every zone includes d/z decoys. All d hosts are fingerprinted by the attacker. Also, in each zone, the attacker must attempt $d/2z$ hosts on average before hitting the target. So with z zones, the average number of decoys that are attempted is $d/2$. The average number of times decoys in a static network is denoted by $h_{st} = d + d/2$.

In summary, the following formulas show the number of fingerprinted hosts, the average number of hosts that are attempted for exploitation, and the average number of engagement with decoys in a static network, respectively.

$$f_{st} = n + d \tag{32}$$

$$x_{st} = \frac{n+d}{2} \tag{33}$$

$$h_{st} = \frac{3d}{2} \tag{34}$$

Analysis of Spatial Mutation against APT. For a network with only spatial mutation and k-anonymity, each time the attacker moves laterally, he needs to fingerprint all the hosts in the zone again. At each host, the attacker needs to fingerprint $(kn)/z$ hosts, and the length of the chain is r. So, the attacker needs to fingerprint $(knr)/z$ hosts, which is denoted by f_{sp}.

Also, at each host, the attacker needs to attempt to exploit on average $(kn)/2z$ hosts. This is because there is no temporal mutation and once the attacker attempts to exploit a host, he does not need to attempt it again from the same source host. Every time an attacker moves laterally in the path, he needs to exploit $(kn)/2z$ hosts on average. The average number of hosts that are attempted is denoted by x_{sp} and with a path of length r, $x_{sp} = (knr)/2z$.

At each host, $1/k$ ratio of fingerprints and exploits hit real hosts, and $1-(1/k)$ hit decoys. The average number of engagement with decoys is denoted by h_{sp}.

$$f_{sp} = \frac{knr}{z} \tag{35}$$

$$x_{sp} = \frac{knr}{2z} \tag{36}$$

$$h_{sp} = (1 - \frac{1}{k}) \cdot (f_{sp} + x_{sp}) \tag{37}$$

To quantify the effectiveness of the spatial strategy, we measure its deterrence ratio against a static network that includes a fixed number of decoys. In our analysis we assume that $r = \sqrt{n}$; that is, the length of the chain is the square root of the number of hosts. We also assume that t_x is 5 seconds, and t_f is 1 second.

Figure 21 compares the deterrence ratio of a network with spatial mutation and

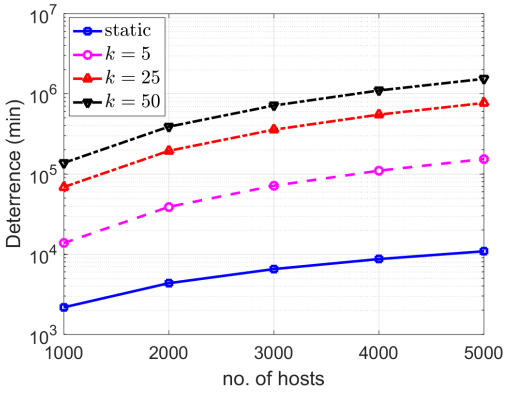

Figure 21: Deterrence ratio with only spatial strategy

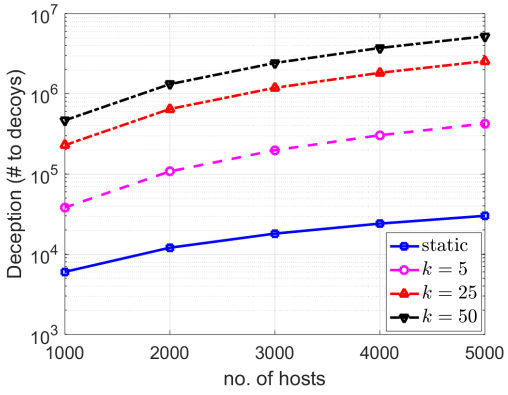

Figure 22: Deception ratio with only spatial strategy

different anonymity numbers against a static network with a relatively high number of decoys where $d = 5n$. The y-axis uses a logarithmic scale for all figures in this section. Note that spatial mutation can achieve deterrence ratio of 10, even with the same number of decoys ($k = 5$). Also, note that as the number of decoys increases the deterrence increases. However, this increase is slowed down with a larger k. Also, note that the deterrence increases linearly to the number of hosts.

Figure 22 shows the deception, which is the number of probes that hit the decoys, assuming that each fingerprint or exploit only needs one communication. Again note that the spatial mutation substantially increases the deception. The deception increases with larger values of k and n.

Analysis of Temporal Mutation against APT. For a network with only temporal mutation, assume λ denotes the mutation interval for all hosts.

We assume that the time for attempting exploitation is larger than the time for fingerprint; that is $t_x >> t_f$. Therefore, most of the attack time is spent on exploita-

tions. The number of hosts that are attempted within one interval is as follows:

$$y = \lfloor \frac{\lambda}{t_x} \rfloor \tag{38}$$

When the mutation is low, y is high. In the worst case, $y = kn$ represents the scenario where the mutation is so slow that all addresses remain unchanged during the attack time. In this case, the attacker needs to attempt $(kn)/2$ hosts in total. In the best case, when $y = 0$ it means that every time the attacker attempts to exploit a host a mutation occurs and the attacker loses his information. In this case, the attacker will select hosts based on uniform sampling with replacement, and he needs to attempt kn hosts in total. Using this notion, we estimate the number of attempts for temporal mutation as follows:

$$x_{tm} = kn - \frac{y}{2} \tag{39}$$

With x_{tm} attempts, the attack would take $x_{tm} \cdot t_x$ seconds to finish. During this time, the following number of mutations happen:

$$u = \frac{x_{tm} \cdot t_x}{\lambda} \tag{40}$$

On the other hand, the number of fingerprints is equal to number of mutations, u, multiplied by the number of hosts that must be fingerprinted after each mutation, which at each host is $(kn)/z$. Therefore:

$$f_{tm} = \frac{ukn}{z} \tag{41}$$

$$\tag{42}$$

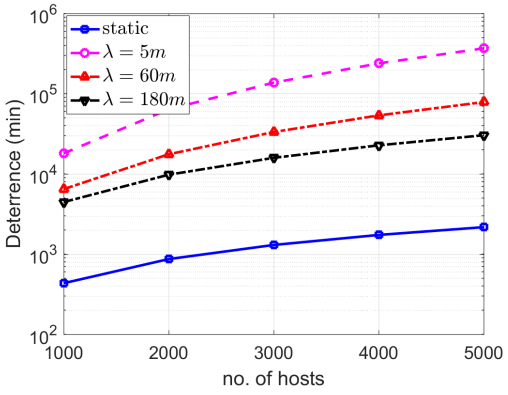

Figure 23: Deterrence ratio with only temporal strategy

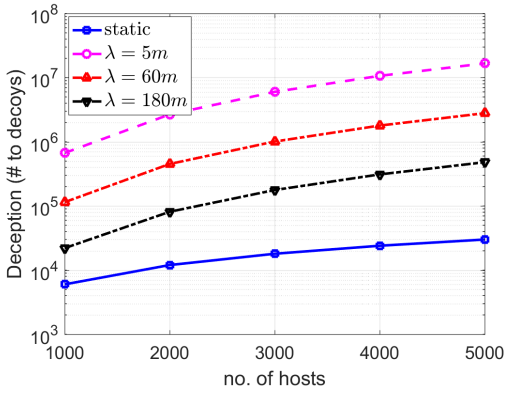

Figure 24: Deception ratio with only temporal strategy

Again, $1/k$ ratio of fingerprints and exploits hit real hosts, and $1-(1/k)$ hit decoys.

$$h_{tm} = (1 - \frac{1}{k}) \cdot (f_{tm} + x_{tm}) \tag{43}$$

Figure 23 compares the deterrence of a network with temporal strategy with the same static network described above. Note that, with short mutation intervals, the deterrence is significantly higher. As the mutation interval becomes larger, the deterrence becomes lower because slower mutation allows the attacker to avoid attempting the same host multiple times for a long time, thus increasing the probability that he hits the target. Also, note that the deterrence increases with the number of hosts because the attacker needs to compromise a longer path of hosts (larger r) and the disruption caused by the temporal mutation becomes higher.

Figure 24 compares the deception for these networks. Note that lower mutation intervals λ results in higher deception. Also, the deception increases with the number of hosts.

Analysis of M-RHM against APT. We assume an M-RHM network with tempo-

ral, spatial and reactive strategies. We assume perfect spatial mutation is achievable with the set of available addresses.

For number of attempts for M-RHM, we use the same methodology as that of the temporal. However, since every lateral movement results in a mutation, the attacker in the worst case needs to scan $r \cdot kn/z$ hosts:

$$x_m = \frac{knr}{z} - \frac{y}{2} \tag{44}$$

where y is defined in Eq. 38. For M-RHM, the number of fingerprints is the summation of those because of the temporal and those because of the spatial mutations. For u mutations and r lateral movements, where each mutation requires $(kn)/z$ fingerprints, we have:

$$f_m = \frac{ukn + rkn}{z} \tag{45}$$

Again, h_m is defined similar to h_{tm} in Eq. 43.

Figure 25 compares the deterrence in M-RHM network with a network with the spatial mutation only. The mutation interval λ is assumed to be 60 minutes. Note that even with $k = 5$, M-RHM outperforms the spatial. Also, note that with a high number of k, the deterrence significantly increases. Also, the difference in the deterrence increases as the number of hosts grows. Figure 26 compares the deception in these networks. Again note that even with $k = 5$, M-RHM achieves a higher deception than the spatial. These results show that M-RHM outperforms the spatial dimension even with smaller anonymity numbers.

Figure 27 compares the deterrence in M-RHM network with a network that has the

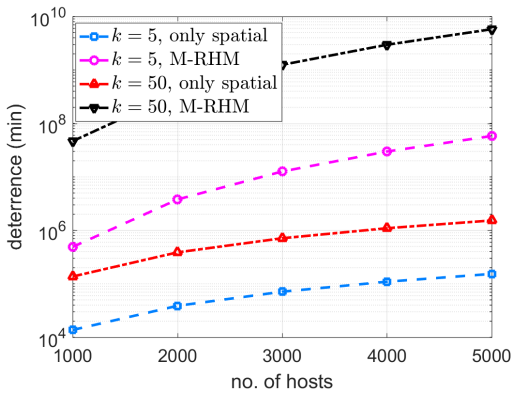

Figure 25: Comparison of deterrence in M-RHM vs. only spatial

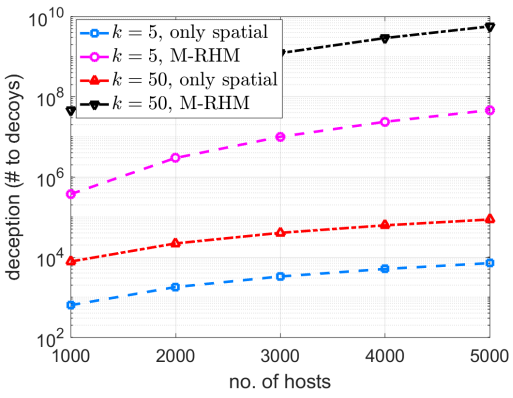

Figure 26: Comparison of deception in M-RHM vs. only spatial

temporal mutation. Note that M-RHM outperforms the temporal, even with smaller mutation intervals λ. Also, again note that lower mutation intervals ($\lambda = 5m$) result in significant deterrence against the attack. Figure 28 compares the deception in these networks. Again, note that the deception is higher for M-RHM even with higher mutation intervals.

These results show that the combination of both spatial and temporal dimensions achieves a deterrence and deception that is higher than that of the spatial or the temporal alone. But is the benefit of this combination more than the sum of individual components? Figure 29 and 30 compare the M-RHM deterrence and deception with the summation of the deterrence and deception for individual spatial and temporal mutations. Note that in the same scenarios, M-RHM achieves both higher deterrence and deception than the sum of individuals. This shows that M-RHM strategy integration is synergistic.

2.3.10.3 Evaluation Summary

M-RHM outperforms previous models for the following reasons:

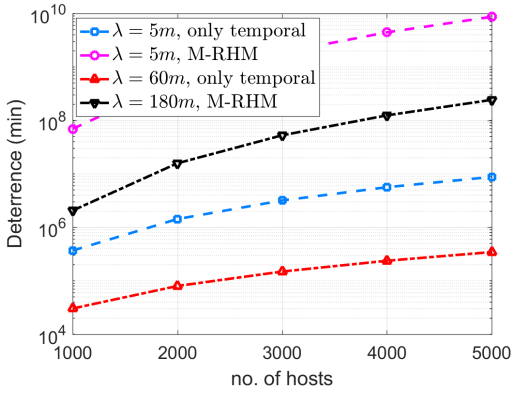

Figure 27: Comparison of deterrence in M-RHM vs. only temporal

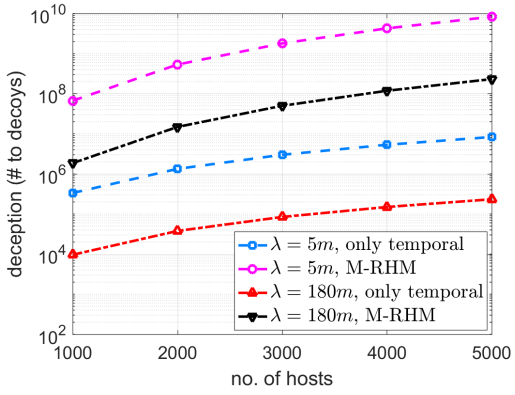

Figure 28: Comparison of deception in M-RHM vs. only temporal

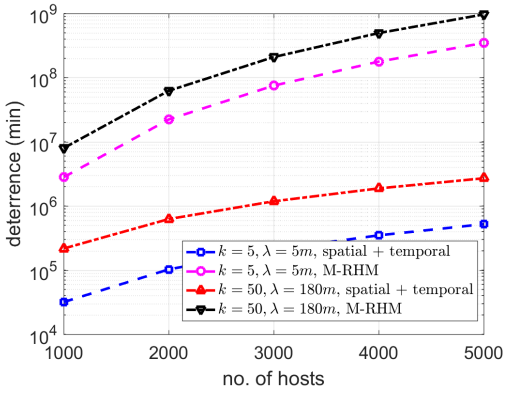

Figure 29: Comparison of deterrence in M-RHM vs. sum of strategies

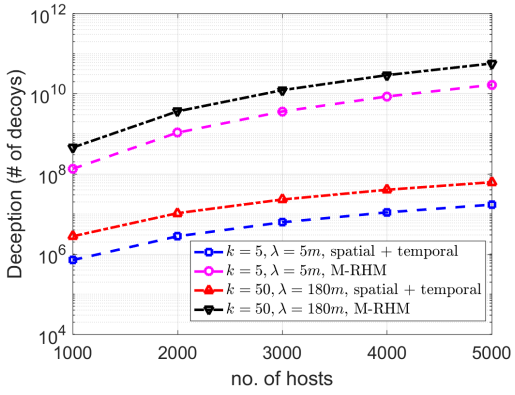

Figure 30: Comparison of deception in M-RHM vs. sum of strategies

- Defeating advanced reconnaissance: contrary to RHM and A-RHM, this model can defeat advanced reconnaissance. This is because, in addition to mutating host IP and MAC addresses, M-RHM anonymizes network fingerprints of hosts by hiding them in a pool of shadow decoys, as described in the next section. Therefore, contrary to previous approaches, even the most skilled attackers are not able to identify a host from previous reconnaissance. Also, contrary to RHM and A-RHM, this approach mutates host IP addresses internally; this means that every internal host has its unique vision of the network; As a result,

different internal (insider) attackers are not able to share their information with each other; more importantly, if attacker laterally moves from a host to another, all her previous reconnaissance is invalidated.

- Defeating multi-stage intrusion (APT) attacks: contrary to previous models that mostly focused on scanners, M-RHM can defeat multi-stage intrusion attacks by disrupting their reconnaissance stage; this is achieved by (1) introducing a spatial mutation into the model to disrupt information sharing in lateral movements; (2) incorporating reactive mutation to disable internal and active reconnaissance; this means that attacker is not able to use tools such as Nmap or Nessus to actively scan internal address space; (3) incorporating fingerprint anonymization, using the idea of k-anonymity and shadow decoys, to defeat advanced reconnaissance to enumerate network hosts; and (4) deceptive mutation both in temporal and spatial dimensions, by replacing a host location (IP address) only with its shadow decoys; this means that contrary to RHM and A-RHM address space layout looks static. This invisibility of mutation increases the probability that even a skilled attacker stays unaware of mutation, thus increasing both deterrence and deception on the attack.
- Defeating internal scanners: the reactive mutation identifies malicious probes to active hosts and remaps them on-the-fly to their respective shadow decoys. This means that (1) network worms can not propagate internally unless they rely on passive reconnaissance (e.g., discovering IPs using open connections); and (2) internal attackers cannot use active scanning tools such as Nmap or Nessus to discover network hosts. This is a significant achievement over RHM

and A-RHM.

- Achieving maximum deception through invisible mutation: contrary to RHM and A-RHM, in M-RHM the address mutation is not visible to attackers. This is because a host only mutates its address with one of its decoys; therefore, an attacker would not immediately become aware of mutation. This lack of awareness would prohibit an advanced attacker from adapting to our mutation; however, an attacker in an RHM network would immediately discover that addresses are being mutated and would adapt to it.

CHAPTER 3: A FORMAL FRAMEWORK FOR CYBER DECEPTION PLANNING

3.1 Motivation

The important role of deception as a warfare tactic has been known in military sciences for thousands of years [124]. Over time, as this warfare has evolved, so has the sophistication of these deception tactics, evolving from simple deception techniques like Trojan Horse by Greeks [124] to complex multi-faceted deception plans like Operation Fortitude by Allies in World War II [124]. With the emergence of cyber warfare, *defensive deception* has re-emerged as a proactive defense paradigm in protecting information systems [96] by diverting attackers from reaching their targets, while potentially learning about their motive and techniques.

Pioneered by preliminary works like Stoll [120], the popularity of defensive deception picked up in the late 90s by the advent of honeypot systems [96] and formation of Honeynet project [116]. Over time, as the nature and sophistication level of cyber threats evolved, more sophisticated deception systems were proposed; from *dynamic honeypots* [72] and *honeyfarms* [126] to *honey router* [41] and *honeyclient* [3]. However, the popularity and evolution of honey techniques have rather diminished since the late 2000s, mostly because the focus has remained on devising isolated systems and techniques. The isolated and limited benefit of these systems has not been able to justify their high deployment and analysis cost [7], especially in contrast with reactive

defensive technologies like intrusion detection paradigms which have constantly risen in prominence due to their high benefit-cost ratio.

While prevention (e.g., firewall) and detection (e.g., IDS) technologies remain a core component of cyber defense, they are no longer adequate in addressing the ever-increasing threats of evolving cyber attacks [62, 125]. In recent years, we have witnessed novel classes of advanced and persistent attacks using stealthy or zero-day threats that can not be fully mitigated by defense technologies [62, 63] such as IDS and firewalls. Examples are very stealthy indirect link flooding attacks [68, 121] or on-the-rise advanced Persistent Threats (APT) [53, 79]. In lack of effective reactive countermeasures, defensive deception, as a proactive defense paradigm, could again play a significant role in resisting against such stealthy and undetectable threats [125].

Our motivation for this chapter is to advance cyber deception as a powerful defense strategy against advanced attacks [53] by disrupting their reconnaissance through strategic and mission-oriented planning. This entails a new outlook and direction in defensive cyber deception.

3.2 Challenges

In this revisiting of cyber deception in defeating reconnaissance, we must note that *random* use of a few individual deception or honey things (e.g., honey hosts, honey services, honey applications, honey files, etc.) where each have their own *isolated* goal will have limited effectiveness, especially against advanced attackers. Instead, effective deception of such attackers requires a *strategic combination* of a group of coordinated lies in a manner that manipulates the attackers' thinking and

leads them to a predetermined false conclusion. This false conclusion would then persuade attackers to adopt a false course of action in their planning, thus leading to a high benefit for defense.

We define active cyber deception as an act of intentional and consistent misrepresentation of a group of facts to provide a distorted depiction of reality to attackers. In other words, the objective of cyber deception is depicting a wrong perception of the reality in receiver's mind, by a synergistic and consistent combination of several small lies to persuade attacker about a big lie. This big lie aims to mislead attackers to the desired state of knowledge (e.g., making attacker believe a real host is a decoy host), thus deflecting the attack from targeting that host, even when attacker's techniques are unknown or undetectable.

Solving deception problems as a combination of a group of lies (deceptive actions) requires a planning paradigm that provides a realistic but formal modeling of deception and its effect on attackers' thinking. This modeling must also consider the benefit and cost associated with different combinations of deceptive actions. This framework must provide necessary paradigms for defining various deception actions, along with their interdependencies, benefits, and costs. More importantly, such framework must provide paradigms for modeling how such deceptive actions would as a whole manipulate cognitive thinking process of attackers who may have different goals and sophistication levels. It must also be able to reason and identify the most beneficial deception plan (set of actions) with the given budget. We show that this planning problem is a generalization of 0-1 knapsack problem and thus it is NP-hard.

3.3 Approach Overview

In this chapter, we present a formal framework for the aforementioned problem of modeling and identifying optimal deception plans against advanced cyber threats. The framework introduces a *deception logic* for defining deception models, where each deception model addresses a specific class of threats (e.g., DDoS, network reconnaissance) in the cyber domain. The deception logic is an abstraction over satisfiability modulo theories (SMT) [12,87] that is extended with Gödel logic [10,43]. Gödel logic is a many-valued logic that provides a logical framework for modeling uncertainty.

Our framework provides necessary user interfaces for knowledge engineers to define their deception models. Alternatively, the deception model could be externally fed into the framework, usually as an output of an automated model generation program. The given deception model is synthesized into an SMT instance, which is then solved by the underlying SMT solver, for which we use Microsoft Z3 Theorem Prover [12]. The Z3 SMT solver finds a satisfiable assignment to the given instance. These assignments define what set of deceptive actions are optimal and are referred to as the *deception plan*. Each deceptive action is actuated in the system by a set of deception techniques that are implemented in the system.

To formalize deception modeling and planning for cyber defense, first, we need to provide formal definition of cyber deception. Deception is defined differently across various disciplines such as philosophy, psychology, military sciences, and cyber security. In philosophy and psychology, deception is defined as *intentionally* causing another person to *believe* in a statement that is considered to be incorrect by the

deceiver [18]. Here, the emphasis is on the intention and the *perlocutionary* nature of deception. A perlocutionary act changes feelings, thoughts or actions of the receiver as intended by the sender. Other examples of such acts are persuading, scaring, and inspiring. This means whether an act of deceiving has occurred on whether a particular effect (intention of deception) has been produced in the target of deception.

In military sciences, this particular effect (intention) must lead to *changing the target's course of actions* to what the deceiver desires. Definitions in cyber security [103, 107] is similar to military sciences in the sense that the focus is on the changing the behavior of the target.

By considering these definitions, we propose the following definition which captures all the important aspects of defensive deception against cyber attackers: *deception is a set of intentional actions aimed at causing a belief in attackers' minds to influence their course of actions in a way that is intended by the defender.*

Given this definition, a formal model of deception must model at least the following *five* core components [36, 107]: reality, belief, cause-effect, actions, and intention:

- *Reality and Belief*: it must be able to model the reality of the domain, as well as attacker's belief on this reality. The reality represents the real configuration of cyber systems, while the belief represents attackers' level of deception on this reality.
- *Cause-effect*: the logic must be able to model cognitive thinking process of attackers, by modeling how each deceptive action would manipulate attacker's belief, and also how an attacker's perception of a fact affects her belief about another fact.

- *Deceptive actions*: the logic must be able to model deceptive actions and their manipulative effect on an attacker's perception of reality. The model must also be able to define inconsistency among actions because conflicting lies would reveal the deception plan.
- *Intention*: the logic must be able to model deception goals as a set of beliefs to which the attacker must be driven.

In addition to these components, we identify the following components as essential elements for modeling deception in cyber defense:

- *Risk modeling*: the logic must be able to model how risk is manipulated by a deception plan. Moreover, it must be able to consider this risk in deception planning. In our model, intention or goal is defined based on risk.
- *Reasoning about uncertainty*: reasoning on deception involves uncertainty. A deception plan aims to deceive an attacker with a high certainty, and the model must be able to reason based on this uncertainty. Better deception plans achieve the deception goals with a higher certainty.
- *Deception Cost*: since the deployment of deceptive actions is costly, the model must be able to consider the cost of individual actions in constructing the deception plan.
- *Attack models*: the framework must be able to model a variety of attack models. Contrary to deception in the human society, the attacker modeling is needed for the formal definition of cyber deception, because there is a difference between deception in general and cyber deception. While in generic deception we usually have complete information about the target of deception, in cyber

deception we can rarely make such assumption. For example, in deceiving a multi-stage network intrusion attack, deceiving a naive attacker is expectedly different (and easier) from deceiving a highly-elite attacker. But identifying whether an attacker is naive or elite is not usually possible, at least not before persistent engagement and analysis of the attacker's behavior. In lack of such complete information about attackers, the deception modeling must support the definition of attack models and their effect in crafting the deception [17].

Finally, to adapt cyber deception plan to new observations about the attacker or changes in the system, the deception framework must also provide formal modeling for the following component:

- *Adaptability*: as we collect more information about an attacker, our assumptions about what the attacker knows or is after changes. The framework must be able to update the deception plan adaptively, by taking recent observations and characterizations regarding attacker's knowledge, beliefs, and motifs into consideration.

3.4 Related Work

Although we can trace back the use of defensive deception in computer security to more than three decades ago in works such as Stoll [120], the interest from security professionals and researchers picked up in late 90's by the advent of honeypot systems and formation of Honeynet Project in 1999. In early 2000s, several seminal works such as [11, 54, 88, 96, 96, 114, 117, 132] proposed different type of honeypots like *honeyd*, *honeytokens*, *dynamic honeypots*, and *honeyfarms* to enhance the security of

information systems by giving insight to defenders and by diverting attackers from production systems. Since then, defensive deception has been considered as one of the pillars of proactive defense. In this section, our objective is to review the literature on deception briefly. We investigate existing literature on cyber deception in two categories: works focused on deception systems, which we broadly refer to as *honey things*; and works that have focused on devising a formal theory, logic, or framework for modeling and planning cyber deception.

3.4.1 Deception Systems

Deception systems or honey things have been the main focus of research on cyber deception. The main example of these systems are honeypots; decoy resources that are placed in a computing system or network to be probed, attacked and compromised by attackers [96]. Honeypots are typically categorized as high-interaction and low-interaction. Low-interaction honeypots emulate services where the level of emulation built into the services determines the degree of intruder interaction with the honeypot [116]. Low-interaction honeypots could emulate the network stack such as *honeyd* [95], the service vulnerabilities like *nepenthes* [11], or networking protocols like *dionaea* [114]. High-interaction honeypots provide a real operating system designed to respond interactively to intruders [116]. Spitzner [116] proposed honeynet, a network of honeypot machines, to present a more plausible network environment to intruders. Kuwatly et al. [72] constructed a dynamic honeypot, based on *honeyd*, that is capable of adapting its configuration by monitoring changes in the configuration of production systems. Vrable et al.proposed honeyfarm [126], which is an architecture

to simplify large honeypot deployments by locating all honeypots in a single place and redirecting the traffic to sink IP addresses to the honeypots in this honeyfarm; therefore, enhancing the utilization of resources. Rowe [105, 106] introduced the concept of fake honeypots, which are production systems with artifacts of honeypots, such as using virtualization and system monitoring tools, created to fool attackers into thinking they have compromised a honeypot, hence reducing the number of attacks on the production systems. Another type of honeypots are tarpits such as Labrea, which consume attacker's resources by keeping their TCP connections open [76].

Another class of honey things are client honeypots; fake client applications that actively crawl on thes Internet and interact with potentially malicious services to find and blacklist malicious ones. Examples are HoneyClient [110], Strider HoneyMonkey [127], Monkey-Spider [54], Capture-HPC [109], and PhoneyC [89].

Several honey-thing technologies have been proposed for protecting network infrastructure; for example, HoneySpot [113] to detect attacks on wireless access points, and Honey Router [41] to detect anomalous behavior of malicious routers.

Finally, several research works have been proposed to protect data breaches, where the general idea is to insert tainted data in a dataset and then observe the access requests to that dataset; major examples are HoneyToken [118], Honeyfile [132], HoneyGen [128], and honeyword [65].

3.4.2 Deception Modeling and Planning Frameworks

We categorize works on deception modeling and planning into three groups of logic-based, game-theoretic, and probabilistic models. In our review of these works,

note that none of the existing works provide a framework that can address generic deception problems in cyber defense. To the best of our knowledge, our work is the first that offers a framework that enables modeling and solving generic cyber deception problems to craft effective and economical deception plans.

3.4.2.1 Logic-based Models

Several logic-based models of deception exist in the formal methods literature. Sakama et al. [107] introduce a modal logic for the deception that can express the concepts of belief, action, and intention and formulate eight different categories of deception. Authors in [108] introduce a propositional multi-modal logic that can represent three modalities: belief, intention, and communication. Using the logic, they formulate various types of dishonest communications between agents. These logical models cannot be directly used to address cyber deception planning problem; however, they can be used to formally express both an act of deceiving and its effect on addressee's beliefs.

In [104], Rowe presents an automated deception planner that take a sequence of operating system commands as input and find viable deception plans that are consistent with constraints that are defined in second-order logic.

Rosis et al. [31] propose a formal deception planning model that formalizes information impact on the receiver's mind. The deception plan consists of the decision of whether to deceive, the selection of a deception object, the form of deception, and a deception instrument. The deception strategy considers various receiver's criteria for believing, including content plausibility, source informativity, and information safety.

3.4.2.2 Game-theoretic Models

Another approach in addressing deception planning problems is modeling deceptive engagement between an attacker and a defender using game theory. Among all types of games, the signaling game [17,36,40,133] has been the primary modeling paradigm since the concept of signaling is inherently a natural fit for deception planning.

Zhuang et al. [133] define a signaling game to model a multiple-period attacker-defender resource-allocation problem in which the attacker attempts to obtain some secret information while the defender tries to keep her secret safe by allocating resources to develop enough security mechanisms in her environment with low cost. Ettinger et al. [36] present a game-theoretic modeling of bargaining tactic problem using belief manipulation and determine its equilibrium. Carroll et al. [17] investigate the effects of deception on the interaction between an attacker and a defender in a computer network. In their game, a defender has a network with a mixture of production servers and honeypot systems and uses camouflaging to either disguise a production system as a honeypot or to disguise a honeypot as a production server. The game-theoretic models, although insightful, only target specific deception problems and are far from being expressive or scalable in defining deception models in a more generic context.

3.4.2.3 Probabilistic Models

Rowe [101] presents a probabilistic model based on a decision tree to determine whether a single deception plan should be performed. The proposed model is for a scenario in which the defender wants to decide whether to deceive attackers about

the availability of network resources or not. To determine when to deceive, the cost of the *do not deceive* branch must be more than the *deceive* one. Rowe [101] extended the idea and show how decision trees can be used for multi-stage deception planning. They built a probabilistic model for an environment where attackers first attempt to guess system administrators' credentials, and if they were not successful, they attempt to obtain the regular user's credentials.

Rowe [102] introduces an obstructive counter-planning model that uses a probabilistic approach to determine the optimal set of deceptive mechanisms (called ploys) to deter the attack. The model builds a Markov model of the attack sequence with state/transition probabilities, determines the set of ploys for each state, and finally using the given benefit/cost parameters for each ploy, it determines the optimal set of deceptive mechanisms as deception plan.

Similar to game-theoretic models, these probabilistic models only provide modeling for specific deception problems and attack scenarios.

3.5 A Formal Framework for Active Cyber Deception Framework

Figure 31 shows the overall architecture, including inputs, components, and processes of the framework. The deception model is given to the framework as input. This formal model is defined using our *deception logic*, that is introduced next. This deception logic is an abstraction over Microsoft Z3 SMT logic, that is extended with G{o:}del fuzzy logic [10, 43].

The deception model is given to a module called *synthesizer* to refine and update it and translate it into an SMT instance [12]. An SMT instance is a generalization of

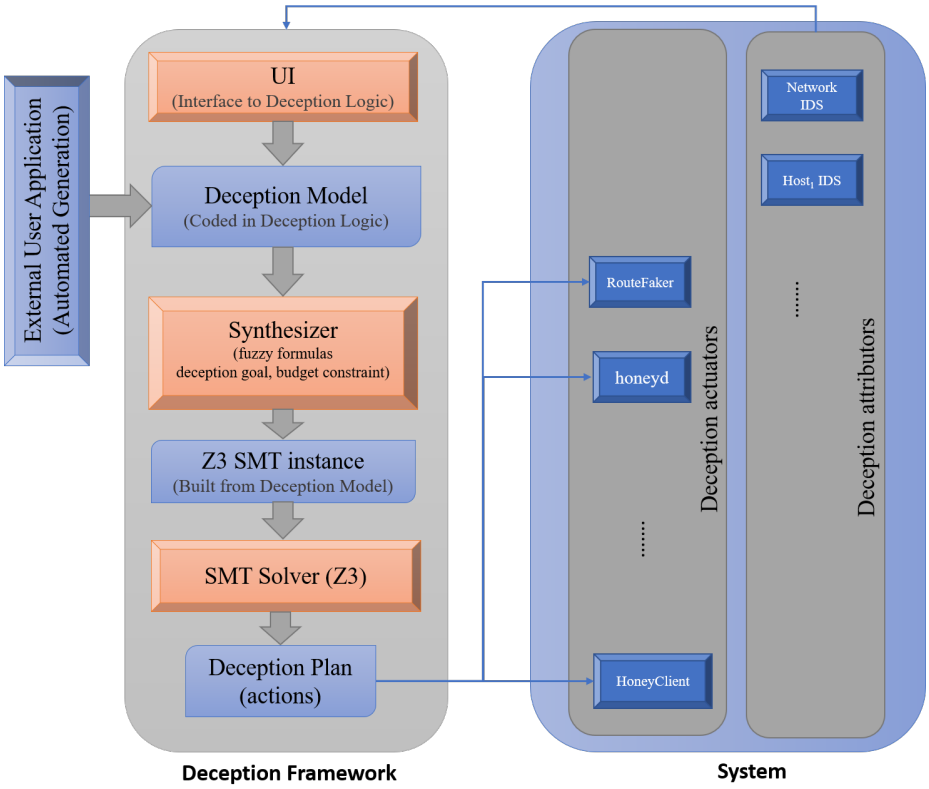

Figure 31: A schematic depiction of the deception framework architecture, components, and processes

a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. SMT formulas provide a much richer modeling language than is possible with Boolean SAT formulas. Although satisfiability problems are NP-complete in general, recent advances in SMT solvers have made them scalable to problems with millions of variables.

The synthesized deception model is, in fact, a Z3 SMT instance that is crafted by adding required parameters and constraints to the given deception model. Next, the framework uses Z3 SMT solver [12] to solve this instance and generate a *deception plan*. This plan determines the optimal set of deceptive actions that are needed to be proactively deployed (actuated) in the system to achieve the deception goal. The

deception plan is implemented in the given domain by *actuators*. These actuators are responsible for applying deceptive actions in the domain. They could be network devices that manipulate inbound or outbound traffic, or system-level processes running on network hosts that manipulate network packets, or the configuration of applications, memory, and other system entities.

After deployment of a deception plan in a system, attackers will interact with that system. These interactions, as well as other changes in the system, allow *attributors* to update the deception model's inputs regarding both attackers and the system. Attributors use inputs from sensors such as IDS, firewall, or other security devices or processes to provide feedback on attacker's current knowledge and changes in the system. Given these new inputs, the framework updates the deception plan. For example, assume that we observe an outbound connection from a honeypot. This means that this honeypot has been compromised and the attacker has privileged access to this system. So, the deception model is again solved by considering this new piece of information to construct a new adaptive plan.

3.5.1 Deception Modeling Logic

3.5.2 Modeling Reality

The deception model for a cyber threat against a system is defined based on *attributes*, where each attribute describes configuration parameters that could be manipulated by actions or parameters that an attacker may have beliefs about them.

These attributes depend on the involved parameters in the deception and change from one threat model to another. The set of attributes that describe a deception

model must be identified as part of defining the model, which is given to the framework as input. The set of attributes is denoted as $\{\varphi_1, \ldots, \varphi_m\}$.

3.5.3 Modeling Beliefs

An attacker's belief on an attribute denotes the state of an attacker's knowledge on that attribute. In other words, it shows the certainty that attacker is deceived about that attribute.

We define symbol D_i as a many-valued variable to denote the degree of certainty that an attacker is deceived on attribute φ_i. This belief variables is defined based on a finite-valued Gödel logic system, denoted as G_{11} [10]. Based on this system, $D_i \in T = \{0, 1/10, \ldots, 9/10, 1\}$ is a finite- and many-valued variable and T defines the set of truth values. For example, $D_{os_1} = 0.7$ means that an attacker is deceived about what OS a host 1 is running with certainty 0.7. Higher certainties define higher degree of deception.

Beliefs are divided into two categories. *initial beliefs* are those that their initial truth values are defined based on actions and attack models. In contrast, *derivative beliefs* are those that their truth values are calculated from initial belief values using the formulas defined by the causality rules. For each initial belief, a formula is defined over the set of actions and attack models as will be in defined in Section 3.5.8. These formulas are given to the model as input.

3.5.4 Modeling Attack models

The set of attack models is denoted as $\Theta = \{\theta_1, \ldots, \theta_n\}$ and given to the model as input.

For an attack model θ_i, The truth value associated with an attack model is denoted as $\chi_i \in T$ and represents the modeler's degree of certainty to observe such an attack model in the system.

Attack model consistency constraints. The modeler must also define consistency rules for attack models. These rules define the inter-dependency among attack models and are defined as follows:

$$\chi_i \leftrightarrow \neg\chi_j \tag{46}$$

For example, assume we have if two attack models of naive and advanced reconnaissance never happen together, then the following rule must be defined in the system:

$$\chi_{adv} \leftrightarrow \neg\chi_{naive} \tag{47}$$

3.5.5 Modeling Actions

An action denotes a potentially deceptive change in the system that explicitly manipulates an attacker's belief on one attribute. Initial beliefs are those for which the attacker's belief can be manipulated directly via a deceptive action.

An action is denoted as Integer variable α_j. The set of actions that could be assigned to an action variable α_j is given as A_j. To denote that no action is adopted on φ_j, we denote $\alpha_j = \emptyset$. The deception plan is essentially an assignment to all action variables in a manner that satisfies the given constraints of the model.

Action Consistency Constraints. The deception plan must present a coherent and consistent deceptive depiction of the system, or else it would result in an unconvincing deception. This consistency is modeled as a system-dependent set of constraints that

are incorporated into the model to define semantic interdependency among actions. The modeler defines action consistencies as follows:

$$(\boldsymbol{\alpha}_i = a_1) \leftrightarrow \neg(\boldsymbol{\alpha}_j = a_2) \tag{48}$$

where $\varphi_i, \varphi_j, \varphi_k$ are attributes and a_1 and a_2 are action values in A_i and A_j respectively.

Action costs. Applying deceptive actions on attributes are costly, as it requires manipulation of attribute values. We assume that these costs are all expressible as financial values. The cost associated with applying action $a \in A_i$ is denoted by a numerical value $c_i(a)$ which is given as input to the model. Only one action can be assigned to each actionable attribute.

Budget constraint. The cost of a deception plan is simply calculated as the linear summation of the cost of all actions of a deception plan.

The budget is given to the model as input, C_{max}. The framework must ensure that the aggregated cost of all actions does not surpass C_{max}.

$$\sum_i \sum_k c_j(k)(\alpha_j = k) \leq C_{max} \quad \text{added at synthesis} \tag{49}$$

3.5.6 Modeling Cause-Effect

Modeling cause and effect between actions, beliefs, and goals aims to model how actions manipulate an attacker's beliefs and then what combinations of these beliefs would successfully lead the attacker to the desired belief that is the goal. These desired beliefs, which are the goal of deception, would change the attacker's course of

action in the defender's favor. Modeling these relationships requires logical formulas for combining the connection between actions, beliefs, and goals.

A causality rule has the following general form:

$$\underbrace{(\boldsymbol{D}_1 \wedge \ldots \wedge D_j)}_{\textbf{scenario 1}} \vee \ldots \vee \underbrace{(D_l \wedge \ldots \wedge D_m)}_{\text{scenario n}} \leftrightarrow D_k \tag{50}$$

where D_i is a belief variable or a formula over belief variables using logical connectives $\wedge$, $\vee$, and $\neg$.

The left side is called antecedent, and the right side of the formula is called consequent. Consequent is a belief variable over attribute φ_k. Note that the antecedent is expressed in Disjunctive Normal Form (DNF).

3.5.7 Modeling Deception Goal

Conceptually, the intention of deception is to make an attacker believe a false statement about the system. In other words, this intention is to induce an attacker toward certain knowledge states. However, a deception model may have several conflicting goals. The goals have different benefits for different attack models or defense objectives.

The aggregate benefit is defined as the likelihood of a belief times impact of that belief. While the impact of a belief is a reality beyond our control, the likelihood or certainty of the belief is determined by the chosen deception plan. In fact, the main contribution of this deception framework is to manipulate belief likelihoods by manipulating an attacker's knowledge regarding them.

The impacts of a goal belief are defined for every attack model, and by considering

the mission and risk values of the enterprise. A variety of techniques such as OWASP risk rating methodology [28] could be used to this aim.

A deception model may have different goal beliefs, each with different impacts. Assume $G_1, \ldots, G_m$ denote the goal beliefs of the system which are a subset of all beliefs in the model. The impact associated with a goal belief G_i is denoted as a value I_i, that is given to the model as input. I_i has an impact value of 0 unless its value is stated as input to the model.

The goal of the deception model is given as the minimum acceptable benefit value that makes a deception plan effective. This minimum benefit is denoted as B_{min} and given to the model as input. The following constraint is added to the model at synthesis to calculate the benefit based on goal values:

$$\sum_i G_i \cdot I_i \leq B_{min} \tag{51}$$

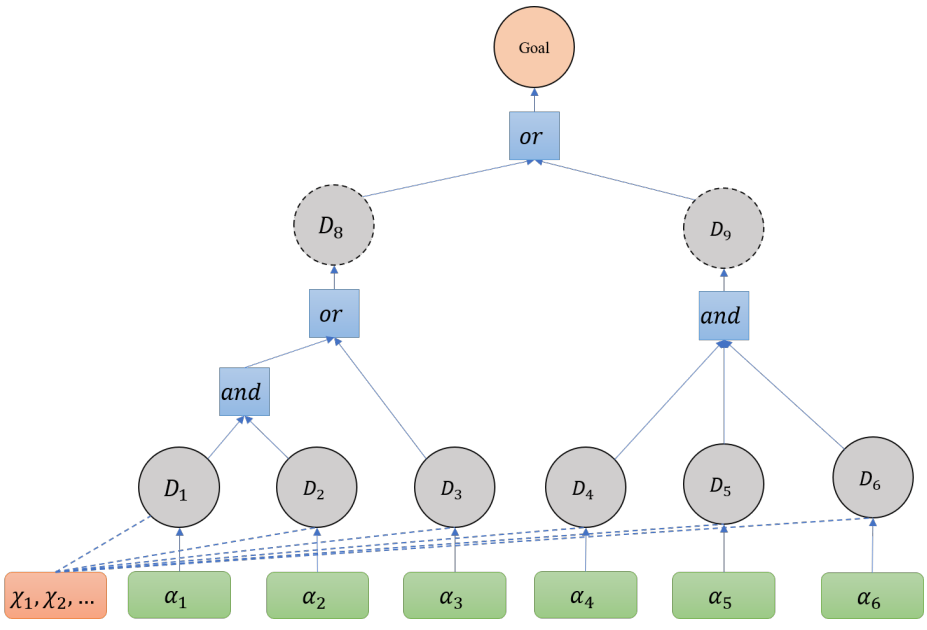

Figure 32: Example of a deception graph

3.5.8 Solving Deception Models

The deception model in essence defines a *deception graph* which shows the interdependencies among actions, beliefs and goals. Figure 32 shows an example of a deception model. The actions (and attack models) determine the initial beliefs; initial beliefs assign values to derivative beliefs, and the goal belief is assigned by the derivative (and also initial) beliefs.

To justify our choices for solving the deception model, assume the deception model of Figure 32. The goal belief is defined as a logical formula over initial belief variables, which in turn are determined by the actions. For simplicity, assume that we only have one goal belief G.

Therefore, given the action space $A = \{A_1, \ldots, A_m\}$ and attack models $\Theta = \{\theta_1, \ldots\}$ where A_i defines the set of action values for action α_i, the deception model defines a logical formula for G, which for a given deception plan denotes the truth value of G.

For example, in Figure 32 the goal could be defined as the logical formula:

$$((D_1 \wedge D_2) \vee D_3) \vee (D_4 \wedge D_5 \wedge D_6) \leftrightarrow G \tag{52}$$

Note that derivative beliefs do not appear in the goal formula. In other words, derivative beliefs are merely aliases for formulas based on initial belief variables. They allow the modeler to model a deception problem as a sequence of intermediary beliefs.

Every initial belief variable is defined based on action variables and attack models.

If we think of belief variables as a classic Boolean propositional variable, then an initial belief, like D_1 in the example, could be defined as follows:

$$D_1 \leftrightarrow (\alpha_1 = x \wedge \chi_1) \vee (\alpha_1 = y \wedge \chi_2)$$

If we assume that belief variables are classic Boolean propositional ones, then the problem of solving the deception model becomes assignments to all action variables α_i, such that G becomes $true$. Therefore, the problem of solving the deception model is a satisfiability problem. We solve this problem by defining a deception model using the satisfiability modulo theories [12]. SMT provides built-in support for arithmetic data types such as integers and real data types which allows us to expand satisfiability assignments from Boolean values to Integer and Real values.

Defining belief variables as Boolean is only possible if we assume a certainty of knowledge in deception modeling. However, such assumption would result in unrealistic deception planning. The problem of deception planning is, in essence, finding a plan that achieves the deception goal with a higher certainty. In other words, the goal is to select a plan to maximizes our certainty on the fact that the deception goal is realized.

Therefore, instead of defining belief variables as Boolean, we define them as many-valued variables. Instead of assuming that D_i (e.g., the attacker is deceived on attribute φ) is either $true$ or $false$, we can define it as a three-valued variable with truth values (F, U, T) where $D_i = U$ means that the attacker is deceived on φ_i with an unknown certainty. Defining D_i as a three-valued variable entails definition of a three-valued logic system that defines U and also provides interpretation for conjunc-

tion and implication. Several three-valued logic systems exist in the literature [38, 94] and the meaning of U differs from one system to another. This semantics defines the meanings of formulas such as $U \wedge F$ and $U \wedge T$. For example, the U state is thought of as neither true nor false as in the Kleene logic [38], but it is considered as both true and false in the Priest logic [94]. Different three-valued logics are appropriate for different types of systems and the choice of a meaningful three-valued logic for a domain depends on the application [97].

To achieve lower granularity in expressing uncertainty, a better design choice is to extend this three-valued logic and define belief variables D_i as many-valued variables. Again, there are many logic systems for defining whether D_i is finite- or infinite-valued variable, and for interpreting conjunction and implication. The correctness of this interpretation depends on the application and is an engineering decision. For example, we can use a many-valued product logic where conjunction is defined as product [49]. Another choice is the Lukasiewicz logic [24] where $u \rightarrow v$ is defined as $\min\{1, 1 - u + v\}$.

By considering a variety of many-valued logic systems, we find the Gödel finite-valued logic [10, 43] as the most appropriate for modeling deception for two reasons. First, the semantics of Gödel logic is considered to be suitable for formalizing *relative* comparison, as compared to logics such as Lukasiewicz [24] or product logic [49] which are more suitable for *absolute* or *metric* comparison. In other words, in Gödel logic, whether a belief D_1 evaluates to 1 or not depends on the relative ordering of the truth values of atomic formulas, not on the set of truth values or the specific values of the atomic formulas [10]. This relativeness in comparison makes Gödel logic very suitable

for deception modeling, because if for a plan F, we achieve $D_1 = 0.7$ and for a plan F' we achieve $D_1 = 0.6$, what is important for our reasoning is that the plan F is better than the plan F', rather than measuring the specific value of this difference.

Second, the min and max operators are good representatives for modeling the concept of conjunction and disjunction in the deception. For example, the believability of fingerprint of a decoy host that consists of a two decoy services of Web Server and a DB server could be defined as the minimum believability among all of its services. This is consistent with reality, because if a decoy host has a very believable Web server, but the DB server is not very believable as a real service, then believability of the decoy host is only as high as the DB server, which is the minimum.

However, in the same example, the believability that this decoy host is exploitable could be defined based on the disjunction which is the maximum operator. This is again consistent with reality because if the Web server has more vulnerabilities than the DB server, then the believability of exploitability of a decoy depends on the Web server, which is the maximum.

We define D_i as a k-valued Gödel logic [10, 43] system denoted as G_{11}, where the truth values are defined as $\{0, \frac{1}{k-1}, \frac{2}{k-1}, \ldots, \frac{k-2}{k-1}, 1\}$. We select $k = 11$ as a trade-off between solving time and granularity of defining uncertainty. Therefore, the set of truth values, T is defined as follows:

$$T = \{0, 1/10, \ldots, 9/10, 1\} \tag{53}$$

In the Gödel logic, the conjunction $\wedge$ and disjunction $\vee$ are defined respectively as

the minimum and maximum of the operands [43]:

$$u \wedge v := \min\{u, v\} \tag{54}$$

$$u \vee v := \max\{u, v\} \tag{55}$$

The negation and implication are also defined as follows:

$$\neg u = \begin{cases} 1 & u = 0 \\ 0 & u > 0 \end{cases} \tag{56}$$

$$u \rightarrow v = \begin{cases} 1 & u \leq v \\ 0 & u > v \end{cases} \tag{57}$$

For example, assume $u = 1/10$ and $v = 1/5$. Then, based on the G_{11} logic system, $u \wedge v = 1/10$ and $u \vee v = 1/5$.

Every deception goal could be rewritten as a logical formula over initial beliefs. This is because the deception goal is defined based on a formula over initial and derivative beliefs, and derivate beliefs are defined based on the initial beliefs. The formula for a goal belief G_k can be defined based on initial beliefs $D_1, \ldots, D_m$ with the following format:

$$(D_1 \wedge \ldots \wedge D_i) \vee \ldots \vee (D_j \wedge \ldots \wedge D_m) \leftrightarrow G_k \tag{58}$$

The initial belief values are as defined as formulas over action variables, action

values, and attack model values.

$$D_i \leftrightarrow (((\alpha_j = a_1 \wedge \chi_1 = b_1) \vee (\alpha_j = a_2 \wedge \chi_1 = b_2)) \wedge t_1) \vee \ldots \\ (((\alpha_j = a_3 \wedge \chi_k = b_3) \vee (\alpha_j = a_4 \wedge \chi_k = b_4)) \wedge t_n) \tag{59}$$

where $t_1, \ldots, t_n \in T$ are constant truth values that define initial beliefs values for various scenarios, $a_1, a_2, a_3, a_4 \in A_j$ are potential actions values of α_j, and b_1, b_2, b_3, b_4 inT are truth values that define certainty with regard to observing that attack model. Note that D_i is defined as a disjunction (maximum) of different assignment scenarios of the form $(((\alpha_j = a_3 \wedge \chi_k = b_3) \vee (\alpha_j = a_4 \wedge \chi_k = b_4)) \wedge t_n)$ where

$$((\alpha_j = a_3 \wedge \chi_k = b_3) \vee (\alpha_j = a_4 \wedge \chi_k = b_4))$$

defines an assignment scenario for which D_i is assigned t_n.

By combining Eq. 58 and 59 we can rewrite a goal as a logical formula over action variables, constant action values, and constant attack model values.

To solve the plan, the planner must determine assignment to action variables based on goal formulas. To achieve this, we rely on the support of *integer* data type and *nonlinear arithmetic* in Microsoft Z3 [98]. This is because the set of truth values T is an ordered set of 11 members, and min and max operators are non-linear.

Given a deception model, the framework solves the modeling by taking the following steps. In here, the instance refers to the Z3 SMT instance that is generated by the framework and based on the deception model.

Step 1: the framework generates the dependency graph of the model (Section 3.5.9). If the dependency graph is not a DAG, the model is invalid and will be rejected.

Step 2: the initial belief variable constraints (Eq. 59) are added to the model.

Step 3: given constraints for initial beliefs D_i, the framework rewrites the goal belief by replacing derivative beliefs with their logical formulas until no derivative belief exists in each goal formula, as exemplified in Eq. 52.

Step 4: using G_{11} Gödel logic interpretations of conjunction and disjunction, the framework adds a goal formula for every goal such as G_k as follows:

$$G_k \leftrightarrow \max\{\min\{D_1, \ldots, D_i\}, \ldots, \min\{D_j, \ldots, D_m\}\} \tag{60}$$

For each initial belief D_i, the framework is given a formula as defined by Eq. 59. Using the same approach as above, the framework converts this formula by redefining conjunctions and disjunctions with min and max operators.

Step 5: given impact values for goal beliefs and a minimum benefit, the framework adds the goal constraint of Eq. 51 to the model.

Step 6: given action costs and a budget, the framework adds the budget constraint of Eq. 49 to the instance.

Step 7: the framework adds action consistency constraints (Eq. 48) and attack model consistency constraints (Eq. 46) to the instance.

Step 8: the framework uses Z3 to solve the generated instance. The result is an assignment to action variables that satisfies the goal and budget constraints, in addition to all the consistency constraints of the model. If no satisfiable assignment to the instance is found, the budget and minimum benefit values must be relaxed.

Figure 33 shows some belief values for a deception plan.

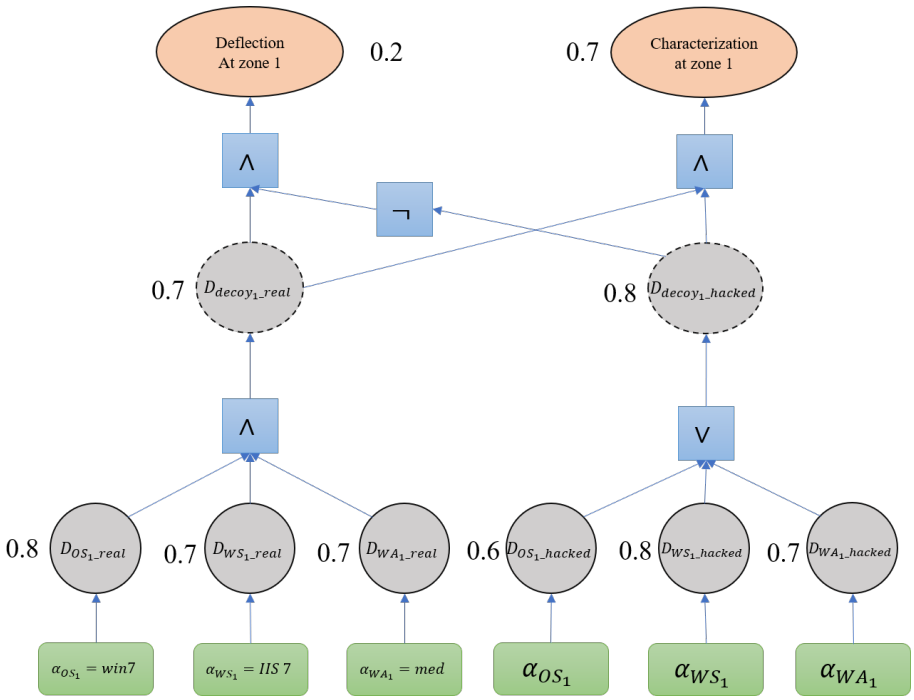

Figure 33: An example of belief values for a deception plan

3.5.9 Assumptions for Modeling

In the previous section, we showed that a deception goal could be written as a logical formula over initial beliefs. This is because the deception goal is defined by on a formula over initial and derivative beliefs. The derivate beliefs are defined based on the initial beliefs, and initial belief values are determined based on values assigned to action variables in addition to the given attack model values (Eq. 59). Therefore, in a well-formed logic with no circular dependency among belief variables (more detail on this later), a goal belief variable could be rewritten as a logical formula over action variables, constant action values, and constant attack model values.

Given this, to make the logical deception model well-formed and meaningful, the given deception model must satisfy the following assumptions:

Assumption 1. *Actions must be independent, except for dependencies that are ex-*

plicitly defined by action consistency constraints. Action variables, constant attack models values and constant truth values in Eq. 59 are the atomic formulas in our logic. An atomic formula is a formula with no deeper propositional structure; that is, a formula which contains no logical connectives. In our logic, this means that action variables, attack model values, and initial belief values must be atomic; in other words, there must be no hidden relationship between pairs of action variables, pairs of attack model values, and pairs of truth values in formulas for initial beliefs.

For action variables, if assignment of a value a_1 to an action variable α_i prohibits or entails assignment of action a_2 to action variable α_j, this dependency must be defined explicitly using action consistency constraints (Eq. 48). In the security domain, this assumption is justified because based on the definition of attributes, every deceptive action of the model affects one configuration parameter of the system. However, when there are consistency dependencies among the configuration parameters (e.g., if OS is Linux, the Web server must not be IIS), this dependency is explicitly defined in the model. Satisfying this assumption ensures that action variables are well-formed atomic formulas with no deeper propositional structure except those defined by the action consistency rules.

Assumption 2. *Attack model values must be independent, except for dependencies that are explicitly defined by attack model consistency constraints.* Based on the discussion for assumption 1, attack model values that are constant and given to the model as input must be independent, unless their dependency is explicitly defined by attack model consistency constraints (Eq. 46). For example, if for a deception model we have two attack models nave and advanced, then we need to define $\chi_{naive} \leftrightarrow \neg\chi_{advanced}$.

However, if for a deception plan that aims to defeat two disjoint threat models such as DDoS and intrusion attacks together, no dependency exists between the two models and no attack model consistency constraint is defined for them.

Assumption 3. *Initial belief values must be independent.* A pair of (action, attack model) such as (α_j, χ_k) may contribute to several initial beliefs. If the formulas for two initial belief D_i and D_j include the same pair of (action, attack model) such as (α_j, χ_k), belief value assignments must be determined based on independent properties of that pair. For our example of Chapter 4, we derive two beliefs from each action. The first belief models *believability of fingerprint* and the second belief models *believability of exploitability.*

In our discussion in Section 4.2.4, we show that these two beliefs are derived from different sources. Believability of a service is derived from the frequency of occurrence of that service in typical enterprise networks (global believability) as well as the given network (local believability). Eq. 76 shows this formula. In contrast, the believability of exploitability is derived from the vulnerability scores of that service, as represented by Eq. 77. Therefore, these two beliefs are defined based on the same action but based on independent sources of information. Still, there might be some hidden correlation between the two beliefs; for example, older services are less believable and at the same time have higher vulnerability scores (for example, Windows XP). However, such correlations must be proven to be small and negligible. Methodologies for the derivation of these initial beliefs must be defined in a manner than ensures no or negligible dependency among these beliefs. This entails development of techniques for quantifying this correlation, which is left to future work.

But when this correlation is not negligible, the two beliefs must be defined as one belief. For example, if *believability of fingerprint* and *believability of exploitability* are dependent, instead of defining them as two individual beliefs, we must define them as one single belief which shows *believability of fingerprint and exploitability.*

Assumption 4. *The belief dependency graph must be directed and acyclic (DAG).* From the definition of causality rules in Eq. 50, we define the *dependency graph* of a deception model as a graph G where belief variables are nodes and there exists an edge from D_i to D_j if and only if D_i appears as in the antecedent of the rule for which D_j is the consequent.

To be able to rewrite a deception goal as a logical formula over belief variables as in Eq. 58, this dependency graph must be a directed acyclic graph (DAG) with no loops. This assumption is reasonable as it is the assumption for other belief networks such as Bayesian [71].

Assumption 5. *Each initial belief is derived from at most one action.* To simplify the definition of the deception model, we assume that only one action can contribute to an initial belief (Eq. 59). This assumption is not necessary since assumption 4 could be extended to define scenario pairs as (action 1, ..., action k, attack model). However, this will complicate the correlation analysis of the initial beliefs in the modeling, because several actions may contribute to a belief, and at the same time several beliefs might be affected by each action. In the deception model, if a set of action variables contribute to a belief, the modeler must define that set of action variables as one single action. However, in this modeling assumption 4 must be satisfied; i.e., if two beliefs depend on the same action, initial belief values must be

defined based on independent properties of that action.

3.5.10 Modeling Adaptability

Modeling adaptability refers to the process of adapting the deception plan to recently collected information regarding attacker's beliefs or system changes. The attributors in the domain provide attribution information to the framework, as depicted in Figure 31.

By analyzing the collected logs from IDS or firewalls, attributors update our initial assumptions about attacker's knowledge or type. For example, assume in our IDS logs, we observe a flow with a decoy as its source address. From this, we know that the honeypot has been taken over by an attacker, and our initial assumption about the certainty of the belief that *attacker believes exploitation of decoy i* is changed. We update the initial values for this belief and solve the deception model again. This can be done by an attributor that checks the outbound connections from decoys and if an outbound connection is observed assigns the initial value of 1 to this belief.

The attributors can also update the inputs as a result of changes in the system. Assume the deception model which is presented against APT in Chapter 4. As an input to this model, we provide the real configuration of the enterprise network, and the deception plan is determined based on this given configuration. Now, assume a new host is added to the network. This will change the input to the deception plan, and therefore the deception model must be updated and solved again. To this aim, we can develop an attributor that probes the enterprise address space at regular intervals to identify if a new host has been added or not. The attributor provides

this information to the framework at regular intervals. If the framework notices a change in the system configuration, it will solve the model with the new inputs to find a deception plan that is adapted to these changes in the network.

The main drawback of this approach is that solving an updated deception model, especially for a large system, requires non-trivial computation time. This limitation may not allow the framework to update the deception plan adaptively. To address this limitation, several deception plans could be determined in advance, each appropriate for various sets of scenarios that are expected to be recurring in the system. The monotonicity assumption [4], i.e., the attacker would never willingly give up previously attained knowledge, could be used to decrease the number of scenarios.

3.6 Evaluation Metrics

The framework needs to be evaluated from three dimensions.

Usability: for the framework to be usable, it must be able to express a wide variety of deception models. Understanding what kind of deception problems are expressible by the model requires a comprehensive investigation of potential deception solutions against various cyber attacks. Usability shows how intuitive and straightforward it is to express a deception model against a specific threat model using our framework. We claim the framework has high usability due to its intuition for modeling deception: how an attacker's observations affect her beliefs, and how this can be used to lead the attacker to a conclusion in the defender's favor. This definition of deception follows human cognitive thinking process, thus allowing complicated deception paradigms to be expressed as a cause and effect relationship among beliefs. In Chapter 4, we

model a complex deception problem in cyber domain to exemplify the usability of our model.

Computational Overhead: It refers to the cost of synthesizing and solving the model. The synthesis process consists of adding formulas for calculating truth values and also including action and goal constraints. All these steps have polynomial complexity.

The goal of deception modeling is to determine an optimal deception plan that satisfies the given constraints. Each action variables can have some options; assuming that a model has n actions, and each has k different option values to be assigned to, the deception planning problem is a selection of one of at most k^n alternative deception plans.

To see that the deception planning problem is NP-hard, we reduce it to the *0-1 knapsack problem*. The general knapsack problem is defined as follows: given a set of items, each with weight and value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The most common type is the 0-1 knapsack problem, which restricts the number of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W, the 0-1 knapsack problem is defined as follows:

$$\text{maximize } \sum_{i=1}^{n} v_i x_i \tag{61}$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \tag{62}$$

Accordingly, our problem could be defined as:

$$\text{maximize} \sum_i D_i \cdot I_i \tag{63}$$

$$\text{subject to} \sum_j \sum_k \boldsymbol{c}_j(k) \cdot (\alpha_j = k) \leq C_{max} \tag{64}$$

Therefore, the deception modeling problem is NP-hard, and therefore we convert the problem to a satisfiability problem, using generalized Boolean/arithmetic format of satisfiability modulo theories (SMT) [12]. SMT formulas provide a much richer modeling language than is possible with Boolean SAT formulas. Although satisfiability problems are NP-complete in general, recent advances in SMT solvers have made them scalable to problems with millions of variables [87].

Solving the model by the Z3 incurs the main computational cost. The actual cost depends on the number of attributes and action variables in that model and number and boundaries of constraints in the model.

Our modeling is computationally efficient because although probability calculation results in non-linear arithmetic, the only variables that must be directly assigned by the solver are belief and action variables. Truth values are driven by assignments to action variables. Moreover, by assuming that a truth value set is discrete, the state space of a belief/action variable could be reduced tremendously.

Effectiveness: Deception is a proactive defense paradigm. In Chapter 2 we showed that this effectiveness could be calculated in terms of *deterrence* and *deception* ratios. This is because, as discussed in 3.5.7 the deception goals basically aim to either deflect or characterize the attacker, or both. The deflection goal could be measured

by *deterrence ratio*, while the *characterization* goal could be measured by *deception ratio*. Therefore, the effectiveness of a deception model could again be explained regarding how much deterrence or deception is incurred on an attacker.

One of the main challenges in evaluating the framework is that since the deception plan aims to defeat human attackers, realistic evaluation of the effectiveness or goodness of deception plans could only be achieved by testing it against human attackers. This is one of the main future research works for this project.

CHAPTER 4: DECEPTION PLANNING AGAINST MULTI-STAGE APT ATTACKS

In this chapter, we use our deception planning framework that is introduced in 3 to construct a deception plan against multi-stage intrusion attacks. Note that this example primarily serves as a hypothetical case study. While we provide values for impacts, costs, and initial beliefs, there is no way to demonstrate the validity of these value assignments, unless the deception plan is tested in red-teaming experiments and with real human attackers. This serves as an example that demonstrates hypothetical benefits of the deception framework, and also provides intuition for defining the model and calculating initial beliefs, impacts, and costs for a deception model.

In this Chapter, we use the term *decoy host* to denote a host with an arbitrary fingerprint that includes a decoy OS and a number of decoy services. This is different from the shadow decoys introduced in Chapter 2 which mimic the fingerprint of a real host.

4.1 Problem Statement

Our goal is to deflect attackers from critical network hosts and also increase the certainty that attackers with different levels of sophistication are characterized.

Assume a typical Enterprise network with zoning for defense-in-depth purposes. Specifically, the network is partitioned into several zones, where each zone has its access control policies. For example, the DMZ is externally accessible from the In-

ternet, while the secure internal network is highly restricted and only accessible by certain servers.

Our threat model is an attacker whose objective is to infiltrate the network to reach the critical hosts, which are usually in the secure internal zones that are not accessible externally. This is the same attack model as the one described in Chapter 2. Due to zoning, the attacker must initiate their attack by compromising a publicly accessible host. Starting from the DMZ, they gradually continue compromising hosts, and moving from one host to another, and from one zone to another, until they finally discover a path to critical network hosts. Figure 34 shows an exemplary network with potential attack paths from the DMZ to the restricted zone.

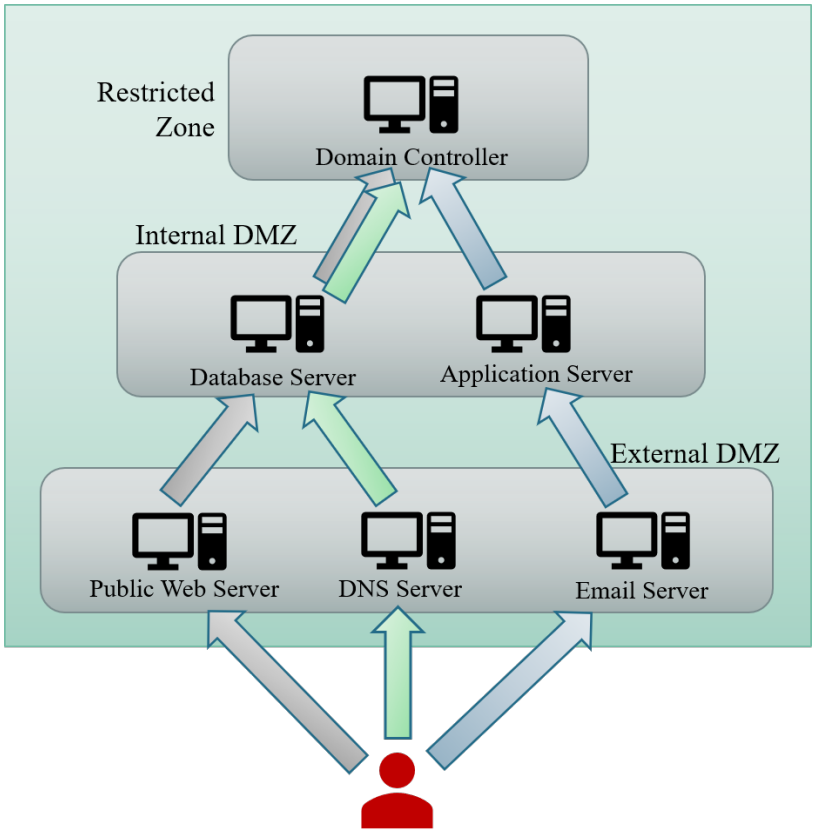

Figure 34: An exemplary network and potential attacks paths

The objective is to complicate an attacker's path toward critical network hosts by placing carefully-configured decoy hosts in strategically-selected locations of the network. In other words, we want to extend the attack graph with deceptive paths in a manner that the certainty that an attacker compromises a critical host decreases. By luring the attacker to traverse these fake paths, we waste an attacker's resources and time. Moreover, by trapping the attacker in decoys, we can characterize the attacker's motives and strategies. Our goal is to determine a deception plan that achieves these goals with a high certainty and bounded cost.

Figure 35 shows the attack graph for the network of Figure 34 extended with deceptive paths for a specific deception plan that consists of two decoys in the external DMZ, 1 in the internal DMZ, and 1 in the restricted zone.

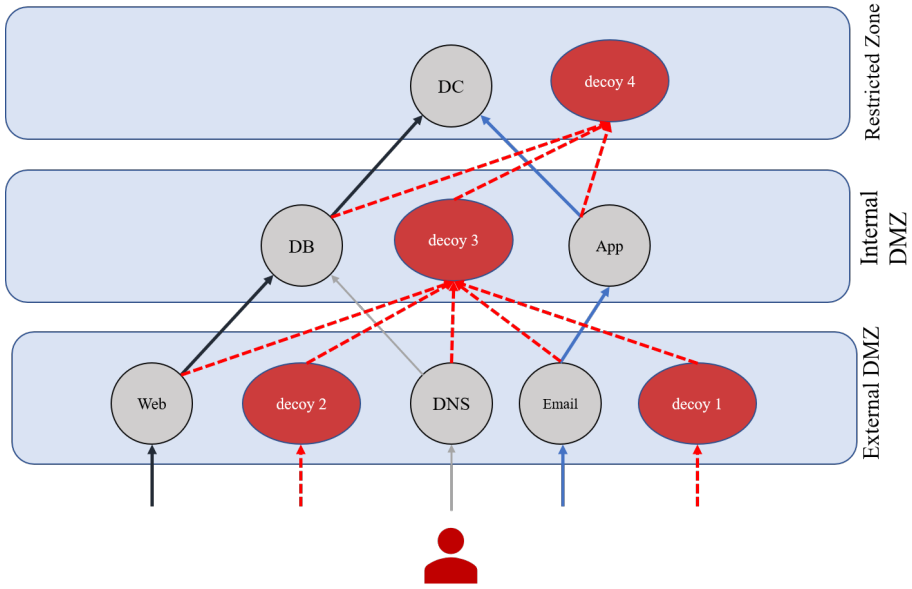

Figure 35: An attack graph extended with deception paths

However, instead of determining the fingerprint and the location of decoy hosts randomly, our goal is to determine a synergistic and also consistent combination of

these decoys, to orchestrate a deception plan that is tailored to the goals as well as the attack graph of the network. Given the set of critical network hosts, their fingerprints, and their locations, the deception model aims to address two fundamental questions:

1. *What decoy services must be located on each decoy host for achieving desired protection?* Each decoy service is modeled as a deceptive action.
2. *Where must each decoy host be placed?* The location of each decoy host refers to the zone in which it is located.

4.2 Deception Model

Each routable and unused IP address in the address space of the network is a candidate location for a decoy host placement. Assume network includes m routable and available (unused) IP addresses. The network also includes n hosts, and each real host belongs to a zone. The network has been partitioned into z zones. Hosts in zone i can access only zones in zone i and $i+1$ and zone 1 is accessible externally.

Table 1: List of Attributes

attribute	**description**	**Initial**	**Deriv.**	**goal**
$real_{i,j}$	an attacker believes the decoy service j on decoy host i is real	✓	×	×
$hacked_{i,j}$	an attacker believe the decoy service j on host i is compromised	✓	×	×
$real_i$	an attacker believes decoy host i is real	×	✓	×
$hacked_i$	an attacker believes that decoy host i is compromised	×	✓	×
$deflected_i$	attacker is deflected by decoy i	×	×	✓
$char_i$	attacker is characterized by decoy i	×	×	✓

Every decoy host i can host a number of decoy services and decoy applications. Each class of decoy service (OS, Web Server, SSH Server, Web application) is de-

scribed by a specific index j. Examples of decoy service classes are operating system, Web server, Web application, SSH server, etc.

4.2.1 Attributes

Figure 36 shows a part of the deception graph for the given deception model. Action variables define decoy services for hosts. If no decoy OS is assigned to a decoy host, then that address is assumed to remain unused.

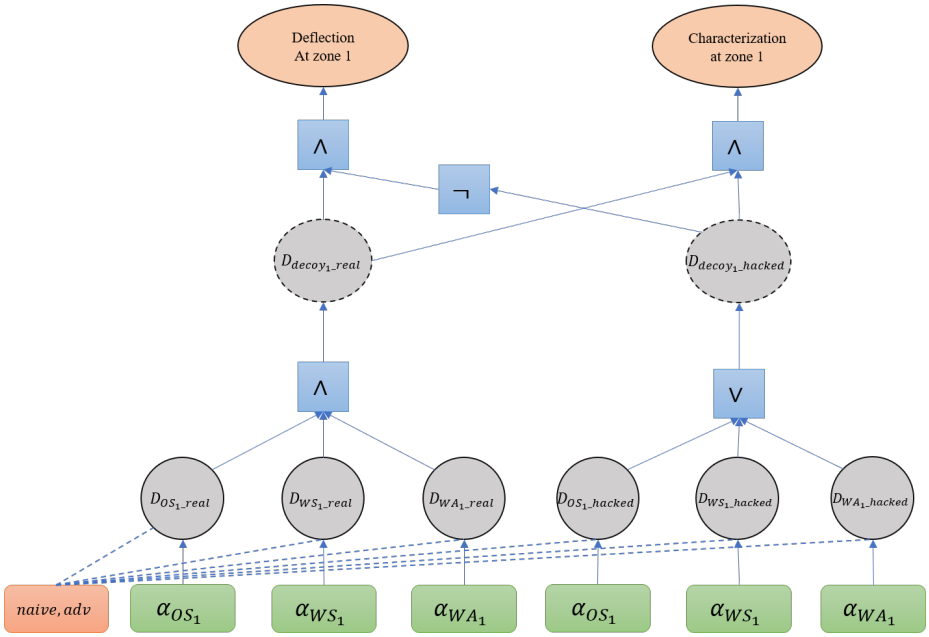

Figure 36: An example deception graph for one decoy host

In the modeling of a deception problem, the first step is to determine the system attributes. Table 1 describes system attributes, as well as their types and real values.

Attributes $real_{i,j}$ describe the believability of a decoy service of class j on decoy host i as a real service. For example, $real_{i,1}$ and $real_{i,2}$ describe this for the operating system and Web server of the decoy host i. Accordingly, $real_i$ shows attacker believes that decoy host i is real.

Attribute $hacked_{i,j}$ denotes the believability of exploitability of a service j of host

i. Also, $hacked_i$ denotes whether decoy host i is considered by the attacker as compromised.

Finally, attribute $deflect_i$ denotes the goal belief that an attacker is deceived and deflected by decoy i. Accordingly, $char_i$ denotes the goal belief that an attacker is deceived and characterized by decoy i.

Deceptive Actions. For each dark address, the deception plan must determine whether it will be designated as inactive or active.

The system must determine the fingerprint of each decoy host regarding operating system and services. Action variable $\boldsymbol{\alpha}_{i,j}$ is the action variable that denotes the decoy service of class j on host i. Specifically, $\boldsymbol{\alpha}_{i,j} = \emptyset$ is a truthful response which means no decoy service of type j is running on i.

Consistency constraints define inter-relationships among values associated with action variables, such that the deception plan is meaningful and believable. To this aim, there must be compatibility between the decoy OS and the decoy services on a decoy host. These rules can be automatically generated from a list of compatible OS and services. For example, the following constraint denotes that if OS is *Linux*, the Web server can not be *IIS* Web server.

$$(\alpha_{i,1} = Linux) \leftrightarrow \neg(\alpha_{i,2} = IIS) \tag{65}$$

The deployment cost of a deception plan depends on the cost of implementing the incorporated actions of that plan. This cost is calculated by considering the resources (memory, CPU, etc.) which that service requires, and its installation and maintenance overhead. Figure 37 shows some exemplary values for action costs for a number of

services.

4.2.2 Attack Models

The attack could be either naive or advanced: $\Theta = \{naive, adv\}$. We assume that these two attack models are independent. In other words, the attacker may use a combination of both naive reconnaissance and exploitation techniques and advanced ones.

4.2.3 Causality Rules

The causality rules are defined in Eq. 66 to Eq. 69.

Believability of Service

$$(\bigwedge_j D_{real_{i,j}}) \leftrightarrow D_{real_i} \tag{66}$$

Exploitability of Service

$$(\bigvee_j D_{hacked_{i,j}}) \leftrightarrow D_{hacked_i} \tag{67}$$

Goals

$$(\neg D_{hacked_i} \wedge D_{real_i}) \leftrightarrow D_{deflected_i} \tag{68}$$

$$(D_{hacked_i} \wedge D_{real_i}) \leftrightarrow D_{char_i} \tag{69}$$

Deception Goal. To show how different goals lead to different deception plans, we consider two goals. The first goal is to deflect attackers from critical network hosts, and the second goal is to characterize the attacker's techniques and exploits. Both goals require a deception that is believable. Therefore, believability is a factor that contributes to both goals.

However, while the characterization goal requires decoy hosts that are easy to compromise, the deflection goal requires decoy hosts that are difficult to compromise. This would increase the attacker's effort on them, and also increases the potential that the attacker is dissuaded from attacking real hosts.

Goal A: Deflection goal. The goal is to deflect attackers from critical hosts in a zone by placing decoys in that zone. To define the goal we need to associate impact values with goal beliefs. The goal belief for a deflection by decoy i is defined by $deflected_i$. Given these impacts, the following goal constraint is added to the model at the synthesis.

$$\sum_{i=1}^{m}(D_{deflect_i} \cdot I_{deflect_i}) \geq Bn_{min} \text{ [added at synthesis]} \tag{70}$$

The aggregate benefit is the summation of the benefit achieved as a result of deflecting attacks by all the decoys. The minimum acceptable benefit is defined by the threshold Bn_{min}.

Goal B: Characterization Goal. The goal here is to maximize an attacker's engagement with decoy hosts at various zones to increase the potential of characterizing an attacker's goals and motifs. The goal belief for engagement with a decoy i is defined as $char_i$. Given the impact value I_{char_i}, the following goal constraint is added to the model at synthesis:

$$\sum_{i=1}^{m}(D_{char_i} \cdot I_{char_i}) \geq Bn_{min} \text{ [added at synthesis]} \tag{71}$$

4.2.4 Numerical Inputs

Impact Values for Deflection Goals. The impact of deflecting an attacker by engaging him with a decoy i depends on two factors: (a) how reachable address i is by an attacker, and (b) criticality of hosts in the zone k to which i is assigned.

Reachability of address i by attacker could be modeled by the likelihood that attacker has access to the zone k, denoted as r_k, multiplied by the likelihood that decoy i is probed by the attacker in N scans, denoted by p_i. As a simple heuristic, we assume that a host in zone k can be reached with likelihood

$$r_k = \frac{1}{k^2} \tag{72}$$

This is based on the assumption that an internal zone k is less reachable than the zone $k-1$ because it is only accessible after the attacker exploits a real host at zone $k-1$. A better approach is to determine the reachability likelihood of a zone based on the vulnerability scores of real hosts and attack paths in the attack graph. This extension provides a more realistic modeling of zone reachability likelihood, and it is left to future work. The probability of probing a host in the next scan is calculated based on the characterization framework introduced in Section 2.3.6.

The characterization framework determines the potential scan probability of an address from the sequence of observed scans to unused addresses of the network. For an address i, this probability is denoted as π_i (Section 2.3.6).

The second factor, criticality of hosts in a zone could be determined by considering the financial values of those hosts. As a simple heuristic, assume $v_j \in [100, 500]$

denotes the financial value of a real host j. The criticality value of a zone k is denoted as c_k and calculated as:

$$c_k = \sum_j v_j \tag{73}$$

where j is a host in the zone k.

The impact of the deflection goal by a decoy i in a zone k could be defined as:

$$I_{deflected_i} = \pi_i \cdot r_k \cdot c_k \tag{74}$$

This definition is, in essence, the definition of risk where v_i denotes the impact, and $p_i \cdot r_k$ denotes the likelihood. These impact values reflect the significance of a zone.

Impact Values for Characterization Goals. The impact of engagement with a decoy i host at zone k is determined by how reachable that decoy host is by an attacker, plus the value of characterization at that level. Assume a characterization at zone k has a value of $k*100$. This is based on the observation that characterization at zone k is more valuable than characterization at zone $k-1$. The impact of characterization belief is defined as:

$$I_{char_i} = p_i \cdot c_k \cdot (k \cdot 100) \tag{75}$$

Initial Belief Values for Believability of Services. Table 37 denote how initial belief values are determined based on the chosen action. The truth values for believability and engagement are given in the table for cases where $\chi_{adv} = 1$ and $\chi_{naive} = 1$ respectively.

The truth values for believability of the fingerprint of a service is calculated based

Type of honey OS/Service/App	Honey OS/Service Instance	Believability of Fingerprint		Believability of Exploitability		Cost of lying action ($)
		Naïve	Advanced	Naïve	Advanced	
Operating System	Windows XP SP 1	1	0.2	0.8	0.9	50
	Windows 10	1	0.8	0.8	0.6	80
	Redhat 7	1	0.6	0.8	0.4	100
Web Server	IIS 6.0	1	0.2	0.8	1	50
	IIS 7.0	1	0.9	0.7	0.8	80
	Apache Tomcat 8	1	0.5	0.7	0.9	100
Web Application	Highly vulnerable	1	0.3	0.6	0.9	25
	Medium vul.	1	0.8	0.4	0.7	50
	Low vulnerable	1	0.9	0.2	0.6	100

Figure 37: An example of decoy OS and services with believability of fingerprint and exploitation values

on actions values and attack models. To this aim, we use the following formula for calculating believability values for a pair of (action value, attack model value) pair as defined in Eq. 59. For a pair of $(\alpha_i = a, \chi_j = b)$ the truth value is defined as:

$$\lambda_{a,j,b} = (\frac{global_a^j + local_a^j}{2}) \cdot b \tag{76}$$

where $global_a^j$ denotes the general believability of having a decoy service a against an attacker of type θ_j, and $local_a^j$ denotes this believability specifically for the given Enterprise network. Parameter $global_a^j$ is independent of the input network and to calculate it we investigate the fingerprints of a large number of machines for both cases where the attacker is using naive or advanced reconnaissance. We calculate $global_a^j$ as the ratio of hosts that are running service a. If no instance of a is observed in the network, then this value is 0. To differentiate between the two attack models, we assume that an advanced attacker can identify critical services from non-critical ones, and only calculates $global_a^j$ by only considering such services.

In contrast, $local_a^i$ depends on the host configurations of the given network. It is calculated in the same manner, but only by considering the fingerprints of the input network.

Initial Belief Values for Exploitability of Services. The values for believability of how exploitable a service could be is calculated based on the vulnerability level of that service. This is based on the assumption that the more vulnerable a service is, the more believable its exploitability is to an attacker. To this aim, we propose the use of *Lai and Hsia*'s model [73] which is an extension of CVSS score metric. For a pair $(\alpha_i = a, \chi_j = b)$, the believability of exploitability is defined as:

$$\gamma_{a,j,b} = \frac{\overbrace{(\sum_{l=1}^{k_a^j} v_{a,l} \times t_{a,l}) \times y_a}^{s_a}}{\max_i(s_a)} \tag{77}$$

where k_a^j denotes the number of known vulnerabilities that belong to an attack model of type j for the decoy service a, $v_{a,l}$ is the CVSS Base Score for vulnerability j, $t_{a,l}$ is the weight of the threat class for vulnerability j, and y_l denotes the asset weight of service l. This value is normalized to a value in the set of truth values, denoted as T. These values are different from different attack models because more vulnerabilities are known to an advanced attacker than a naive one. Figure 37 exemplify some of the initial belief values. These values are hypothetical and have been intuitively assigned by relative comparison of the considered services regarding believability of fingerprint and exploit. In future, we will improve our model by using Eq. 76 and 77.

4.3 Analysis of Deception Plans

4.3.1 Analysis of Deception Plans against Single-Stage Attacks

We first analyze the deception plans for a network with only one zone and then extend this analysis to networks with more zones.

In this thesis, we consider a small network with $n = 10$; that is, the DMZ address space has 10 unused addresses that could be used for deception. In order to visualize our results, we divide all potential decoy hosts into four groups based on their fidelity level that is defined based on the vulnerability score as follows:

$$fidelity_i = 1 - \max_{a \in i} \{\gamma_{a,adv,1}\} \tag{78}$$

This value is calculated based on the given beliefs for an advanced attacker (denoted as adv) with a certainty of 1, and a denotes decoy services that are assigned to decoy i.

- Very low fidelity: decoy hosts with fidelity values in $[0, 0.1, 0.2]$.
- Low fidelity: decoy hosts with fidelity values in $[0.3, 0.4, 0.5]$.
- Medium fidelity: decoy hosts with fidelity values $[0.6, 0.7, 0.8]$
- High fidelity: decoy hosts with fidelity values in the set $[0.9, 1]$.

Figures 38 and 39 show two different deception plans for a deflection goal with two different budgets and various distributions of attack models. First, note that for both budgets, as attack model becomes more advanced (left to right), the fidelity levels of decoys increases. However, for advanced attackers and a limited budget of \$800, the number of decoy host decreases (more addresses are unused), but still

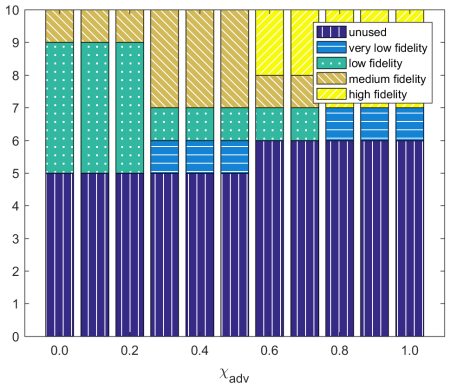

Figure 38: deception plan for deflection goal with budget = $800

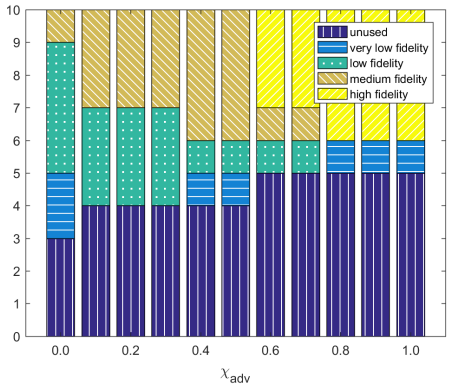

Figure 39: deception Plan for deflection goal with budget = $1000

the budget is spent on creating high-fidelity decoy hosts. This is intuitive because the deflection goal requires believable yet high fidelity decoys especially for advanced attackers. When the budget is increased to $1000 (Fig. 39), the number of unused addresses decreases (more decoy hosts are generated), and for the same distribution, the decoy hosts have higher fidelities.

Figures 40 and 41 show deception plans for the characterization goal and for the scenarios with the same budgets. Note that contrary to deception plans with the deflection goal, decoy hosts are built using lower fidelity services. This is also intuitive because the characterization goal requires vulnerable decoy services that are easy to compromise, but not easy to discover. Also, note that as attacker becomes more advanced, the fidelity level of the decoys increases to account for the fact that an advanced attacker has better techniques to differentiate decoys from real hosts.

To show that this deception plan is better than random planning, we compare the generated deception plan with three different planning strategies and for the deflection goal:

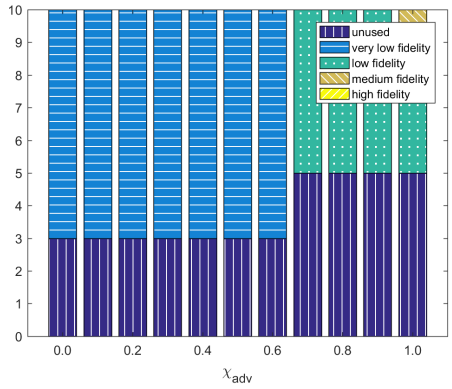

Figure 40: deception Plan for characterization goal with budget = $800

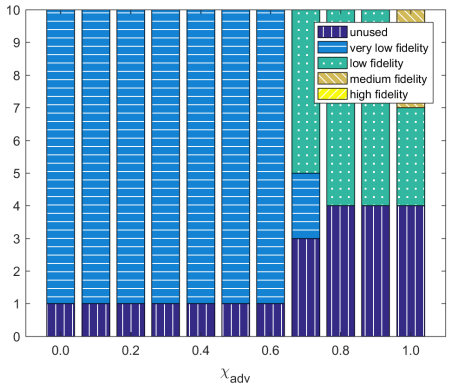

Figure 41: deception Plan for characterization goal with budget = $1000

- *Uniform:* a plan where the budget is uniformly distributed among the four classes of decoys.
- *All low-fidelity*: a plan that only consists of low-fidelity services with the given budget.
- *All high-fidelity:* a plan that only includes high-fidelity decoy hosts with the given budget.
- *Relative plan*: a plan that distributes the budget proportionate to the sophistication level of the attacker.

Figure 42 shows the benefit that is achieved from each of these plans, according to the impact values for goal beliefs that were given in the deception model. Note that the generated deception plan outperforms all other four competing plans.

4.3.2 Analysis of Deception Plans against Multi-Stage Attacks

We assume that $m = 64$ for every zone and $z = 3$. The most critical services are assumed to be in zone 3, which is the restricted zone. Also, services in zone 2, which is the internal DMZ, are more critical than those in zone 1, that is the external

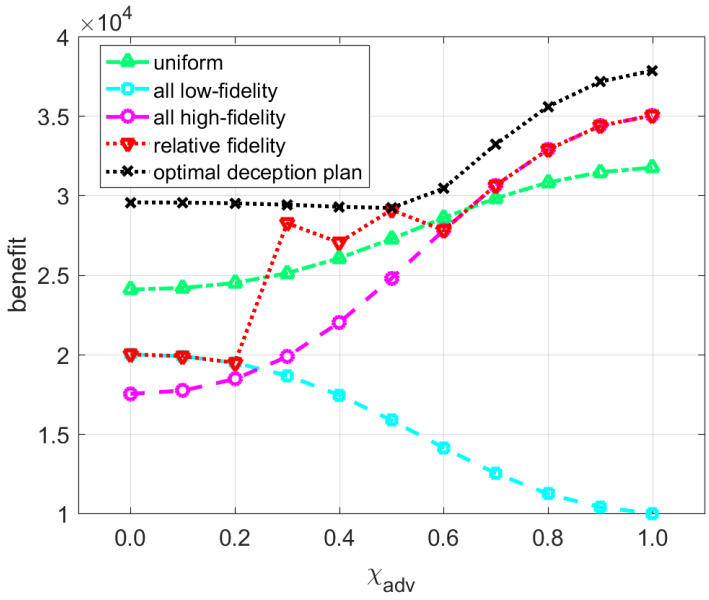

Figure 42: Comparison of the framework's deception plans with alternative scenarios

DMZ. The network includes $n = 30$ real hosts that are uniformly distributed across three zones. The hosts are also divided into three categories regarding criticality and monetary values. The highly critical hosts have high values (benefit impact) of \$500, the medium critical hosts have a value of \$250, while the non-critical hosts have a value of \$100. The zone $k = 1, 2, 3$ can be reached with likelihood $1/k^2$. The critical hosts are in zone 3, the medium hosts are in zone 2, and the non-critical hosts are in zone 1.

To visualize the deception plans and investigate their intuitions, we analyze each plan concerning the number and fidelity values of the decoy hosts that are assigned to each zone. We calculate a decoy host fidelity value based on the fidelity of the decoy services associated with it as defined in Eq. 78.

Figure 43 shows the number of decoy hosts that are assigned to each zone for various attack models and the deflection goal. First, note that for all scenarios the

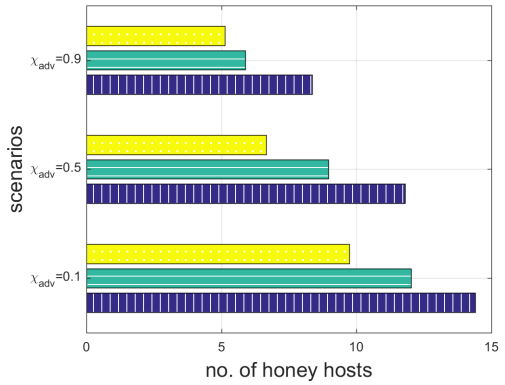

Figure 43: No. of decoy hosts in various zones for deflection goal

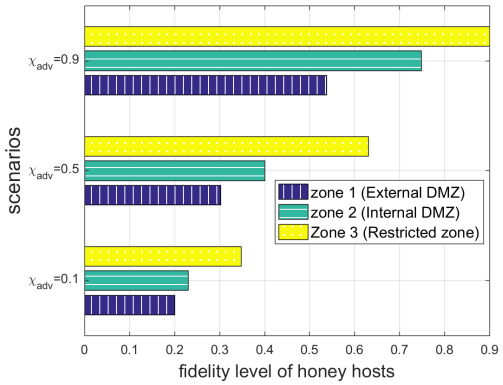

Figure 44: Fidelity values of decoy hosts in various zones for deflection goal

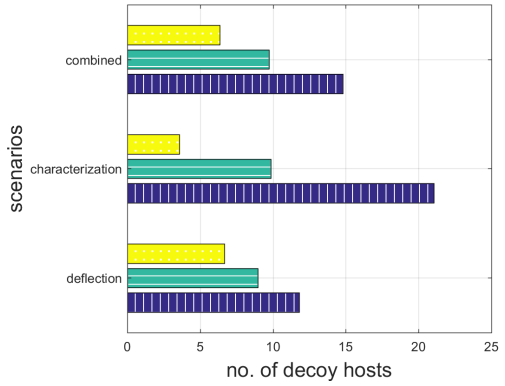

Figure 45: No. of decoy hosts for various goal types ($\chi_a dv = 0.5$)

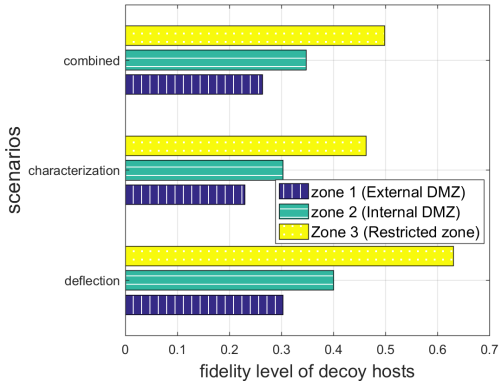

Figure 46: fidelity values for various goal types ($\chi_{adv} = 0.5$)

number of decoy hosts assigned to zone 1 is more than those assigned to zone 2, and both more than that of zone 3. This means that, according to the deception plan, maximal deflection from critical hosts that are in zone 3 is achieved when most decoy hosts are located in the zones that have a higher likelihood of being reached by the attackers.

To understand why this planning is intuitively true, note that in the deception model, the location of a decoy host is determined based on the fact that placing a decoy host in zone k increases the certainty of deflecting an attacker from real hosts

in that zone and since they have higher reachability than hosts in zone $k+1$, the impact of deflection in this zone, as defined in Eq. 74, is higher.

Second, note that as the adversary becomes more advanced, the number of decoy hosts allocated to each zone is decreased, but the fidelity values of decoy hosts increases. This is because, as the results from the previous section also showed, defeating more advanced attackers requires decoy services that are more believable and less vulnerable. This is because an advanced attacker would easily identify a service that has low fidelity. However, high-fidelity decoy services are more expensive, because they have higher computational and configuration overheads.

Figure 45 compares the deception plans for three different goals of deflection, characterization, and a combined goal that is the summation of the benefit of both goals. First, note that for the characterization goal the number of decoy hosts again decreases as we move toward more critical services. This is because it is more likely for an attacker to engage with a decoy host at a zone 1 than zone 2 and so forth. This is also true for the combined goal. Therefore, all three different goals result in deception plans that assign more decoy services to more reachable zones of the enterprise.

Second, note that the characterization goal includes more decoy hosts than both other goals. Moreover, the combined goal plan includes more decoy hosts than the deception plan for the deflection goal. However, as shown in Figure 46, the characterization plan includes decoy services with lower fidelity values. This is because, as we also noted in the results of the previous section, the characterization goal prefers decoy services that are more vulnerable to maximize the potential that an attacker compromises them. In contrast, the deflection goal prefers decoy services that have

higher fidelity, to decrease their likelihood of being compromised and therefore the effort that attacker needs to compromise them. Finally, note that as in the previous section, the combined goal is the middle-ground between deflection and characterization goals.

The deception plans depend on the distribution of critical services in the zones. In previous figures, we assumed that the most critical services are located in the innermost zone, which is zone 3. However, in order to analyze the flexibility of the model to tailor the deception plan to the mission, we compare this with two other scenarios: (1) critical services are dispersed uniformly across all zones (uniform scenario), and (2) most critical services are located in zone 1, and zone 2 includes more critical services than zone 3 (reverse scenario).

Figure 47 shows how the deception plan changes according to these different scenarios. These plans are generated for the combined goal and by assuming equal probability distribution for both attack types.

Note that for the uniform scenario, a higher number of decoy hosts are located in zone 1 as compared to the original scenario (noted as zone 3 in the figure). Also, as shown in Figure 48, the fidelity values of decoy hosts are almost equal for all the zones.

In comparison, the deception plan for the reverse scenario (noted as zone 1) almost assigned all the budget to zone 1, by selecting high-fidelity decoy services and assigning almost all of them to zone 1. This result is intuitive because most critical services are in zone 1, and therefore the highest impact is given to the goals of zone 1. As the result shows, the deception plan is constructed and tailored to the attack model,

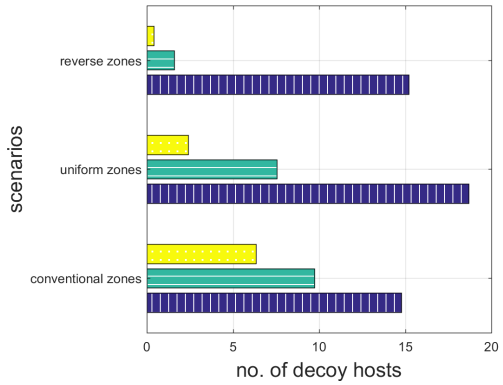

Figure 47: no. of decoy hosts for various network types

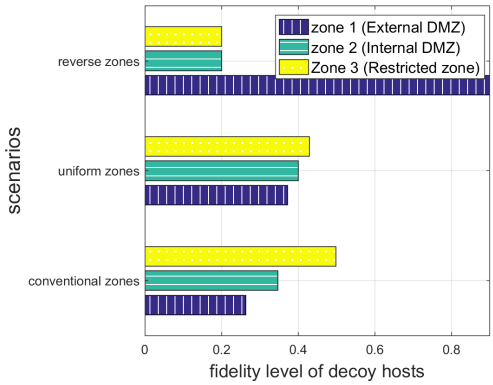

Figure 48: fidelity values for various network types

mission and risk assessments, budget, zoning hierarchy and access control rules of the enterprise.

CHAPTER 5: CONCLUSION AND FUTURE WORK

In this thesis, we introduce three approaches for proactive defense against advanced and novel attacks on networks and enterprises. These approaches provide agility into the system to counter such attacks in their network reconnaissance stage. Contrary to traditional reactive techniques, these approaches can address novel and stealthy cyber attacks.

5.1 Overview of Contributions, Technical Approaches, and Evaluation Results

5.1.1 A multi-dimensional, multi-parameter, and multi-strategy host identity anonymization

In this chapter, we introduced a multi-dimensional, multi-strategy, and multi-parameter host identity hiding technique, called M-RHM, that provides a synergistic composition of several mutation techniques to achieve maximum deterrence against even most advanced reconnaissance launched by stealthy and advanced attackers. To this aim, we provide a comprehensive review and comparison of all major works in this area. We also introduced our preliminary works on host identity hiding, including RHM [56] and A-RHM [59].

Building on these preliminary works, we introduced M-RHM as a cyber deterrence technique that defeats advanced cyber attacks in their reconnaissance stage of the

kill chain. Contrary to previous approaches, M-RHM mutates multiple parameters of networks hosts, rather than only mutating their IP addresses. This is because we show that defeating adversarial reconnaissance can be restated as the problem of anonymizing the identity of a host over time and location; therefore, defeating such reconnaissance is achievable by anonymizing every parameter of a host that could serve as an identifier or quasi-identifier for that host. We show that to achieve full anonymization; we need to anonymize host IP addresses and fingerprints to defeat advanced network reconnaissance. To anonymize fingerprints, M-RHM uses shadow decoys and the concept of k-anonymity from data privacy. We also mutate hosts' MAC addresses to defeat layer-2 reconnaissance, and name mutation to defeat reverse-DNS attacks.

M-RHM also mutates host addresses using a synergistic composition of mutation strategies: *proactive mutation* to defeat unknown attacks, *adaptive mutation* to adapt IP selection and mutation rate to attackers' scanning, and *reactive mutation* to deter internal scans by redirecting them to decoys. This synergistic composition of these strategies enables us to maximize deterrence against both external and internal reconnaissance while avoiding unnecessary mutation and overhead.

M-RHM also mutates host addresses over spatial dimension, in addition to the temporal dimension. In spatial dimension, every internal host has its own unique set of IP addresses to reach other internal hosts; therefore lateral movements and any other reconnaissance sharing between internal hosts is disabled by spatial mutation.

We show that a combination of these techniques would maximally deter adversarial reconnaissance for multi-stage advanced intrusion attacks, in addition to previous

threat models, including external scanners and network worms.

In summary, M-RHM outperforms previous cyber deterrence approaches for several reasons. First, contrary to previous approaches which achieved limited effectiveness against external scanners, M-RHM can almost completely thwart these scanners. For example, while RHM only saves up to $e^{-1} = 37\%$ of network hosts from the most advanced class of scanning called cooperative scanning, M-RHM saves up to 99% of network hosts via adaptive mutation. M-RHM also can also thwart localized scanning approaches such as local-preference and sequential scanning significantly.

More importantly, M-RHM prevents internal propagation of network worms in the private network, because the reactive mutation vector of M-RHM completely blocks internal scanners. This means that network worms that rely on scanning are easily detected and quarantined. This disables propagation of network worms beyond the public hosts that are accessible from the Internet. In addition to scanners, M-RHM achieves high resistance against advanced reconnaissance. In fact, by combining data privacy, deception, and randomization, M-RHM provides high robustness against potential de-anonymization techniques that a skilled attacker may use, especially using fingerprints as quasi-identifiers. This robust anonymization makes M-RHM effective against advanced and persistent multi-stage intrusion attacks. M-RHM deters progression of multi-stage attacks, by making re-identification of an already-probed host difficult. This means that while the attacker collects information, her knowledge does not increase over time; i.e., an attacker may unintentionally probe a host multiple times, since she is not able to identify it, not from its address nor fingerprint.

5.1.2 A Formal Framework for Active Cyber Deception Planning

In the second chapter, we propose a solution for crafting high-agility, dynamic and adaptive cyber deception plans. We introduce a deception planning framework for modeling and identifying optimal deception plans for cyber deception problems. The necessity for introducing such framework emanates from the observation that while individual deceptive actions (like placing a single decoy host in address space) is effective to some extent, unleashing the full potentials of cyber deception requires a synergistic combination of various deceptive actions, and understanding the collective effect of a various combination of these actions on an attacker's thinking process. The framework provides a language for modeling an attacker's thinking process and the effect of a group of deceptive actions (called deception plan) on it, in addition to a quantification of the deception cost for various plans. The framework introduces a *deception logic* for defining deception models, where each deception model addresses a specific deception problem in the cyber domain. This deception modeling logic is an abstraction over satisfiability modulo theories (SMT) combined with Gödel many-valued logic, G_k. The logic can capture both qualitative (e.g., belief, cause-effect, and intention) and quantitative (e.g., certainties, benefit, and cost) traits of deception and allows for reasoning on alternative deception plans.

5.1.3 Deception Planning against Multi-stage APT attacks

In the third work, we use the deception planning framework to devise a deception plan that consists of a variety of decoy services to defeat multi-stage intrusion attacks by disrupting their reconnaissance. Specifically, our goal is to determine a plan that

identifies location, type, and configuration of a number of decoy services in a coordinated manner, to defend a given enterprise network against an APT attack whose objective is to intrude the network to reach the critical services. The objective is to complicate an attacker's path toward critical network services, by devising an optimal deception plan that determines what decoy services with what configuration and in which locations would maximally deceive such APT attacks. By luring attackers to traverse fake paths, we deflect their reconnaissance to waste their resources and time. Moreover, by trapping attackers in decoys, we can characterize their motives and strategies. However, instead of determining location or configuration of these decoy services randomly or individually, we determine them by defining the problem as a deception model in the framework, to orchestrate a setup that is tailored to the mission as well as properties of the given network. We show that the generated deception plan outperforms competing alternative plans. We evaluate deception plans for single-stage and multi-stage attacks and show how different factors including attack model, distribution of critical hosts, and budget would result in different deception plans.

5.2 Future Research

Evaluating agility through red-teaming experiments. For both cyber deterrence and deception techniques, we plan to extend our evaluation to quantify the effectiveness of these techniques against real red-hat or white-hat human attackers; a technique that is known as *red-teaming* [27]. The goal is to evaluate if the presented techniques are effective and sufficient to deter and deceive advanced and adaptive human attackers.

The main theoretical challenge is the design and development of an isolated and flexible evaluation testbed.

Automated generation of deception models. We will propose techniques for automated or semi-automated generation of the deception models. This is done through systematic investigation of existing body of knowledge on attackers' potential tactics and techniques (kill-chain [53]), ATT&CK matrix [85], STIX [29]); systemic analysis of these data structures would provide us with a systematic approach for identifying necessary or potential deceptions that could be used to defeat individual attack techniques. More specifically, using the same methodology for automated generation of attack graphs [112], we intended to propose a methodology and develop tools for the automated creation of deception models against various classes of advanced and persistent threats. This automated tool would extend a network's attack graph with potential fake paths to which attackers could be misled, and quantifies the effect of such deflections on deterring the attacker's progress in the network.

Modeling real-time adaptive planning in deception framework. The deception planning framework includes paradigms for generating an adaptive deception plan based on new observations about attackers or the system. In future, we intend to enable this planning to adapt in real-time to the changes in the system. The objective would be to devise a deception plan that is adapted dynamically and deployed quickly. This adaptation considers an attacker's partially-observable actions in a real-time composition of the plan while avoiding the attacker's potential counter-deceptions. The real challenge is the timeliness of this planning, as well as susceptibility to manipulation by stealthy and evasive attackers. To this aim, the framework will be extended to

include generic interfaces for real-time deployment of an updated deception plan in the system, and also interfaces for real-time information collection and conversion to input parameters that will be given to the deception model.

Constructing a deception kill chain for cyber attacks. The deception planning framework targets the reconnaissance stage of cyber attacks, through providing misleading information. However, to maximally engage an attacker with a deception plan, the deception plan must provide a deceptive step corresponding to other stages of the kill chain; i.e., weaponization, delivery, exploitation, command and control, and actions on an objective such as lateral movements. In the future work, we investigate how the deception planning framework could be extended to provide engaging and believable deception that misleads an attacker into pursuing all stages of her intrusion kill-chain. To this aim, the framework must be extended to include components for modeling attackers' potential courses of actions at each stage, and also mechanisms for validating that the attacker is conforming to our expectations and taking the desired course of action. It must also include mechanisms and paradigms for generating and validating a hypothesis about an attackers' motif and dynamically adapting the deception environment to them. The resulting deceptive kill-chain is referred to as *deception chain* [50] and will define a high-level discrete description of active steps for successful and maximal engagement of a cyber attacker with a given cyber system.

REFERENCES

[1] E. Al-Shaer, Q. Duan, and J. H. Jafarian. Random host mutation for moving target defense. In *SecureComm*, volume 106, pages 310–327. Springer, 2012.

[2] M. Albanese, A. De Benedictis, S. Jajodia, and K. Sun. A moving target defense mechanism for manets based on identity virtualization. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 278–286, Oct 2013.

[3] Y. Alosefer and O. Rana. Honeyware: a web-based low interaction client honeypot. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 410–417. IEEE, 2010.

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

[5] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. P. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Usenix Security*, 2005.

[6] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Comput. Netw.*, 51(12):3471–3490, 2007.

[7] S. Antonatos, K. Anagnostakis, and E. Markatos. Honey@ home: a new approach to large-scale threat monitoring. In *Proceedings of the 2007 ACM workshop on recurring malcode*, pages 38–45. ACM, 2007.

[8] Apache. logresolve - resolve ip-addresses to hostnames in apache log files. `https://httpd.apache.org/docs/2.4/programs/logresolve.html`, 2017.

[9] M. Atighetchi, P. Pal, F. Webber, and C. Jones. Adaptive use of network-centric mechanisms in cyber-defense. In *ISORC '03*, page 183. IEEE Computer Society, 2003.

[10] M. Baaz, N. Preining, and R. Zach. First-order gödel logics. *Annals of Pure and Applied Logic*, 147(1-2):23–47, 2007.

[11] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

[12] N. Bjørner and L. de Moura. $z3^{10}$: Applications, enablers, challenges and directions. In *Sixth International Workshop on Constraints in Formal Verification*, 2009.

[13] W. v. d. G. C. Contavalli. Client subnet in dns requests. `http://tools.ietf.org/html/draft-vandergaast-edns-client-subnet-00`, 2011.

[14] A. Caglayan, M. Toothaker, D. Drapaeau, D. Burke, and G. Eaton. Behavioral analysis of fast flux service networks. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 48:1–48:4. ACM, 2009.

[15] J.-Y. Cai, V. Yegneswaran, C. Alfeld, and P. Barford. An attacker-defender game for honeynets. In *Proceedings of the 15th Annual International Conference on Computing and Combinatorics*, pages 7–16. Springer-Verlag, 2009.

[16] V. Carofiglio, F. de Rosis, and C. Castelfranchi. Ascribing and weighting beliefs in deceptive information exchanges. In *User Modeling 2001*, pages 222–224. Springer, 2001.

[17] T. E. Carroll and D. Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, 4(10):1162–1172, 2011.

[18] T. L. Carson. *Lying and deception: Theory and practice*. Oxford University Press, 2010.

[19] K. M. Carter, J. F. Riordan, and H. Okhravi. A game theoretic approach to strategy determination for dynamic platform defenses. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 21–30. ACM, 2014.

[20] C. Castelfranchi. Artificial liars: Why computers will (necessarily) deceive us and each other. *Ethics and Information Technology*, 2(2):113–119, 2000.

[21] S. Chakravarty. A characterization of binary decision diagrams. *IEEE Transactions on Computers*, 42(2):129–137, 1993.

[22] Z. Chen, C. Chen, and Y. Li. Deriving a closed-form expression for worm-scanning strategies. *Int. J. Secur. Netw.*, 4(3):135–144, 2009.

[23] M. Christodorescu, M. Fredrikson, S. Jha, and J. Giffin. End-to-end software diversification of internet services. In S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, editors, *Moving Target Defense*, volume 54 of *Advances in Information Security*, pages 117–130. Springer New York, 2011.

[24] A. Ciabattoni and G. Metcalfe. Bounded łukasiewicz logics. *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 32–47, 2003.

[25] F. Cohen and D. Koike. Misleading attackers with deception. In *information assurance workshop, 2004. Proceedings from the fifth annual IEEE SMC*, pages 30–37. IEEE, 2004.

[26] F. Cohen, D. Lambert, C. Preston, N. Berry, C. Stewart, and E. Thomas. A framework for deception. *National Security Issues in Science, Law, and Technology*, 2001.

[27] F. Cohen, I. Marin, J. Sappington, C. Stewart, and E. Thomas. Red teaming experiments with deception technologies. *IA Newsletter*, 2001.

[28] O. Community. Owasp risk rating modology. `https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology`, 2016.

[29] S. Community. Structured threat information expression: a structure language for cyber threat intelligence. `https://stixproject.github.io/`, 2016.

[30] L. de Moura and N. Bjørner. Satisfiability modulo theories: An appetizer. In *SBMF '09, Brazilian Symposium on Formal Methods*, 2009.

[31] F. De Rosis, V. Carofiglio, G. Grassano, and C. Castelfranchi. Can computers deliberately deceive? a simulation tool and its application to turing's imitation game. *Computational Intelligence*, 19(3):235–263, 2003.

[32] R. Delaney. Vagrant. *Linux Journal*, 2014(244):1, 2014.

[33] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic host configuration protocol for ipv6. `https://tools.ietf.org/html/rfc3315`, 2003.

[34] Q. Duan, E. Al-Shaer, and H. Jafarian. Efficient random route mutation considering flow and network constraints. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 260–268. IEEE, 2013.

[35] P. Engebretson. *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier, 2013.

[36] D. Ettinger and P. Jehiel. A theory of deception. *American Economic Journal: Microeconomics*, 2(1):1–20, 2010.

[37] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet dossier. Technical report, Symantic Security Response, Oct. 2010.

[38] M. Fitting. Kleene's logic, generalized. *Journal of Logic and Computation*, 1(6):797–810, 1991.

[39] V. N. L. Franqueira. *Finding multi-step attacks in computer networks using heuristic search and mobile ambients*. PhD thesis, University of Twente, Enschede, 2009.

[40] N. Garg and D. Grosu. Deception in honeynets: A game-theoretic analysis. In *Information Assurance and Security Workshop, 2007. IAW'07. IEEE SMC*, pages 107–113. IEEE, 2007.

[41] A. Ghourabi, T. Abbes, and A. Bouhoula. Honeypot router for routing protocols protection. In *Risks and Security of Internet and Systems (CRiSIS), 2009 Fourth International Conference on*, pages 127–130. IEEE, 2009.

[42] F. Gilani. *Software-defined Cyber Agility for Active Cyber Defense*. PhD thesis, UNC Charlotte, 2017.

[43] K. Gödel. Russells mathematical logic. *1944*, pages 123–153, 1944.

[44] S. Golovanov and I. Soumenkov. *TDL4 : Top Bot*, 2011 (accessed november 7, 2011). `http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot`.

[45] J. Grasman. Epidemic modelling: An introduction. *American Journal of Human Biology*, 12(6):846–847, 2000.

[46] M. Gregg. *Certified Ethical Hacker Exam Prep*. Que, 2006.

[47] N. W. Group. Classless in-addr.arpa delegation. `https://tools.ietf.org/html/rfc2317`, 2017.

[48] M. M. Gupta and J. Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy sets and systems*, 40(3):431–450, 1991.

[49] P. Hájek, L. Godo, and F. Esteva. A complete many-valued logic with product-conjunction. *Archive for mathematical logic*, 35(3):191–208, 1996.

[50] K. E. Heckman, F. J. Stech, B. S. Schmoker, and R. K. Thomas. Denial and deception in cyber defense. *Computer*, 48(4):36–44, 2015.

[51] V. Heydari and S. Yoo. Moving target defense enhanced by mobile ipv6. In *7th Annual Southeastern Cyber Security Summit*, 2015.

[52] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 9:1–9:9. USENIX Association, 2008.

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

[54] A. Ikinci, T. Holz, and F. C. Freiling. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In *Sicherheit*, volume 8, pages 407–421, 2008.

[55] T. Jackson, B. Salamat, A. Homescu, K. Manivannan, G. Wagner, A. Gal, S. Brunthaler, C. Wimmer, and M. Franz. Compiler-generated software diversity. In *Moving Target Defense*, volume 54 of *Advances in Information Security*, pages 77–98. Springer New York, 2011.

[56] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[57] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Formal approach for route agility against persistent attackers. In *European Symposium on Research in Computer Security*, pages 237–254. Springer Berlin Heidelberg, 2013.

[58] J. H. Jafarian, E. Al-Shaer, and Q. Duan. Adversary-aware ip address randomization for proactive agility against sophisticated attackers. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 738–746. IEEE, 2015.

[59] J. H. Jafarian, E. Al-Shaer, and Q. Duan. An effective address mutation approach for disrupting reconnaissance attacks. *IEEE Transactions on Information Forensics and Security*, 10(12):2562–2577, 2015.

[60] J. H. Jafarian, A. Niakanlahiji, E. Al-Shaer, and Q. Duan. Multi-dimensional host identity anonymization for defeating skilled attackers. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 47–58. ACM, 2016.

[61] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan. Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 69–78. ACM, 2014.

[62] S. Jajodia, A. K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang. Moving target defense ii. *Application of game Theory and Adversarial Modeling. Series: Advances in Information Security*, 100:203, 2013.

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

[64] X. Jiang and X. Wang. out-of-the-box monitoring of vm-based high-interaction honeypots. In *Recent Advances in Intrusion Detection*, pages 198–218. Springer, 2007.

[65] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160. ACM, 2013.

[66] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *2004. Proceedings. 2004 IEEE Symposium on Security and Privacy*, pages 211 – 225, 2004.

[67] P. Kampanakis, H. Perros, and T. Beyene. Sdn-based solutions for moving target defense network protection. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

[68] M. S. Kang, S. B. Lee, and V. D. Gligor. The crossfire attack. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pages 127–141, Washington, DC, USA, 2013. IEEE Computer Society.

[69] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 272–280. ACM, 2003.

[70] D. Kewley, R. Fink, J. Lowry, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *DARPA Information Survivability Conference Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 176 –185 vol.1, 2001.

[71] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer. Dag-based attack and defense modeling: Dont miss the forest for the attack trees. *Computer science review*, 13:1–38, 2014.

[72] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail. A dynamic honeypot design for intrusion detection. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, pages 95–104. IEEE, 2004.

[73] Y.-P. Lai and P.-L. Hsia. Using the vulnerability information of computer systems to improve the network security. *Computer Communications*, 30(9):2032–2047, 2007.

[74] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 19:1–19:6. ACM, 2010.

[75] A. P. Laudicina. Nessus – a powerful, free remote security scanner. *SysAdmin*, 11(5), 2002.

[76] T. Liston. Labrea:sticky honeypot and ids. *SourceForge.net,[cit. 2012-05-26], Dostupné z:¡ http://labrea. sourceforge. net*, 2, 2009.

[77] J. Long, B. Gardner, and J. Brown. *Google hacking for penetration testers*, volume 2. Syngress, 2011.

[78] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, USA, 2009.

[79] L. Martin. Cyber kill chain. *URL: http://cyber. lockheedmartin. com/hubfs/Gaining_the_Advantage_Cyber_Kill _Chain. pdf*, 2014.

[80] D. Maynor. *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier, 2011.

[81] OpenFlow group at Stanford University. *POX Wiki*, 2014. `https://openflow.stanford.edu/display/ONL/POX+Wiki`.

[82] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, March 2008.

[83] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[84] Microsoft. *Z3: An Efficient Theorem Prover*, 2016. `http://research.microsoft.com/en-us/projects/z3m/`.

[85] MITRE. Att&ck matrix: Adversarial tactics, techniques, and procedures. `https://attack.mitre.org/`, 2016.

[86] C. Morrell, J. S. Ransbottom, R. Marchany, and J. G. Tront. Scaling IPv6 address bindings in support of a moving target defense. In *9th International Conference for Internet Technology and Secured Transactions*, pages 440–445. IEEE, 2014.

[87] L. Moura and N. Bjørner. Formal methods: Foundations and applications. chapter Satisfiability Modulo Theories: An Appetizer, pages 23–36. Springer-Verlag, 2009.

[88] C. Mulliner, S. Liebergeld, and M. Lange. Poster: Honeydroid-creating a smartphone honeypot. In *IEEE Symposium on Security and Privacy*, 2011.

[89] J. Nazario. Phoneyc: A virtual client honeypot. *LEET*, 9:911–919, 2009.

[90] V. Neagoe and M. Bishop. Inconsistency in deception for defense. In *Proceedings of the 2006 workshop on New security paradigms*, pages 31–38. ACM, 2006.

[91] S. Oliver. Mac address randomization joins apple's heap of ios 8 privacy improvements, june 2014.

[92] G. Portokalidis and A. D. Keromytis. Global ISR: Toward a comprehensive defense against unauthorized code execution. In *Moving Target Defense*, volume 54 of *Advances in Information Security*, pages 49–76. Springer New York, 2011.

[93] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 15–27. ACM, 2006.

[94] G. Priest. The logic of paradox. *Journal of Philosophical logic*, 8(1):219–241, 1979.

[95] N. Provos. Honeyd-a virtual honeypot daemon. In *10th DFN-CERT Workshop, Hamburg, Germany*, volume 2, page 4, 2003.

[96] N. Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[97] N. Rescher. *Many Valued Logic (Modern Revivals in Philosophy)*. Gregg Revivals, 1993.

[98] M. Research. Getting started with z3: A guide. `https://rise4fun.com/Z3/tutorial`, 2017.

[99] S. Research. A manifesto for cyber resilience. `https://www.symantec.com/content/en/us/enterprise/other_resources/b-a-manifesto-for-cyber-resilience.pdf`, 2017.

[100] J. Riden. *Know Your Enemy: Fast-Flux Service Networks*, 2017 (accessed September 28, 2017). `http://www.honeynet.org/papers/ff`.

[101] N. Rowe. Planning cost-effective deceptive resource denial in defense to cyber-attacks. In *Proceedings of the 2nd International Conference on Information Warfare & Security*, page 177, 2007.

[102] N. C. Rowe. Counterplanning deceptions to foil cyber-attack plans. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 203–210. IEEE, 2003.

[103] N. C. Rowe. Designing good deceptions in defense of information systems. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 418–427. IEEE, 2004.

[104] N. C. Rowe. Finding logically consistent resource-deception plans for defense in cyberspace. In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, volume 1, pages 563–568. IEEE, 2007.

[105] N. C. Rowe, B. T. Duong, and E. Custy. Fake honeypots: a defensive tactic for cyberspace. *draft, Computer Science Department, Naval Postgraduate School*, 2006.

[106] N. C. Rowe, B. T. Duong, and E. J. Custy. Defending cyberspace with fake honeypots. 2007.

[107] C. Sakama and M. Caminada. The many faces of deception. *Proceedings of the Thirty Years of Nonmonotonic Reasoning*, 2010.

[108] C. Sakama, M. Caminada, and A. Herzig. A formal account of dishonesty. *Logic Journal of IGPL*, page jzu043, 2014.

[109] C. Seifert and R. Steenson. Capture-honeypot client (capture-hpc). *pp. Available at https://projects. honeynet. org/capture–hpc*, 2006.

[110] C. Seifert, I. Welch, P. Komisarczuk, et al. Honeyc-the low-interaction client honeypot. *Proceedings of the 2007 NZCSRCS, Waikato University, Hamilton, New Zealand*, pages 1–8, 2007.

[111] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307. ACM, 2004.

[112] O. Sheyner and J. Wing. Tools for generating and analyzing attack graphs. In *International Symposium on Formal Methods for Components and Objects*, pages 344–371. Springer, 2003.

[113] R. Siles. Honeyspot: The wireless honeypot. *The Spanish Honeynet Project (SHP)(December 2007)*, 2007.

[114] T. Sochor and M. Zuzcak. Study of internet threats and attack methods using honeypots and honeynets. In *Computer Networks*, pages 118–127. Springer, 2014.

[115] V. E. Solutions. 2015 data breach investigation report. `http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigation-report_2015_en_xg.pdf`, 2017.

[116] L. Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 99(2):15–23, 2003.

[117] L. Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[118] L. Spitzner. Honeytokens: The other honeypot. 2003. *Internet: http://www.securityfocus.com/infocus/1713*, 2006.

[119] S. Stafford and J. Li. Behavior-based worm detectors compared. In *RAID*, volume 6307 of *Lecture Notes in Computer Science*, pages 38–57. Springer, 2010.

[120] C. Stoll. The cuckoos egg: Tracing a spy through the maze of computer espionage, 1989.

[121] A. Studer and A. Perrig. The coremelt attack. In *European Symposium on Research in Computer Security*, pages 37–52. Springer, 2009.

[122] S. Suneja, C. Isci, V. Bala, E. de Lara, and T. Mummert. Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 249–261. ACM, 2014.

[123] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[124] S. Tzu. *The art of war*. e-artnow, 500BC.

[125] N. Virvilis, B. Vanautgaerden, and O. S. Serrano. Changing the game: The art of deceiving sophisticated attackers. In *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*, pages 87–97. IEEE, 2014.

[126] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 148–162. ACM, 2005.

[127] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 35–49, 2006.

[128] J. White. Creating personally identifiable honeytokens. In *Innovations and Advances in Computer Sciences and Engineering*, pages 227–232. Springer, 2010.

[129] J. Xu, P. Guo, M. Zhao, R. F. Erbacher, M. Zhu, and P. Liu. Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM, 2014.

[130] J. Yackoski, P. Xie, H. Bullen, J. Li, and K. Sun. A self-shielding dynamic network architecture. In *Military Communication Conference, 2011 - MILCOM 2011*, pages 1381 –1386, nov. 2011.

[131] V. Yegneswaran and C. Alfeld. Camouflaging honeynets. In *In Proceedings of IEEE Global Internet Symposium*, 2007.

[132] J. Yuill, M. Zappe, D. Denning, and F. Feer. Honeyfiles: deceptive files for intrusion detection. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 116–122. IEEE, 2004.

[133] J. Zhuang, V. M. Bier, and O. Alagoz. Modeling secrecy and deception in a multiple-period attacker–defender signaling game. *European Journal of Operational Research*, 203(2):409–418, 2010.

[134] R. Zhuang, S. A. DeLoach, and X. Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[135] C. C. Zou, D. Towsley, and W. Gong. On the performance of internet worm scanning strategies. *Elsevier Journal of Performance Evaluation*, 63:700–723, 2003.