A THEORY FOR UNDERSTANDING AND QUANTIFYING MOVING TARGET DEFENSE

by

RUI ZHUANG

B.E.(Double), HuaZhong University of Science and Technology, China, 2006

M.S., HuaZhong University of Science and Technology, China, 2009

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Abstract

The static nature of cyber systems gives attackers a valuable and asymmetric advantage - time. To eliminate this asymmetric advantage, a new approach, called Moving Target Defense (MTD) has emerged as a potential solution. MTD system seeks to proactively change system configurations to invalidate the knowledge learned by the attacker and force them to spend more effort locating and re-locating vulnerabilities. While it sounds promising, the approach is so new that there is no standard definition of what an MTD is, what is meant by diversification and randomization, or what metrics to define the effectiveness of such systems. Moreover, the changing nature of MTD violates two basic assumptions about the conventional attack surface notion. One is that the attack surface remains unchanged during an attack and the second is that it is always reachable. Therefore, a new attack surface definition is needed.

To address these issues, I propose that a theoretical framework for MTD be defined. The framework should clarify the most basic questions such as what an MTD system is and its properties such as adaptation, diversification and randomization. The framework should reveal what is meant by gaining and losing knowledge, and what are different attack types. To reason over the interactions between attacker and MTD system, the framework should define key concepts such as attack surface, adaptation surface and engagement surface. Based on that, this framework should allow MTD system designers to decide how to use existing configuration choices and functionality diversification to increase security. It should allow them to analyze the effectiveness of adapting various combinations of different configuration aspects to thwart different types of attacks. To support analysis, the framework should include an analytical model that can be used by designers to determine how different parameter settings will impact system security.

A THEORY FOR UNDERSTANDING AND QUANTIFYING MOVING TARGET DEFENSE

by

RUI ZHUANG

B.E.(Double), HuaZhong University of Science and Technology, 2006

M.S., HuaZhong University of Science and Technology, 2009

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Approved by:

Major Professor
Dr. Scott A. DeLoach

Copyright

Abstract

The static nature of cyber systems gives attackers a valuable and asymmetric advantage - time. To eliminate this asymmetric advantage, a new approach, called Moving Target Defense (MTD) has emerged as a potential solution. MTD system seeks to proactively change system configurations to invalidate the knowledge learned by the attacker and force them to spend more effort locating and re-locating vulnerabilities. While it sounds promising, the approach is so new that there is no standard definition of what an MTD is, what is meant by diversification and randomization, or what metrics to define the effectiveness of such systems. Moreover, the changing nature of MTD violates two basic assumptions about the conventional attack surface notion. One is that the attack surface remains unchanged during an attack and the second is that it is always reachable. Therefore, a new attack surface definition is needed.

To address these issues, I propose that a theoretical framework for MTD be defined. The framework should clarify the most basic questions such as what an MTD system is and its properties such as adaptation, diversification and randomization. The framework should reveal what is meant by gaining and losing knowledge, and what are different attack types. To reason over the interactions between attacker and MTD system, the framework should define key concepts such as attack surface, adaptation surface and engagement surface. Based on that, this framework should allow MTD system designers to decide how to use existing configuration choices and functionality diversification to increase security. It should allow them to analyze the effectiveness of adapting various combinations of different configuration aspects to thwart different types of attacks. To support analysis, the framework should include an analytical model that can be used by designers to determine how different parameter settings will impact system security.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Dr.Scott DeLoach, who always provides me the guidance and advice right to the point. I'm grateful for the research opportunities to explore the beloved topics and I greatly appreciate the time Dr.DeLoach spent to help revise and improve this dissertation. I also would like to thank my Committee members, Dr.Xinming(Simon) Ou, Dr.William Hsu, Dr.Steve Warren and Dr.Kevin Lease for their suggestions and input during this dissertation work and especially thank for the support and help from Dr.Simon during my PhD study. In addition, I appreciate all other factulty members who has instructed me during this whole process.

I'm grateful to the help and knowledge learned from my colleagues and greatly appreciate the encouragement and support from all my friends during this fruitful journey.

This end is another beginning...

Dedication

To my parents, my wife and our newborn baby.

Chapter 1

Introduction

1.1 Motivation

Cyber security is currently implemented in an ad hoc and inconsistant fashion due to the priority of business concerns over system security. Technologies used in enterprise networks are usually implemented to satisfy the business needs such as communication, data processing and customer support while leaving cyber security as a second priority. Enterprise network administrators are left to patch potential vulnerabilities, scan the system to remove potential intrusions, maintain access lists to add or remove users, and modify firewall rules to limit communications between the internet and internal hosts. Due to the complexity and heavy workload of modifying a system, once it is deployed, the configuration could remain unchanged for a long period of time. This static nature of current enterprise networks gives attackers an asymmetric advantage – time. Attackers can spend as much *time* as necessary to perform reconnaissance of the target network, study and determine its potential vulnerabilities, choose the best *time* to launch the attack and can even maintain a backdoor without being discovered for a long period of *time*.

Recently a new approach called *Moving Target Defense* (MTD) was proposed to *eliminate* the attacker's asymmetric advantage[1,2]. A high-level intuition that demonstrates the

key idea behind MTD is shown in Figure 1.1. The notion of *attack surface* was introduced by Manadhata *et,al*[3,4] to indicate the vulnerable components in a computer system that could be exploited. Since attackers need to perform exploration or reconnaissance to investigate the configuration of the target system before launching an actual attack, we introduce a concept called *exploration space* to represent this space. In this thesis, the *attack surface* will represent a specific configuration in which a vulnerability exists. For example, if we consider *exploration space* from an IP address perspective, then for a machine running in a typical C class subnet (say 192.168.0.x) whose IP address can be changed during runtime, the *exploration space* is a set $\{192.168.0.1, 192.168.0.2, \ldots 192.168.0.254\}$ and the size of this space is 254. However, since a machine that runs a vulnerable service has only one active IP address when running, the *attack surface* size is 1. Formal definitions of *attack surface* and *exploration space* will be given in Chapter 6.

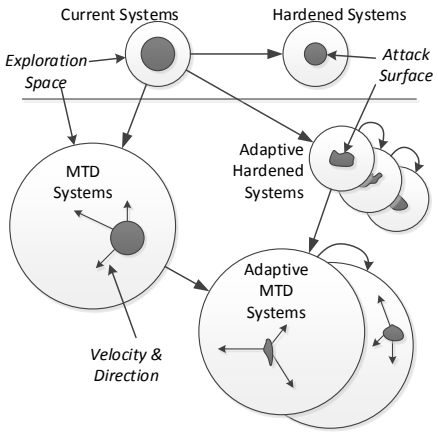


Figure 1.1: MTD High Level Intuition

For most critical networks and systems, the typical way to increase system security is to reduce the attack surface, or in other words, harden the system. To reduce the *attack surface*, administrators usually patch and update operating systems, remove unneccessary software, delete obsolete user credentials, close unused service ports, set up firewalls, *etc*[5,6]. Nevertheless, such approaches are caused by a legacy information system design where

ease of use, maintainence and business needs take precedence over security. Moreover, for complex systems such approaches could quickly lead to convoluted firewall rules, inadequate authentication mechanisms and fragmented policies due to complex software configuration, and significant access control and credential maintenance efforts. These traits basically guarantee a relatively long period of static configuration, thus leaving the attacker a large time window in which to penetrate the system.

More advanced adaptive hardening approaches capture input from intrusion detection systems (IDS) and reactively launch automated responses to patch or block services to thwart ongoing attacks[7,8,9,10,11,12,13,14,15,16,17]. These automated responses change the *attack surface* at runtime, which eases the maintenance overhead of administrator. However, such approaches require a significant effort to develop and maintain large numbers of signatures for identifing intrusions and malware and have the potential for triggering a large number of false-positive alarms, which could be harmful for interrupting normal operations. Also, such reactive defenses are ineffective against new and zero-day attacks.

Instead of focusing on reducing the attack surface, MTD seeks to enlarge the exploration space during the design phase and shifting the attack surface to force re-exploration during the runtime phase. Intuitively, by increasing the exploration space and shifting the attack surface, an attacker needs to spend more effort locating and re-locating vulnerabilities. Previous research, such as network[18,19,20,21] and memory address space randomization[22,23,24], instruction set randomization[25,26], host IP mutation[27,28], and software diversification[29,30,31] tried to increase the difficulty of discovering the target systems' configuration by enlarging the exploration space or proactively shifting the attack surface. Moreover, by taking advantage of adaptive hardening approaches, more advanced moving target defense systems can incorporate feedback from an IDS to proactively launch automated responses to change or shift the attack surface during runtime, thus increasing penetration difficulty even further[32,33,34].

Although there are many ongoing research efforts, moving target defense is still in its

infancy. Most of the previous work focuses on some specific aspect of system configuration, such as IP address[18,19,20,21], memory layout[22,23,24], instruction set[25,26], html keyword[35,29], SQL query[36], database table keywords[29], etc. While recently a few comprehensive frameworks[37,38,39,40,41,42,43] have been proposed, most of these frameworks are still at the conceptual level and significant effort is required to bring them to fruition, from both theoretical and practical standpoints.

A major challenge is the need for new metrics. As indicated by Huang[31], existing metrics[3,4] for attack surface areas are not suitable for evaluating a moving *attack surface* because two basic assumptions of the existing metrics have been broken. One assumption is that the attack surface remains unchanged, while the other is that the target attack surface is always reachable by attackers. Thus, new metrics are required to take into account MTD's changing and unpredictable nature. Manadhata[44] extended his original attack surface definitions[3] to include definitions for a shifting attack surface. This extension allows modeling the interaction between the defender and the attacker as a two player game, using game theory to determine optimal defense strategies. However, as admitted by Manadhata, the potential state and action space explosion are serious problems. The paper also leaves the instantiation of the model in software as future work. Kant[45] points out that the changing of the attack surface can be done by exploiting MTD or more generally by introducing diversity. However, quantitative models for guiding the design of good diversification techniques and assessing their effectiveness remain largely unexplored. Christodorescu *et,al*[29] also indicate that a fundamental challenge in understanding the impact of diversification is to introduce a precise, computationally-meaningful way to measure the increase in difficulty for the attacker.

1.2 Thesis Statement

A theoretical framework for moving target defense systems will provide insights into the key design choices for an MTD system as well as enable objective measurement and analysis of the effectiveness of different movement mechanisms against different types of attack.

The theoretical framework should clarify the most basic questions such as "What is a moving target defense system?", "What can be moved?", "How and when?".

The framework should also provide support for making key design decisions such as:

- What features should be implemented to reduce the attacker's intrusion success likelihood?
- How the diversification of functionality will impact security?
- What adaptation interval is required to maintain security at certain level with acceptable costs?
- How much can security be increased by implementing a specific MTD defense when compared to static systems?

To answer these questions, the theory framework should provide a set of fundamental definitions to support the measurement and analysis involved in a MTD system.

A common paraphrase of Lord Kelvin (a.k.a William Thomson) is "If you can not measure it, you can not improve it". However, measuring security is hard, if not impossible[46]. The key reason behind this is that the attacker's effort is often linear in terms of security layers. The effort required to break each layer is low. Once a layer is penetrated, the next layer is exposed and the the attacker has almost unlimited time to attack it. Thus, the total attacker effort is a summation of the effort required for each layer. Bellovin[46] claims that what is really needed for defense systems is an exponential property where the intrusion effort is proportional to the *product* of each defense layer's difficulty. I propose that MTD provides a way to achieve this exponential property thus providing a way to measure

security. By constantly adapting configuration aspects, such as IP addresses, port numbers, instruction sets, OS types, program keywords, etc, MTD systems can overcome the brittleness of standard defenses since, once a layer is penetrated, the attackers have limited time to reach their objective before an adaptation forces the attacker to re-penetrate previously compromised layers.

The framework should help designers decide how to use existing configuration aspects and functionality diversification to provide the defense strength for each layer. To support measuring such strength, the framework should provide foundamental definitions such as *attack surface* and *exploration space* that are appropriate for the dynamic nature of MTD systems without making limiting assumptions.

The framework should also be general enough to be used by designers to analyse the effectiveness of adaptating various combinations of configuration aspects to thwart different types of attacks. To support analysis, the framework should provide an analytical model that can be used by designers to easily determine how different parameters and settings will impact the security provided by an MTD system.

1.3 Research Approach

To address these issues, I propose to define a taxonomy based on a survey of MTD research and provide a basic MTD design that will separate *what* the system should do from *how* the system will implement it. This taxonomy and design will show that the MTD system could use diversification at the Service layer (running software configurations), OS layer (operating system settings such as type, version, memory size, etc), and Network layer (such as IP address, port number) within a random adaptation process that proactively changes the mapping from the required functionality (*the what*) to the system implementation (*the how*). This framework will support integrating existing low-level MTD techniques to thwart a broad range of attack types.

Based on this taxonomy and design, I present a concrete simulated MTD system and how to model an automated pivoting attack component to penetrate this MTD system. Simulation results show that increased adaptation speed reduces the number of successful attacks, and an intrusion detection enabled MTD system could reduce the number of successful attacks even further.

Next, I extend the simulator and present an efficient and scalable analytical model that reveals and captures the relationships between the key elements involved in a MTD system. The model will provide insight to MTD designers so that better decisions can be made when employing an MTD system. Simulation results show the analytical model can be effectively used to estimate the success likelihood of intrusion and significantly simplify the measurement of the increase in system security.

Based on the insights gained from the survey, simulations and analytical model, I propose a complete MTD theory that can be used to understand key parameters and interactions between attacker and MTD system, evaluate the potential effectiveness of an MTD system from attack surface, adaptation surface, engagement surface, and quantifing the effectiveness in terms of success likelihood of intrusion.

This thesis uses simulated multi-hop remote exploit attacks (or pivoting attacks) in an enterprise network context as a specific example to show how this framework can be used to guide the MTD design and implementation as well as to analyze how different MTD parameter settings can impact security. To validate the MTD theory, this thesis uses the *return-to-libc* attack, which is a concrete remote exploit attack. However, the framework is general and similar methods can be used to determine what features to implement to defend against other types of attack.

1.4 Contributions

In this moving target defense research, I make the following contributions:

1. A taxonomy that categorizes current moving target defense techniques to help better understand the state-of-the-art MTD research. The taxonomy categorizes related work by Adaptable Aspects, Tactics, Strategies, and the Attack Type it is intended to thwart. It also helps to point out the potential MTD research areas.

2. A theoretical framework for moving target defense that supports understanding and quantifing the effectiveness of such defenses.

    - A MTD system theory that defines the key concept of a moving target defense system.
    - A cyber attack theory that defines key concepts of attack such as attacker knowledge, attack type, attack instances and exploration space.
    - A MTD theory that defines key concepts to capture the interactions between attacker and MTD system, such as attack surface, adaptation surface, engagement surface, coverage and success likelihood of intrusion.

3. An evaluation of the framework and analytic model in a simulated MTD testbed.

    - A MTD defense component that performs random adaptations as well as event triggered adaptations.
    - An offensive component that features modeling the automated pivoting attacks to emulate multi-hop remote exploit attacks against simulated enterprise networks.
    - A probabilistic analytical model that explicitly illustrates how to measure MTD system security in terms of intrusion success likelihood to individual targets.
    - A set of experimental simulations to validate the proposed analytical model and typical examples to demonstrate that the theoretical framework can guide the design and development of an MTD system that protects enterprise network systems better than static defenses.

1.5 Overview

This thesis is organized as follows. Chapter 2 describes the related mathematical, information theoretic and cloud computing background necessary to understand the rest of the thesis. Chapter 3 presents a taxonomy for categorizing the state-of-the-art MTD research. Chapter 4 discusses the work on a high-level MTD architecture design, a simulator based on this design[32,33,34], and initial experiments to study the effectiveness of MTD. Chapter 5 extends the simulator presented in Chapter 4 to investigate a scalable analytical model that can be used to analyze multi-hop attacks to a group of hosts. Simulation results to evaluate the accuracy of this model will also be presented. Chapter 6 proposes a theoretical framework that comprises MTD System Theory, Cyber Attack Theory and MTD Theory in order to understand the interactions between attackers and MTD systems, evaluating the potential effectiveness from attack surface, adatation surface, engagement surface and coverage, and quantifing the effectiveness of MTD in terms of success likelihood of intrusion. Chapter 7 ends with a conclusion and potential areas of future work.

Chapter 2

Background

This chapter presents background information that would be beneficial for understanding the following chapters' contents. Related mathematics, such as geometric series, product rule of probability and conditional independence are introduced first, then related concepts, such as enterprise network, cloud computing and IT automation are presented. Based on the discussion of state-of-the-art IT automation technology, we will gain a high level understanding of how enterprise networks that are comprised of a cluster of physical machines can be configured and deployed remotely and scaled easily. We will also see how it enables a cloud to become an example of such enterprise networks comprised of clustered virtual machines instead of bare metal hosts.

2.1 Related Mathematics

As a reference, this section shows related mathematic concepts, definitions and theorems. As most of the involved theorems are common and well known, they are provided directly without proofs. Interested readers who want to know more about the proofs of these theorems may find these resources helpful[47,48].

2.1.1 Geometric Series

In mathematics, geometric series often refers to a series with constant ratio of successive terms.

Definition 2.1. *A geometric series is a series whose ratio of successive terms is constant. Let a represent the first term of the series and q represent the common ratio, then the geometric series can be represented as:*

$$a + aq + aq^2 + aq^3 + \ldots = \sum_{k=0}^{\infty} aq^k$$

A geometric series with infinite elements could be converge or diverge. The condition for a geometric series to converge and the final converged value stated in this theorem will be used in Chapter 5 to derive a scalable analytical model of MTD.

Theorem 2.1. *For the geometric series to converge, the absolute value of q must less than 1 and when* $\mid q \mid < 1$*:*

$$a + aq + aq^2 + aq^3 + \ldots = \sum_{k=0}^{\infty} aq^k = \frac{a}{1-q}$$

2.1.2 Product Rule of Probability

We will use the product rule of probability in Chapter 6 when quantifing the success likelihood of intrusion under a concrete MTD scenairo.

Theorem 2.2. *Let A, B represent events, if* $P(B) > 0$*, then* $P(A, B) = P(A|B)P(B)$*. If* $P(A) > 0$*, then* $P(A, B) = P(B|A)P(A)$*.*

This rule can be extended to more general case.

Theorem 2.3. *Let* $A_1, A_2, \ldots, A_n$ *,where* $n \geq 2$*, represents n events, then* $P(A_1, A_2, \ldots, A_n) = P(A_1, A_2, \ldots, A_{n-1})P(A_n|A_1, A_2, \ldots, A_{n-1})$*.*

2.1.3 Conditional Independency

Conditional independency also plays an important role when we analyze the effectivenss in terms of success likelihood of intrusion.

Theorem 2.4. *Let A, B, C represent three events, if A and B are independent with each other, then $P(A|C) = P(A|B,C)$, $P(B|C) = P(B|A,C)$.*

2.2 Related Concepts

As indicated in Chapter 1, this thesis considers *moving target defense* in cyber security context, more specifically, it focuses on the security of enterprise network environment. Thus, it's necessary to provide a basic introduction to what is an enterprise network.

2.2.1 Enterprise Network Basics

An enterprise network is an enterprise's communications backbone that helps connect computers and related devices across departments and workgroup networks, facilitating insight and data accessibility[49]. An enterprise network also eliminates isolated users and workgroups, facilitates system and device interoperability, effectively combines different communication protocols and improves internal and external enterprise data management. It can integrate all kinds of computer operating systems such as Windows, Linux and Mac computers and mobile devices, such as smart phones and tablets. In addition, enterprise networks can be combined with local and wide area networks according to operational requirements.

The key component that comprises an enterprise network is *machine*, which is a host or node that resides in an enterprise network environment, which can either be physical or virtual.

Typical examples of machines in an enterprise network environment are physical devices, such as servers or workstation computers, routers, cellphones, etc. or virtualized devices such

as virtual machine, virtual switch or virtual router created by virtualization techniques such as VirtualBox[50], VmwareWorkstation[51], KVM[52], Xen[53], LXC[54], etc. A *machine* will be the main unit that will be considered to adapt in this thesis.

2.2.2 A Glance at IT Automation

IT automation has been around for decades as computers and softwares have filled the world. Companies often own hundreds or thousands of physical servers, which are burdens for system administrators to manually setup servers, script operating system and applicaton installations, maintain user credentials and service dependencies, perform network configuration changes, configure storage repositories and deliver security settings. Ideally, all these processes should be automated to make them repeatable, robust and consistent. Enabled by today's virtualization and cloud computing technology, companies are able to easily create and configure tens of thousands or even more of virtual machines. For example, Xen[53], KVM[52], LXC[54], VirtualBox[50], etc. all provide API for creating and destroying virtual machines. Openvswitch[55] provides a programmable virtual switch as an alternative to traditional virtual network interface. Puppet[56], Chef[57], Ansible[58], Salt[59] provides software automation tools. Enabled by these techniques, cloud computing platforms such as Amazon Web Service (AWS)[60], Windows Azure[61], Google Cloud Platform (GCP)[62], OpenStack[63], etc. allow the company to build up their whole infrastructure totally based on virtualized devices. Companies like Netflix[64] and Adobe[65] are concrete examples that use Amazon Web Services as their enterprise network infrastructure.

Figure 2.1 shows an example network environment. Traditionally, to reduce the workload of configuring each server/workstation manually, different operating system templates and software packages are stored in the network file servers. Administrators can use automated procedure such as contained in bootable disks to retrieve required operating system and software packages from file servers or network repositories to manage each server/workstation.

However, this method still needs administrators to configure each machine locally and in-

dividually. To reduce the workload even further, tools like Cobbler[66] use Preboot Execution Environment (PXE) booting[67] and a kickstart technique to remotely provision operating systems on hardware servers. By taking advantage of this technique and Wake-on-Lan (WOL)[68], Metal as a Service (MAAS)[69] from Ubunt enables administrators to treat physical servers like virtual machines in the cloud. Administrators will be able to easily power up, tear down, provision the OS, check or redeploy hardware nodes at will over the network.

Juju[70], which is an cloud automation tool that enables administrators to configure, manage, maintain, deploy and scale the services running in each machine efficiently. With predefined *charm* files, which are essentially configuration commands for different applications, users can easily build up infrastructure with provided services as defined in these files. In addition, users can also easily create their own *charm* files using configuration management assets such as Puppet[56] or Chef[57] to bring up their customized infrastructure. When MAAS and Juju are paired, it turns physical machines into an elastic cloud-like resource. From this perspective, the cloud is just an example where all physical machines are virtualized.

Consider Figure 2.1. The following describes the typical process when use MAAS and Juju together to deploy and configure services on bare-metal servers.

1. Install MAAS in Controller. This includes downloading Operating System Images from Ubuntu and these images could be saved in the File Server.

2. Use PXE to remotely signal and recognize machines A, B, C (mainly through MAC address of the network interface on that machine). Then register and enlist machine A, B, C in MAAS.

3. MAAS uses WOL to remotely power up machine A, B, C and then uses PXE boot to retrieve the Operating System Images from MAAS and install Operating System to A, B, C.

4. Install Juju in Controller.

5. Use Juju to deploy and configure services to machine A, B, C using *charm* files. Example services could be web server services such as Apache and Tomcat, database services such as MySQL and PostgreSQLs. Juju also supports to easily establish and maintain relationships between services.

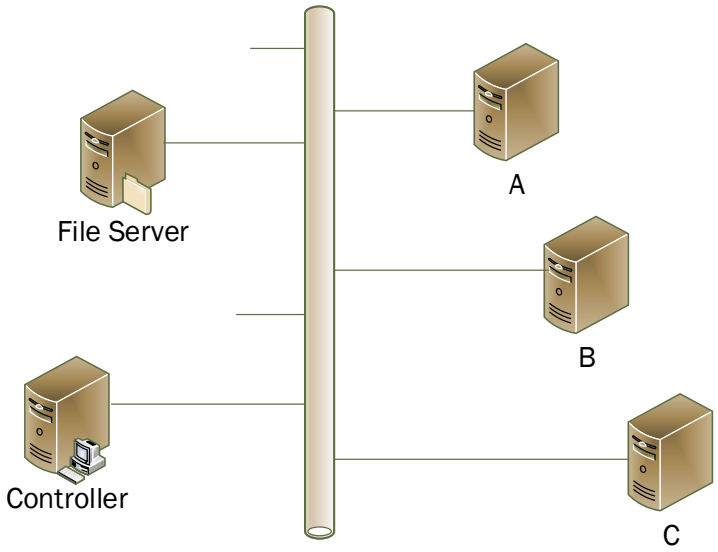


Figure 2.1: IT Automation Overview - Hardware

In a cloud environment, as shown in Figure 2.2, virtualization techniques such as Xen, KVM, LXC, Virtualbox, can be used to replace MAAS and easily bring up, tear down and deploy virtual machines (VMs). When these machines are initially created, the cloud platform can use stored images (which are actually OS templates) and user data (which are scripts that run after the OS is installed to set up user accounts or install basic software) to configure the provisioned VM. Such software could be Puppet or Chef configuration automation tools whose daemon processes run in the background and wait for configuration commands. After the VMs are up and running, the master controller node can push further configuration settings remotely to these provisioned VMs to setup various services.

Instead of simplify maintaining, configuring and remotely deploying various services to

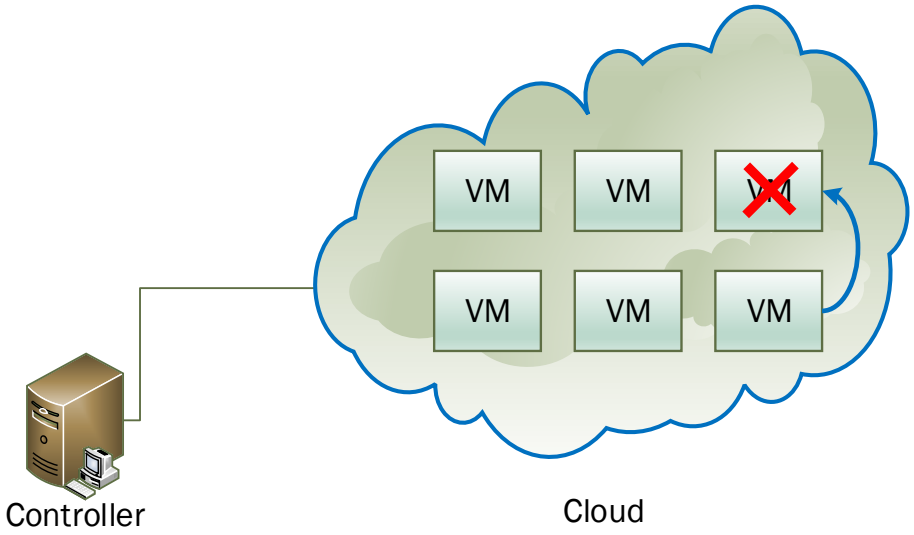


Figure 2.2: IT Automation Overview - Cloud

different hosts, new techniques, such as Software-defined Networking (SDN)[71] and OpenFlow[72] provide the administrators a programmable and flexible way to make traffic flow decisions and manage network communications. Enabled by these techonologies, one could imagine the potential of creating and maintaining versatile and dynamic network topologies for *moving target defenses*.

2.3 Conclusion

This chapter introduces important background information that will be helpful to comprehend this thesis. Firstly, it presents some mathematical definitions and theorems as a reference. Secondly, it discusses the main application context, which is enterprise network systems, that we focus our MTD theory towards. Last but not least, it provides a brief overview of cloud computing and state-of-the-art configuration management techniques, that will help in understanding some of the key definitions involved in Chapter 6's MTD

theory. This introduction to IT automation will also help the reader understand why these definitions are practical and applicable in enterprise network environment.

Chapter 3

Related Work and A Taxonomy

The initial concept of moving target defense was proposed in the 2009 National Cyber Leap Year Summit[1,2]. Since then, a lot of research devoted to this area has been published. Although *moving target defense* as a concept is new, the ideas behind it have emerged from previous work.

A formal classification and well-defined taxonomy will not only help better understanding related work from several different dimensions, such as *how*, *when*, *what* to move, and *which* attack to thwart, but it will also support the potential of combining different techniques to foster a holistic approach to MTD. Based on a survey of related work, Figure 3.1 presents a diagram that categorizes previous and ongoing research. To the best of our knowledge, this taxonomy is the first to try to categorize and organize state-of-the-art research in MTD.

When talking about *moving target defense* systems, key questions include:

- "What is a *moving target defense* system?"
- "What can be moved?"
- "How can it be moved?"
- "What are the security benefits of such a system?"

To help answer these questions and better understand moving target defense, a taxonomy for MTD research is proposed. This review and taxonomy also supports the theoretical framework proposed in Chapter 6, which includes a formal definition for *moving target defense* systems.

In the first level of the taxonomy, moving target defense research has been classified into four categories according to the *Attack Type*, *Adaptable Aspects*, *Tactics*, and *Strategies*. The classification is driven by the above list of key questions and refined by adopting some military terminologies, such as tactics and strategies. In the *Attack Type* category, the related work that claims to use *moving target defense* ideas to thwart different types of attacks is summarized. This category helps to clarify the security benefit of an MTD system. Similarly, the *Adaptable Aspects* category groups research by the configuration aspects or parameters that can be changed to achieve the defensive objectives. The *Tactics* category categorizes the different techniques used to achieve different moving mechanisms. This category groups these techniques into two sub categories: diversification and randomization. These two terms are widely used in the literature but without a clear definition or distinction of their relationships. For example, diversification actually provides space for randomization. This category will clearly reflect this relationship. The Strategies category categorizes work by when a move should be launched. Entities in this diagram are then grouped according to the different characteristics of the four categories.

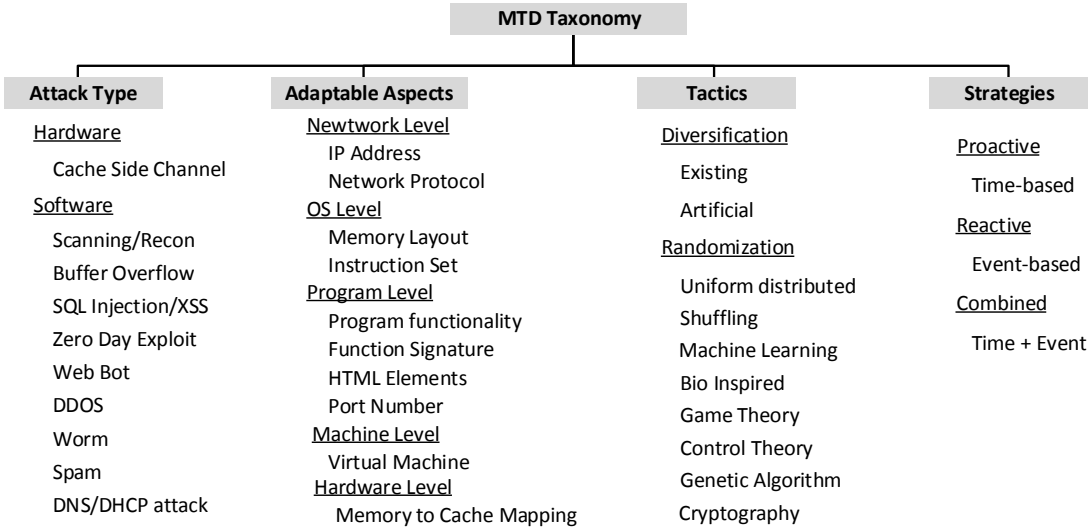


Figure 3.1: MTD Taxonomy

3.1 Attack Type

Cyber attacks target both hardware and software. Various techniques have been created and utilized to either steal, alter, or ruin a specific target by hacking into a vulnerable system. Although MTD research is at an early stage, researchers have already considered adopting this concept to protect the system, which includes both hardware and software aspects.

3.1.1 Hardware Attacks

From the *hardware* perspective, a well known type of attack is side channel attack. Side channel attacks[73] are a form of reverse engineering by which an attacker can infer how the electronic circuits works and what data it is processing by analyzing the information leakage such as heat, electromagnetic emissions or even sound, without direct access to the circuits itself. Lee[74] presents a moving target defense approach for secure hardware design. The motivation is that despite strong cryptography and software isolation mechanisms, cache side-channels can still leak critical information. Lee proposes a novel cache design in which

a redesigned address decoder circuit and a longer cache index, coupled with a random replacement algorithm, allow it to perform dynamic memory to cache mapping. As a built-in hardware component, there is no software modification required. However, despite the benefits, the accurate performance comparison with conventional caches and verification of security enhancements are challenging.

3.1.2 Software Attacks

From a *software* perspective, the moving target approach has also been adopted to thwart some of the most typical cyber attacks.

3.1.2.1 Scanning/reconnaissance

Scanning/reconnaissance is a critical step for information gathering before an actual intrusion is launched. Attackers usually use existing or customized tools to scan the target network and system in order to figure out host information, such as OS type, IP address, open ports, running services and potential vulnerabilities. Thus, one way to effectively enhance system security is to invalidate the scanning results. Al-Shaer *et al.*[27] presents an architecture called Mutable Networks or MUTE for moving target defense. The main goal is to restrain an attacker's capability in scanning or discovering network targets by dynamically changing network configurations, such as IP addresses and routes. It uses a formal approach based on BDD (binary decision diagrams)[75] to model network behavior as well as create randomized and constrained network configuration mutations. However, in its current form, the work is purely theoretical. To make it practical and effective, MUTE must be fast, unpredictable, operationally safe, deployable, and scalable.

Jariath *et al.*[28] uses OpenFlow to develop an MTD architecture that transparently mutates IP addresses with high unpredictability while maintaining configuration integrity and minimizing operation overhead. The key idea is to not touch the real IP address, which can only be reached by authorized entities, but instead mutate a host's virtual IP address,

which can be acquired by the DNS. The benefit of this technique is that it thwarts network scanning, which is the key step in network intrusion. The emulation results are impressive as they claim to invalidate 99 percent of external scanners and saves up to 90 percent of the hosts from zero-day unknown worms. However, the cost to user's normal operation is unknown. Also, the author indicates that more investigations need to be performed for other types of attacks, such as DDoS and application-layer attacks.

Research[18] has shown that scanning an IPv6 subnet takes around $8.77 * 10^{10}$s. MT6D[20] provides an effective way to thwart scanning attacks in the IPv6 environment by dynamically changing IP addresses to invalidate scan results. MT6D[20] can also be used to prevent attacks where scanning/reconnaissance are used to obtain privacy and anonymity, as IPv6's StateLess Address AutoConfiguration (SLACC) track protocol potentially exposes hosts information, which could potentially be used to map human users to network traffic if the network remains static. Another scanning/reconnaissance-based attack, the man-in-the-middle (MITM) attack, can also be thwarted as MT6D rotates the addresses of the attacker's intended victim.

3.1.2.2 Buffer overflow

Buffer overflow usually happens due to the lack of buffer boundary checking. Manipulated input can overwrite the adjacent memory and alter the original way the program performs. Many software vulnerabilities exist because of such flaws. Popular attacks such as *remote exploits* and *worms* work over the network to take advantage of such vulnerabilities to obtain control of systems or to cause denial of service without prior access to the vulnerable system. Existing techniques such as StackGuard[76] and ProPolice[77] help to mitigate the buffer overflow attack. From a moving target defense perspective, address space layout randomization (ASLR)[22] and address space layout permutation (ASLP)[23]– which randomizes the memory positions of the stack, heap and libraries of running programs–provides orthogonal enhancements[24] to help further invalidate the buffer overflow related attacks.

3.1.2.3 SQL injection/XSS

SQL injection/XSS is a technique that tries to insert malicious SQL statements into a web application's entry field for execution, so that database contents can be revealed to the attacker. Usually the vulnerabilities used by *SQL injection* are due to incorrectly filtered string literal escape characters and weakly typed user inputs. It can be thwarted by applying instruction set randomization to SQL language keyword[38]. *Cross-Site Scipting(XSS)* takes advantage of known vulnerabilities in web applications and injects malicious scripts into a compromised sites' web pages that deliver modified contents to client-side web browsers from trusted sources so that sensitive credential or session information are gained by the attacker. Gundy *et al.*[78] presents *Noncespaces*, which utilizes HTML source randomization to enable web clients to easily identify untrusted content to prevent XSS vulnerability exploitation. By randomizing various database aspects, such as table and column names and javascript runtime APIs, end-to-end software diversification[29] techniques can be used to mitigate SQL injection and XSS attack.

3.1.2.4 Zero day exploit

Zero day exploits are software bugs or holes that are found by hackers and can be used to gain privilege or corrupt the vulnerable system that all previously unknown to the software vendor or developers. Although it is usually first known only to a small group of hackers due to the significant effort required to discover and develop effective exploit, today's ability to quickly share information can lead to severe problems. SCIT-MTD[41,42,79,80] proposes to utilize virtual machine rotation to reduce the intrusion time window and increase attack difficulty. Here rotation means bringing online servers offline and launch a cleansing procedure, which will remove potential malware and thus mitigate the impact of zero-day vulnerabilities. Then, offline servers that are already in a pristine state are used to replace these servers to keep providing services for clients. However[31], SCIT uses an open source version of Terracotta[81] to share information among servers, which means it's not a completely stateless

solution and poisoned states can still be disseminated.

In addition, OpenFlow random host mutation[28] results show it can save up to 90 percent of network hosts from zero-day worms. Although zero-day vulnerabilities can bypass IDS systems, Chapter 4 shows that, an MTD system that randomly refreshes various hosts helps mitigate this type of attack, since all nodes will eventually be removed from the system.

3.1.2.5 Web bot attack

Web bot attacks use automated programs to perform large amounts of web browsing behavior attacks, such as account registration/login, comment form/email spamming, online game/vote cheating, etc. Fifty one percent of web site traffic is non-human and is often malicious as the result of various automated hacking tools and web bots[35]. NOMAD, a novel approach that randomizes key HTML elements, is proposed to prevent the automated web-bot based resource access, form or comment submitting attacks while not affecting legitimate users' access[35].

3.1.2.6 DDos attack

DDoS attack represents the distributed denial of service attack, which uses a large number of compromised systems (botnets) to attack a single target by flooding traffic messages from multiple sources to force it shutdown or to deny services to legitimate users. Jia *et al.*[82] developed MOTAG, which employs dynamic, hidden proxies as moving targets to secure service access for authenticated clients against flooding DDoS attacks. To reach protected services, the authenticated clients are assigned to individual proxy nodes that will redirect packets and enforce session policy checking. When the proxy is under DDoS attack, the client will be re-assigned to an alternative proxy to evade the ongoing attack and maintain legitimate access. Also, since DDoS attackers assume end-hosts or targets use fixed IP-address or routes, the proposed MUTE[27,28] and MT6D[20], which proactively change the IP addresses and routes, can also be used to protect network infrastructures against this type

of attack.

3.1.2.7 Worm attack

A *worm* is usually a stand-alone piece of malware that employs the network to replicate and spread itself to other machines by relying on existing software bugs or holes in the target system. The damage caused by a worm can be as simple as just consuming bandwidth or as complex as turning worm infected machines into botnets that can be used by its owner to profit from spams or DDoS attacks. Portokalidis *et al.*[38] shows the potential of instruction set randomization to contain some extremely elaborate worms like Conficker and Stuxnet. A *hitlist worm*, which takes advantage of a precalculated lists of vulnerable targets, can spread extremely fast and infect a million hosts in less than two seconds[83,84]. Antonatos *et al.*[19] proposed a defense mechanism called *network address space randomization* (NASR) to intentionally accelerate hitlist decay to contain the spread speed. There are limitations to the applicability of NASR, such as services that require static addresses, applications that do not tolerate address changes, and ineffective against DNS hitlist worms. However, results showed that it is effective in restricting and slowing down the infection of *IP hitlist worms* and forcing these worms to exhibit scan-like behavior, which makes them more easily detected.

3.1.2.8 Spam

Spam is usually thought of as unsolicited emails or news postings. Spam filter designers try to train good filters by using recently received emails. On the other hand, spammers try to reverse engineer the existing filters to generate messages that can circumvent these filters. Colbaugh[85,86] proposed to introduce movement into the classifier-based defense. The key strategy is first to divide the original feature set F for each activity behavior into K randomly selected subsets. Next, they train one classifier for each subset of features to get K classifiers. Then specify a scheduling policy to select these classifiers to minimize the

effectiveness of adversary adaptation. Results based on empirical study showed that this strategy outperforms static filters.

3.1.2.9 DNS/DHCP attack

The Domain Name System (DNS) provides the mapping from easily remembered host names to IP addresses. Attacks targeting DNS usually try to modify the domain name server's database to divert the traffic to other malicious addresses. The SCIT-DNS[87] approach– where a cluster of servers constantly rotate between primary DNS, secondary DNS, Offline for Cleaning and Cleaned for rebooting states to confine the damage of successful attack to a limited time–digitally sign dynamic DNS updates using a private key while keeping the key offline at all times. The authors claim SCIT-DNS ensures high availability and master file data integrity even in the face of unknown or undetected attacks. However, as indicated in the paper, extending the current cluster model to an arbitrary number of servers instead of three and utilizing rotation schemes other than pure round robin requires additional work. The Dynamic Host Configuration Protocol (DHCP) works to automatically provide basic parameters for network interface configuration. However, the weakness of this protocol is that it provides the attacker a possibility to masquerade as either valid DHCP clients or servers. Rowe *et al.*[88] suggests introducing diversity to the DHCP protocol by inserting more states and transitions to the protocol's finite state machine to thwart such attacks. More discussion for this work will be given in Section 3.2.1.3 and 3.3.2.6.

3.2 Adaptable Aspects

Deciding what to move, or the adaptable aspects to use, is a key step in developing moving target defense systems. This decision has a direct impact on what kinds of attacks the MTD system will be able to constrain. It turns out that classifying research according to the different levels of a machine's functioning environment is very natural. The levels used

in this classification include the the *network level*, *OS level*, the *program level*, the *machine level* and the *hardware level*.

Network level configurations range from physical network to logical network. Elements include physical devices, such as routers, switches and firewalls, policies, such as router configuration and firewall rules, protocols, such as TCP/UDP/IP/NAT, logically separated networks, such as VPN, and network location identifiers, such as IP/MAC addresses.

OS level configurations abstract the operating system installed on top of either a physical or virtualized hardware and provides the context for applications run at program level.

Program level configurations include all the applications or programs that can be installed and run in an operating system. Applications or programs run at this level needs the environment provided at OS level as well as network level to communicate with remote machines.

Machine level configurations are composed of the adaptable aspects range over the above three levels. This level mainly represents that a machine (either physical or virtual) itself, as a configuration unit, can be adapted.

A machine can be either physical or virtual. The Hardware level abstracts the underline environment a machine runs in and mainly refers to the physical configurations, such as CPU, memory, cache, circuits, disks, etc.

3.2.1 Network Level

At the *network level*, to increase the difficulty of identifying the target, the *adaptable aspects* can be divided into three subcategories: *IP address*, *port number* and *network protocol*.

3.2.1.1 IP Address

IP addresses have been considered an adaptable aspects in previous work[20,27,28,32,33,34] to increase the intrusion difficulty and thwart different types of attacks, such as scan/mapping,

worms, DDOS, etc. Dynamically changing IP addresses can effectively invalidate the addresses discovered by attackers that they use to target the victim machines.

3.2.1.2 Port Number

Port number also has been proposed as an adaptable aspect[27,32,33,34] that could benefit moving target defense. Dynamically changing port numbers could make it more difficult for the attacker to locate the target service. It provides a similar effect as IP address changing (see section 3.2.1.1) and can be used together with IP address changing to support the exponential property for layered security, as discussed in Chapter 1.

3.2.1.3 Network Protocol

Network Protocols have also been thought of as adaptable aspects to thwart network based attacks. Rowe *et al.*[88] propose to insert artificial states and transitions into existing network protocol state machines to counter network-based attacks. The authors used a modified DHCP protocol to demonstrate how this idea can be used to thwart the DHCP attack. By inserting a new state into both DHCP server and client, forged DHCP messages sent by the attacker will move the victim client or server to a state that is unknown to the attacker. However, valid clients or servers will have the modified protocol specification and will be able to complete the state transition. As pointed out by the author, this approach may introduce increased costs due to service interruption when switching to the new and shared secret states.

3.2.2 OS Level

In *OS Level*, existing research has focused on randomizing the *memory layout* and diversifying the *instruction set*.

3.2.2.1 Memory Layout

Memory layout mainly refers to the segments of an executable file such as stack, heap, data and linked libraries' addresses or positions in the memory. Previous attacks such as remote exploits and worms take advantage of the standard layout of compiled programs and relies on smashing the memory through malicious code injection to alter the ordinary execution. Randomizing the actual memory addresses of executable segments or data has been shown to noticeably reduce such attacks[22,89,23].

3.2.2.2 Instruction Set

The *Instruction set* represents the machine language or commands that can be understood by a specific type of processor. Guanav, Boyd *et al.*[25] proposed instruction set randomization (ISR) as an approach to counter code-injection attacks. By creating process-specific randomized instruction sets, an attacker who doesn't know the key of the randomization algorithm will fail to inject code that can be recognized by the processor, thus leading to a runtime exception. Later[26], to counter the attacks that attempt to circumvent the low-level machine language ISR, Boyd *et al.* demonstrates that this approach can also be applied to high-level scripting or interpreted languages, such as Perl and SQL. In Global ISR[38], they propose the holistic adoption of ISR across all system layers, which requires all binaries in the system to be pre-randomized with different secret keys. New installations require user authorization to continue and finish this randomization process, which makes unauthorized injection of binary code unable to execute as they are unrecognized.

3.2.3 Program Level

Many applications are built on top of OS. A modern software system's complexity almost guarantees that vulnerabilities will exist, resulting in an imperfect security system. *Adaptable aspects*, at this level, include three areas: *program functionality*, *function signatures* and *HTML elements*.

3.2.3.1 Program Functionality

Due to the high cost of developing software systems to fulfill a wide range of user needs, continued feature augmentation, backward compatibility and reusability, many software systems are complex and provide more functionality than users require, desire or are even aware of. Rinard *et al.*[90] views security vulnerabilities as undesirable functionality present in a system. They discuss several low-level techniques, such as input rectification, functionality replacement, loop perforation, cyclic memory allocation, *etc.* to help either excise or change system functionality that *may* help eliminate security vulnerabilities while still leaving the program able to provide acceptable functionality. From MTD perspective, eliminating security vulnerabilities can be viewed as reducing the attack surface.

3.2.3.2 Function Signature

Function signatures refer to the names, key words and static identifiers used in a software system to communicate between system components. Concrete examples include database table or column names, application programming interface (API) function names, etc. Christodorescu *et al.*[29] shows how to apply transformations to subprograms and execution environments so that each program instance actually uses syntactically and semantically distinct subprograms to diversify the function signatures. For example, they demonstrate that by randomizing the identifiers associated with various aspects of database interface, such as the table name and column name of a table, the difficulty of SQL Injection is increased. Also diversifying the JavaScript runtime environment, such as randomize the Document Object Model API method names, XSS attacks can be mitigated.

3.2.3.3 HTML Elements/Programming Language

HTML elements usually refer to the specific source component of a web page that have attributes and contents. Critical *HTML elements* include registration forms, submission and file upload buttons, etc. These are usually used by servers to collect or update information

stored in a remote database. NOMAD[35] focuses on how to randomize key HTML elements to prevent widely used web bot attacks with a relatively low overhead. This work can also be viewed as a specific example of end-to-end software diversification as defined by Christodorescu *et al.*[29]. *Programming languages* can be viewed as specific instruction sets at the application level. As discussed in Section 3.2.2.2, concrete examples include techniques that randomize SQL and Perl language keywords.

3.2.4 Machine Level

A *Machine* as a unit to provide redundancy, improve fault tolerance and ensure high availability has been wildly used for decades. From a moving target defense perspective, SCIT[79,87,42], TALENT[40], ChameleonSoft[43], MEERKATS[39] and this thesis[32,33,34] all consider virtual machines as a unit of replacement. Combining machine replacement with other adaptable aspects from network, OS or program level will provide more flexible adaptation choices and defeat a broad range of attack types as shown in Chapter 6.

3.2.5 Hardware Level

Adaptable aspects from the *hardware/physical level* in current research has only one subcategory, which is *memory to cache mapping*.

3.2.5.1 Memory to cache mapping

As discussed in Section 3.1.1, side channels can leak secret information despite strong cryptography and software isolation mechanisms. These hardware problems are difficult to solve using only software solutions. Lee[74] proposed to design new hardware circuits for caching that enables dynamic and random mapping of memory addresses to cache instead of a direct and static mapping to foil the cache side-channel attack. The benefit of such design is that it has no impact to the software. However, there are challenges as already indicated in Section 3.1.1.

3.3 Tactics

After determining which adaptable aspects to move, the next step is to consider how to move. From a military perspective, the techniques used to achieve this movement are called *tactics*. In this section, the most popular techniques used in current MTD research are summarized and categorized into two sub categories: diversification and randomization. These two terms are popular in related literatures but with some key distinctions and relationships missing.

The first distinction is about diversification. Generally, diversification introduces functionally equivalent but internally unique variants of a piece of software. These variants form a space for selection. Some spaces, such as IP address space, memory space and port number space, are introduced by existing diversification. While other spaces, such as instruction set space and function signature space, are created by artificial diversification to relatively fixed configuration aspects, such as instruction set and function signature. This is why diversification category has two sub-categories: existing and artificial.

Randomization is closely related to diversification, as diversity provides the space for randomization. In the physical world, for military units to perform tactic maneuvre, space is clearly indispensable. This is also true for moving in the cyber world. The difference is that space in the physical world is three dimensional, while in the cyber world the moving space is created by diversification.

In addition, randomization usually refers to randomly choosing a variant from current space. Random choice is usually understood as making selections such that the probability of each selection forms a uniform distribution. However, if we view randomization as a decision making process to make selections based on a specific probability distribution applied to the variants in current space, then it turns out randomization can generalize to all AI techniques, such as genetic algorithms, machine learning, game theory, etc. Indeed, saying that a specific variant will always be chosen given corresponding environmental information is nothing more than saying that the probability of choosing this variant is 1 with the probability of all other variants at 0. This way, it captures another key missing relationship

between randomization and all other AI techniques which is also why these AI techniques are put as sub-categories of randomization. The formal definition of diversification and randomization is given in Chapter 6.

3.3.1 Diversification

Cloud or enterprise configurations typically prefer identical or homogeneous settings for a large number of servers due to the ease of maintenance. However, this leaves the potential of a single vulnerability being exploited by an attacker to compromise many similarly configured machines. Diversification was proposed to mitigate such situations as an attacker may compromise one variant but will probably not have the knowledge of all different variants. Alternatively, forcing the attacker to learn all variants will consume significant attacker resources and time.

Evans *et al.*[91] presents a model for thinking about dynamic diversity defenses. The authors analyze a few example defenses and attacks using the model and show scenarios where MTDs are and are not effective. The authors present several of the most commonly used low-level automatic diversity techniques, such as address space randomization and instruction set randomization, which range over both existing and artificial diversification space. Results show that these low-level diversity techniques have limited effectiveness to high-level attacks and the authors suggests using composition and N-variant systems to improve the effectiveness of diversify defenses. Composition strategies require the attacker to break multiple different diverse defenses simultaneously, whereas N-variant systems require an attacker to break multiple variants of the same diversity defenses simultaneously. ChamelenonSoft[43], MEERKATS[39] and N-Variant[37] all use these strategies in their system design. Similar principles have also been proposed by Jackson *et al.*[30] in their compiler-generated software diversity work where two orthogonal techniques have been suggested. One is multi-variant execution, where a monitoring layer can monitor and execute diversified variants and examine difference in behavior to indicate possible attacks. The other is

to use large-scale software diversity where all users get their own unique variant. Since the attackers have no knowledge of the internal structure of each variant, they cannot construct an attack. In short, MTD systems should make proper use of existing as well as artificial diversification space to improve their effectiveness.

3.3.1.1 Existing

As discussed in adaptable aspects , there are MTD work tries to randomize the selection of several configuration aspects by taking advantage of existing configuration space, such as IP address space[18,19,20,21,27,28,32,33,34], port number space[27,32,33,34], and address layout space[22,89,23]. All these are related to the available choices that already exists in the system.

3.3.1.2 Artificial

Diversification also refers to proactively diversifying relatively fixed configuration aspects such as instruction sets, HTML elements, etc. to provide new space for randomization. There are mainly four artificial diversity techniques.

One is to use cryptography algorithms. Perhaps the most obvious example is our daily usage of password to log into various systems. Christodorescu[29] suggests diversifying identifiers, such as database tables or column names and javascript runtime APIs, to thwart SQL injection/XSS attacks. Global ISR[38] seeks to diversify the instruction set from all levels such as machine/assemble language, SQL and Perl language keywords, to thwart code injection and SQL Injection attacks. NOMAD[35] and Noncespaces[78] both try to diversify HTML sources to thwart web bot and XSS attacks. All these work make use of cryptography techniques to enlarge the fixed configuration factor's space.

Another artificial diversity technique takes advantage of compiler techniques. Instead of generating fixed machine code for a given program, compilers techniques[30] allows us to generate functionally equivalent but internally unique variants of the program. Adopting compiler-generated software diversity makes it less likely that a single attack (such as worms)

could affect large numbers of targets. In addition, it will be no longer possible for an attacker to analyze their own copy of software to find exploitable vulnerabilities.

The third diversity technique takes advantage of cloud computing and IT automation. As discussed in Chapter 2, virtualization platform and configuration management tools can be used to easily create, configure and deploy services to machines, which potentially provides a powerful way to compose different machine, OS, service implementations, etc. to provide the same functionality to the user. Huang[31] proposes to introduce such diversity for web services to better protect web servers.

The fourth diversity technique is to artificially manipulate data files, protocols, network topologies, etc. Intended users can copy, slice or regroup data files at will to fill their purposes. For example, in Section 3.1.2.8, Colbaugh *et al.*[85,86] proposed to diversify classifiers by dividing the original feature set into K randomly selected subsets to train and get K classifiers instead of one. Rowe *et al.*[88] suggests generating a diverse set of DHCP protocols by artificially injecting intermediate states. Chapter 2 also discussed the potential to diversify network topologies through Software-defined Networking (SDN)[71] and OpenFlow[72].

In general, these four artificial diversity techniques together with existing diversification can be used to achieve the suggested Composition and N-variant principles for MTD system.

3.3.2 Randomization

Randomization has been used in existing MTD research to make intrusion more difficult[27,28,20,38,22,23,18]. The key idea in randomization is to take advantage of existing or artificially introduced configuration space to randomly pick the next system configuration variants in these spaces to add more uncertainty to the attacker, which in turn increases the difficulty of intrusion. As indicated, randomization is usually understood as making selections by assuming the probability of each selection forms a uniform distribution. However, this concept generalizes if we view it as a decision making process that makes selections based on a specific probability distribution. This allows AI techniques to be viewed as

producing different probability distributions of variants in the current space and uniform distribution becomes a specific instance of randomization.

3.3.2.1 Uniform Distributed

Several existing approaches[27,28,20,18] try to randomize the machine's IP address in existing IPv4 or IPv6 configuration space to increase the difficulty of locating the target. Memory layout randomization[22,23] uses existing memory address space to randomize memory positions of the stack, heap and libraries of running programs to add difficulty to buffer overflow attacks. Global ISR[38] diversifies instruction sets for an OS or programing language, which can be considered as an artificial instruction set space. During runtime the actual working instruction set can be randomly selected from this artificial space, which will significantly increase the difficulty in correctly identifying the real instructions. Although not explicitly indicated, these approaches all make selections based on uniform distribution which applies the maximum uncertainty to attacker. This thesis[32,33,34] investigates the effectiveness of randomly picking virtual machines according to uniform distribution from a group of VMs to change IP address to prevent remote exploit attacks.

3.3.2.2 Shuffling

As discussed in Section 3.1.2.6, MOTAG[82] hides the proxy nodes from the public forcing attackers to use insiders to locate proxy nodes and attack them. To quarantine insider-assisted attacks, MOTAG employs a shuffling mechanism to randomly pick proxy nodes to defeat them. The author gives a detailed analysis for shuffling optimization and then presents a greedy shuffling algorithm. The experiment results show that MOTAG can protect a majority of innocent clients from DDoS attacks assisted by hundreds of insiders within a small number of shuffles.

3.3.2.3 Machine Learning

In Section 3.1.2.8, a machine learning based approach, which diversifies classifiers to add difficulty for spam adversaries to reverse engineer spam filters, was proposed by Colbaugh and Glass[86]. The approach decreases the adversarie's ability to generate junk email with enough difference from the training data to bypass machine learning-based filters. The authors present a scheduling policy (see Section 3.3.2.5) to select these classifiers and claims the strategy is simple, flexible and near optimal for a broad range of security problems. Case studies show that the proposed algorithm outperforms standard static methods. However, the data set used for network intrusion detection analysis is specific –KDD Cup99[92] and there are harsh criticisms on these datasets[93,94].

3.3.2.4 Bio Inspired

Cui[95] proposed a new host-based defense mechanism that they call Symbiotic Embedded Machines (SEM) or Symbiotic. It is based on perpetual mutation and diversity that is inspired by a natural phenomenon known as Symbiotic Defensive Mutualism. This phenomenon generally refers to any association between different species where the survival or evolutionary fitness of one or more partners is enhanced by the association. The key idea in this approach is that defenses are provided by defensive mutualism. First, each Symbiote and host program is created uniquely by rewriting using advanced polymorphic code engines. This diversity provides inherent protection against attackers. Second, Symbiote treats the entire host program as an external and untrusted entity, which eliminates the traditional trust relationship between anti-virus software and the underlying OS. This way, Symbiote has full visibility into the code and execution state of its host program, which can passively monitor or actively react to the exploitation and incorrect behavior at runtime. At the same time, host program requires Symbiote to successfully execute (which costs computation resources) in order to operate. Any successful attempts to disable, modify or remove the Symbiote will render the host program inoperable. The author describes the Symbiote

code structure and argues it can reside within any level of the software stack due to the fact that it makes no assumption about the functionality of the host program. The idea sounds promising, but the author does not provide a prototype system. In addition, the impact on system performance and functionality are not addressed. Azab *et al.*[43] propose a biologically-inspired defense framework called *ChameleonSoft*, which builds over a novel cell oriented architecture. More discussion about this will be given in Section 3.5.

3.3.2.5 Game theory

Colbaugh[86] investigated MTD in the framework of a repeated two-player games with incomplete information. The defense system and adaptive adversaries were modeled as a hidden mode hybrid dynamical system (HM-HDS) to specify a scheduling policy for selecting classifiers during each time period to minimize the effectiveness of adversary adaptation. However, key assumptions are made regarding HM-HDS that only hold under some conditions or applications in which empirical data assessment are allowed. Manadhata[44] modeled the interaction between defender and attacker as a two player stochastic extensive game[96] and used game theory to determine optimal defense strategies. However, as admitted by the author, the potential state space explosion and action space explosion are disadvantages of this model.

3.3.2.6 Control theory

Rowe *et al.*[88] view MTD as an optimal secure reconfiguration problem. The goal is to develop novel control theoretic approaches such that, based on a range of cyber maneuver techniques, a most cost-effective technique can be selected to counter a detected ongoing attack. The authors model the defense as a closed control loop where state transitions occur due to contingencies as well as selected cyber maneuvers. The paper proposed a maneuver to thwart the DHCP attack. By generating a diverse collection of DHCP protocols through artificially injected intermediate states and altered state transition paths and messages,

attackers who make use of the standard DHCP state machine will fail. However, as revealed by the authors, other maneuvers could also be considered. In addition, estimating the security state and selecting the most cost-effective maneuvers between different states are primary research challenges.

3.3.2.7 Genetic algorithm

Crouse[97] proposes an interesting approach to model a computer's configuration as a chromosome where an individual configuration setting is a trait or allele. Then the author employs a Genetic Algorithm to find temporally and spatially diverse secure computer configurations. The paper presents the experimental results based on a simulated environment composed of 256 computers, each with 80 parameters. However the author does not show what these parameters are. The paper shows results from temporal and spatial diversity, configuration vulnerabilities and perceived configuration vulnerabilities aspects. The experiments start with vulnerable configurations and eventually find more secure and diverse configurations. However, the impact of the evolving genetic algorithms to system operations is unknown. In addition, it is also unknown how the author deals with system configuration dependencies and constrains.

3.3.2.8 Cryptography

Yackoski *et al.*[98,21] designed the SDNA architecture to work on top of IPv6. A hypervisor within each network node transparently rewrites packets entering and exiting the OS to conceal real addresses from OS. Each SDNA entity operates independently to process packets and directly coordinates with other relevant SDNA entities to facilitate communication, achieve scalability and avoid single point of failure. During transmission, packets generated from the OS will be intercepted by SDNA, who uses packet metadata and a cryptographic key to rewrite the IP addresses before sending it to the network. When receiving packets from the network, SDNA verifies the packets using a shared key and then rewrites the

IP addresses in the packets before offering the packets to the OS. Using this approach, various dynamics can be imposed at the network layer. Packets that failed verification can be dropped or redirected to a honeypot. However, SDNA interferes with network security components such as IDS, logging, etc. To accommodate SDNA dynamics without modifying these security components, one could have intermediate nodes reconstruct the original packets based on SDNA mechanisms. To deploy SDNA on existing systems, it requires enabling IPv6 and modifying the network configuration. Also, an SDNA entity and hypervisor must be added to each host.

3.4 Strategies

Another category is *Strategy*. In military terms, strategies can be understood as a high level plan to achieve military objectives or win the war. Similarly, here Strategy is used to summarize high level plan that can be adopted to win the cyber war. As discussed in Chapter 1, MTD systems seek to proactively enlarge the exploration space during design and either proactively or reactively shift the attack surface during runtime. In proactive mode, these changes occur at random times, thus a *proactive strategy* is based on time. In reactive mode, when incorporating detectors, such as IDS and system monitors, a *reactive strategy* becomes event-based. These events could be related to intrusions, operational errors, system failures, etc. Obviously, as captured by Figure 1.1, proactive and reactive strategies could coexist in MTD system, which was titled as the third strategy type: *combined*.

3.4.1 Proactive

With a proactive strategy, MTD systems launch adaptations at will, which is summarized as a time-based sub-category.

3.4.1.1 Time-based

SCIT[87,42,79] uses a fixed time interval to constantly rotate and replace virtual machines to restrict the damage of successful attacks to a limited time. Each virtual machine loops and rotates through three phases: online working, offline for cleansing potential compromise, and rebooting to replace current online functioning VMs. More discussion about SCIT can be found in Section 3.1.2.4 and Section 3.1.2.9. Later, Huang[31] suggests to introduce diversity by combining different OS, web server programs and web applications to create a moving attack surface for web service. It takes a similar routine as SCIT to rotate the virtual machines. Rowe[88] suggests to proactively launch cyber maneuvre where a specific maneuvre that artificially injects intermediate states and alter DHCP protocol state machine's transition path has been proposed. Section 3.3.2.6 gives more discussion about this work. In MT6D[20], Jariath[28] proposes to proactively change the IP address at specified time and frequency. Chapter 4 also presents a pure random MTD system design that seeks to proactively change different adaptable aspects , such as VM, IP address, port number during runtime at specified time interval. Chapter 5 provides additional insight by investigating key relationships between adaptation time interval and attack time interval. Crouse and Fulp[97] introduce genetic algorithms to proactively evolve vulnerable configurations to eventually find more secure and diverse configurations. Cui and Stolfo[95] propose to proactively inject diverse Symbiotes into existing host programs. ChameleonSoft[43] and TALENT[40] both suggests creating a checkpoint strategy such that poisoned state can easily removed through roll back. TALENT also dynamically changes the live-migration destination platform at randomly chosen time intervals to create a moving target for attackers. Note here live-migration is not the same as VM migration, which can only be done with a homegeneous OS and hardware. TALENT uses OS-level virtualization to sandbox an application and migrate the environment. MEERKATS[39] proposes a high level vision for a Cloud Security Architecture. The proposed Evade component periodically moves data from one location to another randomly to impede the targeted attack. This move is lightweight as it uses

pre-established ciphertext replicas and data is migrated by simply transferring small key shares instead of the actual data. The other proposed components – CSSH (Collaborative Self-healing and Service Hardening), CSIFT (Cross-System Information Flow Tracking), DIGIT (Deceptive Information Generation, Injection and Tracking) and DREME (Diversified Replica Execution and Monitoring Environment) – all reflect the idea of proactive defense. ChameleonSoft and TALENT both introduce proactive diversity to increase system resillience as well as the difficulty of intrusion. More discussion about ChameleonSoft, TALENT and MEERKATS is given in Section 3.5. In addition, related work that has been discussed in Section 3.3.1.2 that refers to proactively diversify corresponding adaptable aspects through artificial diversification all belong to this category. From the time-based perspective, the differences between all these approaches lie in the fact that some proactive strategies are applied at design and implementation phase while others are applied at runtime. In addition, some are only applied once while others repeat many times.

3.4.2 Reactive

Reactive strategies mainly refers to launch adaptations based on environmental information changes, such as exploitation detected, system crash or operation error happens, etc. All these situations can be summarized as event-based.

3.4.2.1 Event-based

Instead of simply rotating VMs at fixed time interval as in SCIT, MAS[31] installs anomaly detection engines in each VM to enable event-driven rotation. Rowe *et al.*[88] model cyber-defense as a control system where the system has Secure Normal, Insecure Normal, Emergency and Restorative states. A system is in an Insecure Normal state if it's operating normally but with indications of attack from IDS or runtime monitors. Such a system needs a preventative control action to pull it back to Secure Normal state, otherwise if the system is partially compromised it will go into an Emergency state and then to a Restorative

state to interrupt services and block malicious activity and use new secure resources to help the system transition back to Secure Normal state. As introduced before, a diverse DHCP protocol example where protocol's state machine has been injected by artificial states to thwart DHCP attack has been demonstrated. Biologically inspired Symbiotic Embedded Machines (SEM)[95] can monitor and react to observed events, such as malware that attempts to hijack the host's execution environment, to prevent it. Colbaugh and Glass[85,86] model an attacker's adaptive learning ability to reverse engineer machine learning enabled spam filters and defenses as repeated, incomplete information games. MEERKATS[39]'s proposed components, such as DMCC (Distributed Monitoring and Crosschecking), are also examples of reactive strategy. ChameleonSoft's[43] failure recovery mechanism also dealing with malicious induced failures by adversaries. TALENT[40]'s live-migration can be triggered not only periodical but also by malicious activities. In addition to a pure random MTD design, Chapter 4 also presents an intelligent MTD design that includes an analysis engine that produces adaptation triggers, based on detected intrusions and failures.

3.4.3 Combined

Based on the previous two categories, it's not hard to see that recent MTD research tends to embrace both proactive and reactive strategies. The following sub category simply includes the research that adopt both strategies.

3.4.3.1 Time + Event

From the time-based and event-based discussion, we see that MAS[31], Rowe *et al.*[88], MEERKATS[39], ChameleonSoft[43], TALENT[40] and the work in this thesis[32,34] all use both proactive and reactive strategies in their overall approach. Although there may be other work also considered both strategies, these examples should be enough to demonstrate the ideas.

3.5 Using the Taxonomy

This taxonomy starts with the attack type and then discusses adaptable aspects, tactics and strategies. Different configuration factors that can be adapted have been introduced from several dimensions, such as network level, OS level, program level, etc. Adaptable aspects provide input to tactics, tactics classify all different kinds of techniques that can be adopted to fulfill the strategy, and the strategy captures the high-level plan to achieve the goal of MTD. These three categories together determine what kinds of attack the MTD system can thwart, which in turn increases system security.

From this taxonomy, we can see that, from network dimension, adaptation applied to IP address, DHCP protocol can be used to constrain attacks, such as worms, scanning/reconnaisance, DDos or DHCP attack. Randomization of OS level aspects, such as memory layout, can be used to defeat buffer overflow attacks. Program level aspects can be used to defeat attacks, such as SQL injection/XSS and web bot intrusion. Machine level aspects, such as VMs, can be used to mitigate the impact of zero day vulnerabilities. Obviously, to thwart as many different attacks as possible, future MTD systems should consider techniques from all of these categories. Based on this observation, this taxonomy can be used to help evaluate the strengths and weakness of an MTD system based on the types of attack that can be prevented. For example, MT6D[20] provides a practical MTD solution at the network level, but it is not effective for attack types at the program level, such as SQL injection/XSS, etc.

Based on this review, most of the existing MTD research has focused on low-level configuration units. However, recent efforts are starting to focus on extending or combining these low-level aspects to provide a more comprehensive MTD solution. These again reflect the trend that future MTD system should properly incorporate the configuration units from different levels as discussed in the taxonomy to provide a holistic solution. The following will summarize several typical research efforts towards this trend; most have already been referenced in previous sections. Nevertheless, MTD research is still in its infancy and significant

effort is required, both theoretically and practically.

Portokalidis[38] proposes a holistic adoption of ISR (instruction-set randomization) across the software stack to prevent the execution of unauthorized binaries and scripts regardless of their origin. This approach requires the programs be randomized with different keys during a user-controlled installation, which effectively combines the benefits of code whitelisting/signing and runtime program integrity. The paper discusses how ISR can be implemented in hardware as well as entirely in software.

Keromytis[39] proposes a novel architecture, MEERKATS, for the cloud environment that enables an environment to constantly change along several dimensions to create an unpredictable target for an adversary. MEERKATS focuses on both software and data, not just protecting, but leveraging them to improve mission resilience. MEERKATS includes many subcomponents. Existing techniques, like instruction set randomization, migration, and N-schedule, have been leveraged and integrated into different subcomponents. While the main goal of the paper was to give the vision of MEERKATS and describe the ways to prototype it, such a system has not been built yet.

Okhravi *et al.*[40] proposes the TALENT (Trusted dynAmic Logical hEterogeNeity sysTem) framework. By using two key ideas, containers and a portable checkpoint compiler, TALENT can create an OS level virtualization environment and migrate running, mission-critical applications across heterogeneous platforms while preserving the state (execution state of the process, open files and sockets) of the application. It is designed to work with general purpose system languages (such as C). In this paper, the threat model assumes that the hypervisor, hardware and OS-level virtualization logic are trusted. The paper shows that after optimization, the environment migration time has been reduced to one second. However, the current prototype is still focused on providing high availability and no guarantees are made about the migrated state not being corrupted.

Azab *et al.*[43] proposes a biologically-inspired defense framework that builds over a novel cell oriented architecture to achieve moving target defense. The key principles are:

1. employ multidimensional software diversities such as functionally-equivalent, behaviorally-different code variants

2. decouple functional roles and runtime role players

3. separate logic, state, and physical resources to induce a spatio-temporal software behavior encryption

It presents a prototype of Behavior Encryption and recovery mechanisms and also studies the provisioned level of security. Although this design seems promising, the key assumption made in the design prototype, to create a checkpoint at each stage of the program, ignores the potential performance impact on complex software systems. In addition, how to correctly implement the rollback to a clean checkpoint and the costs of hot cell shuffling to normal user's access are all unknown.

Carvalho *et al.*[99] describes a human-agent teamwork command and control framework for moving target defenses. The author argues that the reason behind this work is the fact that there are important interdependencies between different defense tools and functionality of critical applications and services. In addition, different operational contexts will require different configurations of these different tools. Thus, it's important to provide a framework that can enhance human system interactions to better support the coordination of different MTD tools.

This taxonomy mainly reviews MTD work in enterprise network context, there may be MTD work from other context that is not covered in this taxonomy.

3.6 Conclusion

This chapter presents a review of related MTD work as well as a taxonomy for these works. From the taxonomy, we see that many different MTD mechanisms have been tried to defeat different attack types. These mechanisms include different combinations of strategies and

tactics that applied to various adaptable aspects . It not only helps one understand the state-of-the-art research in MTD, but also helps point out future research directions. It also underscores the need for a comprehensive theoretical framework for MTD, which is presented in Chapter 6.

Based on the review, almost all of the work that provides analysis of MTD security benefits are qualitative. As discussed in Chapter 1, developing new metrics for moving target defense is a major challenge. Operation and security are both important concerns today. Unfortunately, these two aspects are usually irreconcilable in MTD. Increased security usually brings increased costs. New metrics for MTD should not only help measure and analyze the different mechanism's impact on security, but it should also help MTD designers to make better decisions about system parameter settings such that reasonable trade offs between security and operation can be achieved.

The tension between operation and security in MTD brings another major challenge. This challenge is engineering based and lies in how to sanitize potentially compromised machines while not interrupting normal operations significantly. MT6D does a good job at network-level although IP collisions are still possible. But at the machine-level, ensuring a potentially compromised VM is cleaned is still a major challenge. Although ideas like checkpointing have been proposed in Chameleonsoft[43], no concrete system that evaluates the performance or validates such an approach's applicability is presented. Also as discussed in Section 3.1.2.4, SCIT is not a completely stateless solution and poisoned states can still be disseminated. TALENT[40] also provides a checkpoint mechanism, but leaves how to ensure the migrated virtual machines are not corrupted as future work. Rowe *et al.* points out that selecting the most cost-effective maneuvers between different states, especially the transition from a Restorative state to a Secure Normal state (which essentially represents how to ensure compromised state to be cleaned), are important research challenges.

This thesis focuses on tackling the first major challenge. The next chapter presents a more comprehensive high-level MTD system design that supports this taxonomy. A set of

exploratory simulation and related experiment results will also be discussed to show our first step of investigating the effectiveness of MTD system.

Chapter 4

Exploratory Experiments

In this chapter, a high-level MTD system design framework is presented. Then a simulation, based on this design, is used to investigate the degree to which proactive and random adaptations can decrease an adversary's chance of success. A set of experiments are conducted to examine both a purely random MTD system, as well as an intelligent MTD system, which uses attack indicators to augment adaptation selection. The results show that the attacker's success likelihood can be reduced under such MTD system.

4.1 High-Level System Design

The high-level architecture of a proposed MTD system[33,32,34] that adapts in a purely random fashion, is shown inside the dashed box in Figure 4.1. This system produces random adaptations that do not inhibit correct system operation. The key to making these random adaptations is that they are based on a *Logical Mission Model*, which captures an abstract view of the Physical Network's current configuration along with the functional requirements of the network. The driver is the *Adaption Engine*, which orders random adaptations to the network configuration at random intervals. These adaptations are implemented by the *Configuration Manager*, which controls the configuration of the *Physical Network*. The ar-

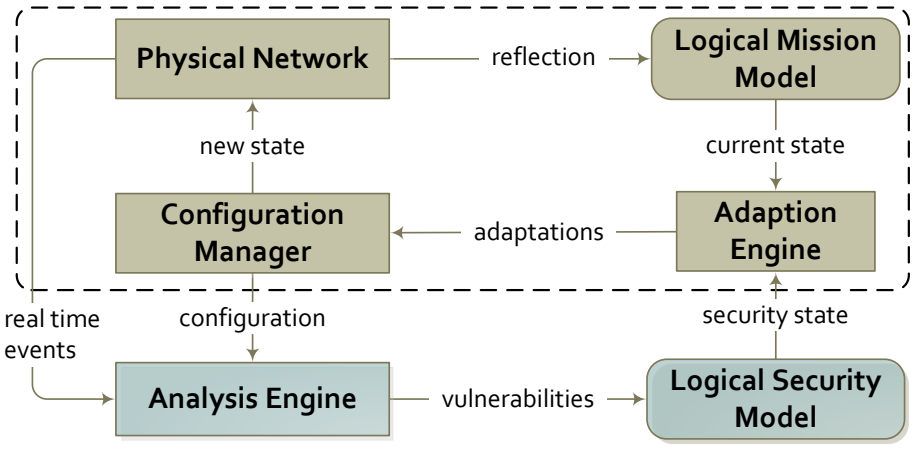


Figure 4.1: Moving Target Defense System Designs

chitecture of an *intelligent* MTD system is the complete system shown in Figure 4.1. The basic operation of the random adaptation remains the same; however, an *Analysis Engine* has been added to take real-time events from the Physical Network and the current configuration from the Configuration Manager to determine possible vulnerabilities and on-going attacks. The *Adaptation Engine* is extended to look at the network's current state along with its security status, as captured in the Logical Security Model. The *Logical Security Model* also consists of two runtime models: a goal model and a model of system vulnerabilities. The goal model captures the system's security goals while the vulnerability model is in the form of a novel *Conservative Attack Graph* (CAG)[33,32,34], which captures both known and unknown system vulnerabilities and how an attacker might move through the system to gain specific privileges.

4.1.1 Resource Mapping System

Chapter 3 discussed the tension between operation and security in MTD systems. An MTD system proactively launches adaptations to increase system security, which at the same time breaks the common operation patterns as appeared in static system. One key problem caused by this is how to ensure each component can still correctly locate and communicate with other components it depends on. Thus, a *resource mapping system*

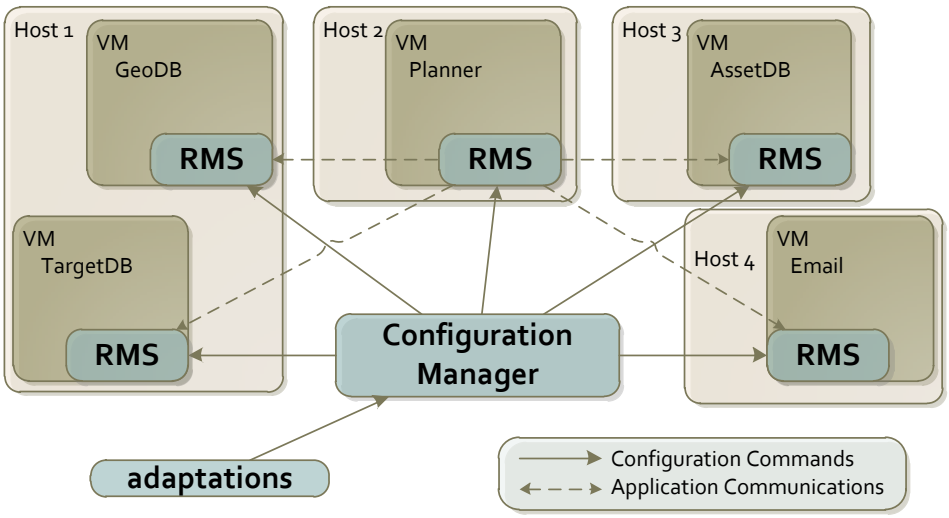


Figure 4.2: RMS System

has been proposed. Ideally, an RMS can be implemented as communication enforcement component that knows the location of all related components in the system. The RMS interacts with the Configuration Manager, to get the up-to-date location information of the various resources.

One possible design of the RMS's use in MTDs is shown in Figure 4.2. Assume the overall mission is executed in a cloud computing environment, then each VM could be configured to have a RMS component to support communications with other roles. Such RMS component can be implemented by making use of existing OS firewalls or, more rigorously, as a dedicated operating system layer. The limitation of this design becomes evident if attackers compromise a critical role or VM. In this case, roles with which the compromised role initiates communications can be easily located and attacked since the compromised role's RMS knows their location. However, the attacker *must* follow the exact communication pattern defined by the Logical Mission Model; communication outside the pre-defined paths can be easily detected. In addition, adaptation can come to the rescue as, eventually, the VM of the compromised role will be refreshed and the attacker will lose any gained privilege.

Other possible solutions include using an existing cloud-computing platform, such as OpenStack's APIs, or Software Defined Network (SDN), such as Openflow, protocol to set

up and update the corresponding communication rules at network-level between various component. These rules could be understood as routing tables maintained by the Configuration Manager.

No matter which solution is being selected, the *purpose* of RMS is to ensure the overall system functionality given a constantly adapting MTD environment. This purpose obviously increases the complexity of normal operational requirements, however, it also restricts the possible intrusion paths used by the attacker, which makes any violations more easily to be detected.

4.1.2 Adaptation Engine

In traditional adaptive systems, the adaptation algorithm would attempt to provide optimal or near optimal configurations[100]. The objective of this component is to produce *effective* configurations that are significantly different in some aspect while limiting the costs of adaptation, or essentially maximizing the entropy of the configurations. Effective configurations must be functionally correct and consistent, and have a tolerable impact on network performance; the physical network and logical mission models are designed to allow the adaptation engine to predict these impacts based on the capabilities of the machines assigned to the system roles.

While the use of intelligent adaptations allows the MTD to react to suspected intrusions instead of simply adapting randomly, using intelligent adaptations in conjunction with purely random adaptations allows the MTD to effectively mitigate unpredicted attacks as well as mask the actions of the intelligent control of system. While the use of random adaptations may not keep an attacker from learning all aspects of how the MTD system responds, it will make the learning process more difficult and time consuming. Additionally, by incorporating responses to suspected intrusions into adaptations, the system can react to suspected intrusions much sooner than a normal intrusion response system since even responses to false positives will leave the system in an operational state with no more

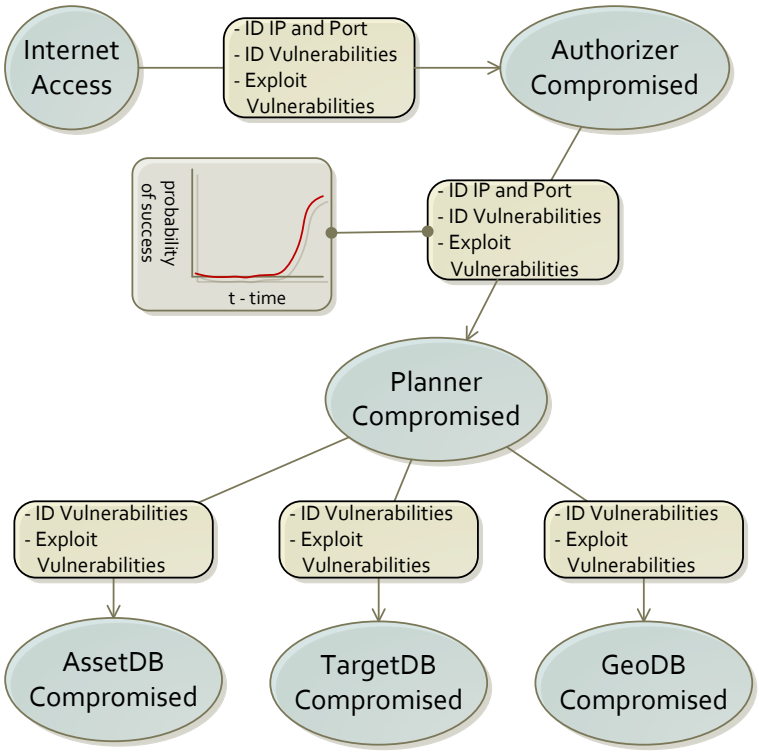


Figure 4.3: Conservative Attack Graph

overhead expended than for a random adaptation.

Since the Adaptation Engine is the main decision making facility for the MTD, it must be able to control the various adaptable configuration unit of the system that range from *Network Level*, *OS Level*, *Service Level* to *Machine Level* as categorized in Chapter 3.

4.1.3 Analysis Engine

The purpose of the Analysis Engine is to infer the most critical vulnerabilities and most likely attack activities so the Adaptation Engine can make intelligent adaptation choices. To analyze the effect of an MTD on computer networks, a *conservative attack graph* (CAG) has been proposed[32,33,34]. The key output of the Analysis Engine is the CAG that captures known and unknown vulnerabilities and indicates paths the attacker might take in attacking the system.

Assuming unknown vulnerabilities in CAG actually reduces the size of the state model

and makes it easier to apply stochastic analysis. Modeling an attacker both gaining and losing knowledge and privileges in CAG invalidates the typical monotonicity assumption[101] of most attack-graph work and requires a state-machine model, rather than traditional dependency attack graphs[102,103,104]. Previous state-enumeration attack graphs[105,106] have encountered scalability challenges when applied to large networks[104]; however, Chapter 5 shows an analytical model that is efficient and scalable to tackle this non-monotonicity challenge of MTD.

As an example, Figure 4.3 shows the CAG for the mission planning scenario used in this thesis. The topology of the conservative attack graph is partially derived from the logical mission model, where dependencies between roles are explicitly captured. In normal operation, valid users log in from the internet through the Authorizer node and interact with the Planner node. The Planner node interacts with the user by using data from the three database nodes, GeoDB, TargetDB, and AssetDB. The only legitimate access paths in the system are (1) from the Internet to the Authorizer, (2) from the Authorizer to the Planner and (3) from the Planner to the three database servers (AssetDB, TargetDB, and GeoDB). The conservative attack graph captures these logical access paths.

The conservative attack graph can also be viewed as a state-transition system. Each arrow is annotated with a label describing the activities involved to move from one state to the next. The effort involved in the activities can be measured in various ways. For example, one can ascribe a success-likelihood to time diagram to indicate how much time it will take the attacker to reach a certain success likelihood for a specific action.

4.2 Simulated MTD Testbeds

To determine if this approach has merit, simulated MTD testbeds were developed to reflect the MTD approach discussed above. The simulated MTD testbeds have three components, the *Defense component*, the *Attack component* and the *Ground Truth component*. In gen-

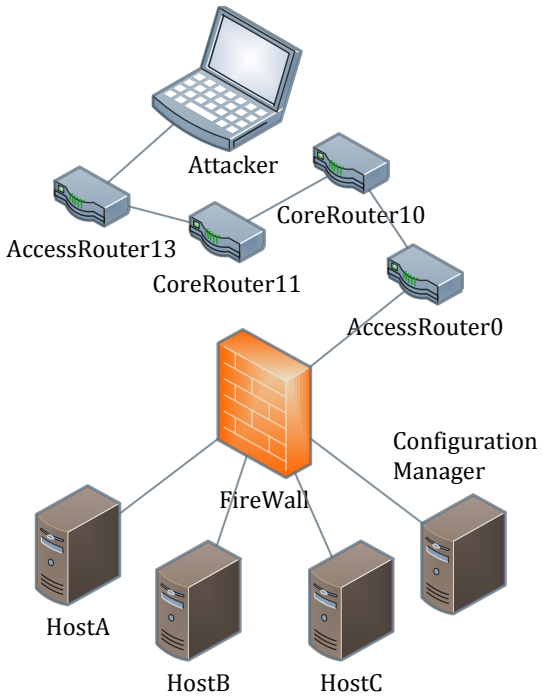


Figure 4.4: Network Topology

eral, *Defense Component* combines the functionality of Configuration Manager, Adaptation Engine and Analysis Engine as described from Figure 4.1. As time elapses, at each adaptation time interval, the Defense component will select a valid state from current configuration space through appropriate decision making option and perform an adequate adaptation to transform the system to the selected configuration state. The updated configuration state will be sent to the *Ground Truth component*. The *Attack component* simulates the attacker and uses the CAG to allow it to know where to attack next to achieve its goal. The nodes of CAG represents active VMs running in the system and the edges represents the communication enforcement between nodes. Values in the CAG edges denote the attacker's probability of intrusion success between nodes assuming both nodes remain static. The *Ground Truth component* maintains the up-to-date ground truth system configuration information. The *Ground Truth component* not only receives updated configuration state informaiton from Defense component after new adaptation has been launched, but also feed these information to Attack component when requested.

To make it clear, lets again use the mission planning scenario example to explain the overall idea. Figure 4.4 shows the network topology of this scenario. The *Defense component* contains three physical machines (hosts) and five active VMs. These five VMs can being assigned to any host to play any of the five roles: Authorizer, Planner, TargetDB, AssetDB, or GeoDB. The *Attack component* simulates the attacker and uses the CAG shown in Figure 4.5 to allow it to know where to attack next to achieve its goal, which is the TargetDB. The edges in the graph (with the exception of the Internet to Authorizer edge) show the valid paths supported by the RMS. In normal operation, valid users log in through the Authorizer node and interact with the Planner node. The Planner node interacts with the user by using data from the three database nodes, GeoDB, TargetDB, and AssetDB. The attacker is assumed to locate at the Internet node and wishes to attack the TargetDB.

Since the only available attack path is to penetrate from Internet to the Authorizer, the Authorizer to the Planner, and then from Planner to TargetDB. The edge values in the CAG denote the attacker's probability of intrusion success between nodes assuming both nodes remain static. As shown, the attacker has a 40% chance of compromising the TargetDB if (1) it has already compromised the Planner and (2) the Configuration Manager does not adapt to either the Planner or the TargetDB during the time step. In a real system, these probabilities would be based on the current probability of unknown and known vulnerabilities of the roles.

Each simulated attack has several steps. First the current configuration state is retrieved from the Ground Truth component to simulate that the attacker somehow obtained the system configuration after conducting reconnaissance. Next, the attack waits Δt time intervals, which simulates the time required to actually launch and finish an attack, then an updated configuration state is retrieved and used to determine whether the attack has succeeded or not. To determine attack success, first a random probability value is generated to check whether it's less than the CAG edge value for the current attack. If it does, the simulation determines if the VMs on either the attacker's current node or the attacked node have been

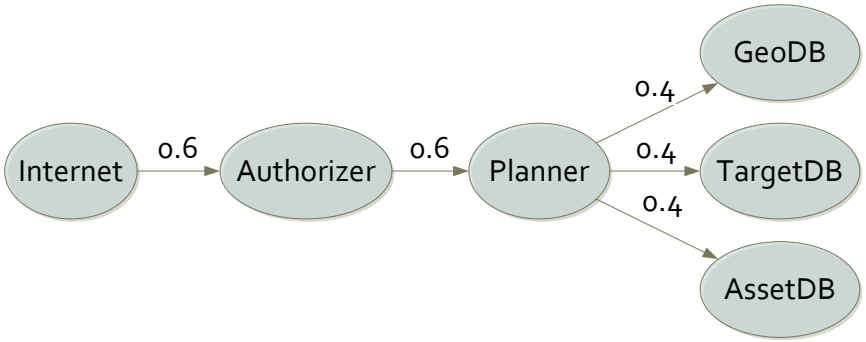


Figure 4.5: Simplified Conservative Attack Graph for Simulation

refreshed; if either of them has been refreshed, the attack fails. If the attacker's current node was the VM that was refreshed, the attacker is pushed back to its previous node. If neither were refreshed, the attack succeeds.

Both the Attacker and the GroundTruth owns a conservative attack graph (CAG). The key difference between this two conservative attack graphs lies in the authenticity about the MTD system configuration. The Attacker's CAG reflects the attacker's belief of the system configuration, which is not necessarily true when facing a constantly changing environment. While the CAG contained in the GroundTruth component reflects the actual and up-to-date system configuration. Comparing the difference between the attacker's belief and the genuine MTD configuration is the key for judging whether the current intrusion is a success or not and update the VM instances the Attacker currently obtained, detailed algorithms will show this in a moment.

In this chapter, several assumptions has been made to simplify the simulations and focus on investigating the key effect of MTD to attack. The next chapter presents a scalable analytical model, which enables the capability to predict these simulation results. In the next step, after a more comprehensive MTD theory has been developed as discussed in Chapter 6, current simulation and analytical model can be extended to investigate more broad attack scenarios with unnecessary assumptions relaxed.

1. Adaptations are applied at a specified time interval and are random in nature (which

is extended in the third simulation to include intelligent adaptation).

2. Adaptations are limited to VM refreshing.

3. All VMs assigned to play a given role have the same configuration except for its ID and IP address.

4. The attacker has the full knowledge of the logical system configuration and once a node is compromised, the attacker can immediately know where the the next node is located to attack.

5. Attacks are restricted to the VMs playing the five roles.

6. The attacker knows immediately when a VM it has compromised has been refreshed.

While these assumptions make the simulation easier, they are also tilted toward the attacker since only the IP address from the *network level* is used, and each VM assigned to play a given role has the same configuration, advanced diversification techniques at *OS level* and *service level* is not considered, which would make compromises more difficult. Also the attacker is assumed to know immediately where the next node is located to attack and when a compromised VM has been refreshed, which typically both costs more time to realize in practice.

In addition, the simulation also reserves important characteristics of intrusion in reality. Although assumed the attacker knows the logical system configuration, that doesn't necessarily mean the attacker knows the exact vulnerability of each machine and how to make use of them, which still requires the attacker's effort to learn and explore, and is indirectly incorporated into the attack time interval. This assumption only means the attacker is aware that there are four VMs, and the Planner is directly reachable from the internet and also can communicate with the other three VMs.

4.2.1 Defender Modeling

In this section, several algorithms will be presented to show how the MTD simulator *Defense component* performs the *moving target defense*.

The *Defense component* in the simulator will maintain the set of roles, R, the current available IP addresses set, I for allocation, the set of physical hosts, H, which can be used to launch new VM instances and current set of tasks, T. A task, $t = \langle r, h, vmid, ip \rangle$, is composed by a role, r, a host, h, an IP address, ip, a VM instance with $vmid$. It represents that a VM instance that has $vmid$ will be launched on host, h, with IP address, ip, to play the role, r. As stated in above assumptions, all VMs assigned to play a given role have the same configuration except its ID and IP address, thus a task is a flat combination of r, $vmid$, ip and h, without the need to create more complex *role* specification and *machine* implementation.

Algorithm 1 adaptation(R, I, H, T, n)

Require: Parameters R: set of roles, I: ip table, H: set of physical hosts, T: set of tasks, n: number of nodes to refresh
1: $T_{new} = generateTask(R, I, H, T, n)$
2: **for each** t_{new} *in* T_{new} **do**
3: $t_{old} = \langle r, i', h', v' \rangle = getTaskFromRole(T, t_{new}.r)$
4: inform h' to shutdown v'
5: inform h to start v
6: $T \leftarrow T - \{t_{old}\}$
7: $T \leftarrow T \bigcup t_{new}$
8: inform $GroundTruth$ to remove t_{old}
9: inform $GroundTruth$ to add t_{new}
10: **end for**

The moving mechanism adopted in the MTD simulator is as follows: for the node set, V, at each time interval, T_r, named as $adaptationInterval$, the simulator randomly pick, n, nodes in V, $n \leq |V|$, shutdown the old VM instances, which assigned to play the corresponding roles, and start new VM instances to replace them. Algorithm 1 shows the overall process of the adaptation. It first calls algorithm 2 which will return a new set of tasks, T_{new}, according to the number of nodes that chosen to refresh in one adaptation. Then in lines 2-10 it iterates through all the new tasks in T_{new} to inform corresponding hosts to shut down old VM instances and launch new VM instances, it will also notify the GroundTruth

component to update these changes. Notice that Algorithm 1 and 2 follow the assumption that adaptations are limited to VM refreshing. As already discussed in Chapter 3, direct machine modification is not suggested as it can not guarantee the potential compromise to this node is totally sanitized.

Algorithm 2 generateTask(R, I, H, T, n)

Require: R: roles, I: IP table, H: physical hosts, T: tasks
1: $T_{new} \leftarrow \emptyset$
2: $count \leftarrow 0$
3: **while** $count < n$ **do**
4: $r \leftarrow$ randomly pick a role r from R
5: $i \leftarrow$ randomly pick an unassigned IP address i from I
6: $h \leftarrow$ randomly pick a host h in H
7: $v \leftarrow$ generate a new ID v for new VM
8: $T_{new} \leftarrow T_{new} \cup \langle r, i, h, v \rangle$
9: $count++$
10: **end while**
11: **return** T_{new}

When the GroundTruth component receives the notification from the Defense component, it will update the mapping information between the role and the actual VM instance's IP and VMID information. Algorithm 3 shows this process. Lines 1-5 of Algorithm 3 correspond to line 9 of Algorithm 1, which informs the GroundTruth component to add a new task, lines 7-11 of Algorithm 3 correspond to line 8 of Algorithm 1, which informs the GroundTruth component to remove the old task.

Algorithm 3 groundTruthProcessMesssage(m)

1: **if** m contains new task information **then**
2: $t_{new} \leftarrow m.getTask()$
3: $node \leftarrow groundTruth.cag.getNode(t_{new}.r.id)$
4: $node.ip \leftarrow t_{new}.i$
5: $node.vmid \leftarrow t_{new}.v$
6: ...
7: **else if** m contains old task information **then**
8: $t_{old} \leftarrow m.getTask()$
9: $node \leftarrow groundTruth.cag.getNode(t_{old}.r.id)$
10: $node.ip \leftarrow null$
11: $node.vmid \leftarrow null$
12: ...
13: **end if**

This section models the *defense* of a moving target system, the next section will show how to model the intrusion.

4.2.2 Attacker Modeling

There are various different kinds of attacks existing today. In this offense modeling section, I will mainly focus on the typical multi-hop remote attack launched from external sites. In this attack type, the attacker attempts to compromise, or make use of, intermediate machines while jumping board to pivoting attack machines that hide deeper. Here we assume each VM instance that plays a role in the defense component has a remote exploitable vulnerability. The RMS enforces the communication paths for the VM instances inside the MTD system. Thus the attacker needs to follow the paths defined in CAG to perform the multi-hop attack.

One complete attack will start from node i, and keep penetration until either the attacker compromises the target node or the attacker totally lost any compromised node other than i due to VM refresh. During this process, the attacker will gain and lose node privileges. Before given the algorithms, two notions called *single step attack* and *complete attack* are introduced to help understand the simulations.

Definition 4.1. *A single step attack represents the intrusion effort involved in compromising a node b, through a directly connected node a. Each single step attack has an associated time interval, T_a, termed as* attack interval *to represent the time cost of this attack.*

Definition 4.2. *A complete attack represents the overall intrusion that the attacker either compromises the target node or totally loses any compromised nodes and gets pushed back to the origin.*

Definition 4.3. *A frontier edge, $f = \langle p, c, t \rangle$, represents a single step attack that will be launched from parent node, p, to compromise child node, c, and finishes at time tick, t. Let E represent all the edges in CAG, then obviously, $\langle p, c \rangle \in E$.*

Algorithm 4 shows the main attack simulation flow. Since the Defender will change the system configuration during runtime, the Attacker is also assumed to be diligent and will keep trying every possible intrusion frontier edge available in the time until it either compromises the target node, or totally gets pushed back to the starting node.

Let N represent the set of nodes currently obtained by the attacker. Let mn represent the most recently acquired node. Let F represent the current frontier edges that the Attackers plan to penetrate based on N. In practice, F is a priority queue of f, $f \in F$, ordered by t.

Not every attack will be successful, thus there will be a success probability, p_r, associated with each edge. This probability depends on the vulnerability the child node has.

As shown in algorithm 4, there will be $maxTimesOfAttack$ number of complete attacks performed. In each complete attack, the GroundTruth history CAG list, $G = \{g_1, g_2, \ldots, g_{tmax}\}$, will be regenerated. Here $tmax$ is the maximum time ticks the simulator needs to run, $tmax$ should be sufficiently large, such that it is enough for a complete attack to finish (either compromise the target node or totally get pushed back to origin). Notice that for two complete attacks, the overall time spent on each complete attack might vary due to the uncertainty involved in adaptation as well as each single step attack's success likelihood. In time range $[1, tmax]$, every *adaptation interval*, T_r, there will be an adaptation occurred. Thus in $\{g_1, g_2, \ldots, g_{tmax}\}$, the CAG in $\{g_1, g_2, \ldots, g_{T_r-1}\}$ will be the same. Similarly, the CAGs in $\{g_{T_r}, g_{T_r+1}, \ldots, g_{2*T_r-1}\}$ are also the same, and the rest can be analysed in the same manner.

In Algorithm 4, Lines 3-19 will initialize the acquired nodes set, N, and corresponding frontier edge set, F. Notice in line 5, the algorithm randomly picks a start attack time tick between $[0, T_r]$, this actually reflects the real world situation that, when the intrusion is being launched, the attacker does not necessarily know how long there will be another adaptation happens. Lines 20-48 perform the main loop in one complete attack as previously described. Line 23 calls Algorithm 5 to judge whether the current frontier edge can be successfully penetrated or not. Algorithm 5 first calls Algorithm 6 to compare the node in N with the corresponding node in GroundTruth to check whether they have the same VM instance's identifier *vmid*. If not then it means the acquired node has been refreshed, thus the attacker loses the privilege and this node needs to be removed from N, also all corresponding frontier edges that start from this node need to be removed from F. Then

Algorithm 4 attackSimulation

```
1: for i = 0 → maxTimesOfAttack do
2:     regenerate the groundtruth
3:     N ← ∅
4:     F ← ∅
5:     ct ← randomly pick an integer in [0, reorganizationInterval]          ▷ Simulate the start time of attack
6:     mn ← node i in attacker's cag
7:     N ← N ∪ {mn}
8:     E′ ← get all edges start from mn
9:     for each ⟨p, c⟩ in E′ do
10:        if c ∉ N then
11:            ctcag ← get cag at ct from ground truth
12:            ip ← ctcag.getNode(c.getID()).ip
13:            vmid ← ctcag.getNode(c.getID()).vmid
14:            c.ip ← ip
15:            c.vmid ← vmid
16:            f ← ⟨mn, c, ct + attackInterval⟩
17:            F ← F ∪ {f}
18:        end if
19:    end for
20:    while mn.id ≠ targetID or (|N| > 1 and |F| > 0) do
21:        f ← poll a ticked edge from priority queue F
22:        ct ← f.t
23:        b ← singleStepAttack(f)                                          ▷ Call Algorithm 5
24:        if b then
25:            N ← N ∪ {f.c}
26:            mn ← f.c
27:        end if
28:        for each n in N do
29:            if n is not start node i then
30:                E″ ← get edges start from n
31:                for each e in E″ do
32:                    if e.c ∉ N then
33:                        for each f in F do
34:                            if not e ⊂ f then
35:                                cag ← get cag at ct from ground truth
36:                                ip ← cag.getNode(c.getID()).ip
37:                                id ← cag.getNode(c.getID()).vmid
38:                                c.ip ← ip
39:                                c.id ← id
40:                                f ← ⟨mn, c, ct + attackInterval⟩
41:                                F ← F ∪ {f}
42:                            end if
43:                        end for
44:                    end if
45:                end for
46:            end if
47:        end for
48:    end while
49: end for
```

Algorithm 5 singleStepAttack(f)

refreshPushBackCheck($f.t$) ▷ Call Algorithm 6
$cag \leftarrow$ get cag at $f.t$ from ground truth
$d \leftarrow$ randomly generate a double value between [0,1]
$e \leftarrow$ get edge from $f.p$ to $f.c$ in $attackerCag$
if $d > e.prob$ **then**
    return false
end if
$beliefIp \leftarrow f.c.ip$
$beliefVmid \leftarrow f.c.vmid$
$trueIp \leftarrow cag.getNode(f.c.id).ip$
$trueVmid \leftarrow cag.getNode(f.c.id).vmid$
if $beliefVmid \neq trueVmid$ **then**
    return false
else if $beliefVmid == trueVmid$ and $beliefIp \neq trueIp$ **then**
    if $\langle f.p, f.c\rangle \in cag.E$ and $f.p \in N$ **then**
        return true
    else
        return false
    end if
else
    if $f.p \in N$ **then**
        return true
    else
        return false
    end if
end if

Algorithm 6 refreshPushBackCheck(t)

Require: Parameter t: the time tick to check
$cag \leftarrow$ get cag at tick t from ground truth
for each n *in* N **do**
    if n is not start node i **then**
        $vmid \leftarrow cag.getNode(n.id).vmid$
        if $vmid \neq n.vmid$ **then**
            $N \leftarrow N - \{n\}$
            $E' \leftarrow$ get all edges start from n in $attackerCag$
            for each $\langle p, c\rangle$ *in* E' **do**
                for each f *in* F **do**
                    if $\langle p, c\rangle \subset f$ and $t == f.t$ **then**
                        $F \leftarrow F - \{f\}$
                    end if
                end for
            end for
        end if
    end if
end for

Lines 3-7 of Algorithm 5 mimic the real world situation that not every exploit will be success. If lines 3-7 of Algorithm 5 pass the probability check, then lines 8-25 perform a last check to judge whether the current single step attack was a success or not. The main idea is that at the finish time $f.t$ of current single step attack, only when node $f.p$ is still in acquired node set N and the belief IP and VMID in node $f.c$ are the same as the one in GroundTruth, this single step can be considered as success.

The algorithms in offense and defense modeling are general in terms that they can be applied to different graph structures, such as trees, directed and undirected graphs.

4.2.3 Simulations and Results

Three high-level simulations were developed to reflect the MTD approach discussed above. The first simulation, which we call the RMS-only Simulation, was developed to evaluate the effectiveness of our MTD approach using an existing network simulator called NeSSi2, an open-source, distributed and discrete-event based network security simulator with support for complex application-level scenarios based on a simulated TCP/IP protocol stack[107]. In this simulation, we assumed the attacker had full knowledge of the logical system configuration and only attacked through the restricted communication path. In the second and third simulations (which are termed as *broad attack simulations*), while the attacker still has full knowledge of the logical system configuration, the attacker also attempts attacks between nodes not directly connected via enforced communication path. For these broad attack simulations, we developed a unique event-driven simulator based on the experience of building our first NeSSi2-based simulator. In the first two simulations, a basic MTD system that adapted randomly at a specified time interval was implemented. However, in the last simulation, we upgraded the MTD to an intelligent MTD system that could detect on-going intrusions when attacks were attempted outside the RMS system.

4.2.3.1 RMS-only Attack Simulation System

The three main components of the RMS-only testbed include the Defense component, the Attack component and the Ground Truth component. The *Defense component* contains the Configuration Manager, three physical machines (hosts) and five active VMs. These five VMs can be assigned to any hosts to play any of the five roles: Authorizer, Planner, TargetDB, AssetDB, or GeoDB. The Configuration Manager is the core of the Defense component and combines the functionality of the Configuration Manager and the Adaptation Engine from Figure 4.1. At each adaptation time interval, the Configuration Manager selects an adaptation by creating a *new* task, $t_{new} = \{role, host, vmid, ip\}$, by (1) randomly picking a role, (2) randomly picking a host, (3) generating a new unique VM ID, and (4) randomly picking an unassigned IP address. The Configuration Manager finds the associated old task, $t_{old} = \{role, host', vmid', ip'\}$, within its set of existing tasks, T, by matching role names. It then informs the old task's current host, $host'$, to shut down the $vmid'$ VM and tells the new host, $host$, to start up a new VM at address ip to play the $role$. Finally, the Configuration Manager updates the Ground Truth component with the current configuration.

The *Attack component* simulates the attacker and uses the CAG shown in Figure 4.5 to allow it to know exactly where to attack to achieve its goal, the TargetDB. Since the only available attack path is to penetrate from the Internet to the Authorizer, the Authorizer to the Planner, and then from Planner to TargetDB. The edge values in the CAG denote the attacker's probability of intrusion success between nodes assuming both nodes remain static. As shown, the attacker has a 40% chance of compromising the TargetDB if (1) it has already compromised the Planner and (2) the Configuration Manager does not adapt either the Planner or the TargetDB during the time step. In a real system, these probabilities would be based on the current probability of unknown and known vulnerabilities of the roles.

Each simulated attack has several steps. First the current CAG is retrieved from the Ground Truth component. Next, after waiting Δt time intervals (which simulates the time

required to launch an attack), an updated version of the CAG is retrieved and used to determine whether the attack has succeeded or not. To determine attack success, we first generate a random value and check to see if it's less than the CAG edge value for the current attack. If it does, the simulation determines if the VMs on either the attacker's current node or the attacked node have been refreshed; if either of them has been refreshed, the attack fails. If the attacker's current node was the VM that was refreshed, the attacker is pushed back to its previous node. If neither were refreshed, the attack succeeds.

The *Ground Truth component* maintains the current CAG. The Ground Truth component receives adaptation information from Configuration Manager and updates the CAG as required. It also supplies the current CAG to the Attack component when requested. The Attack component, Defense component, and Ground Truth component are implemented as NeSSi2 components along with the three host resources: hostA, hostB, and hostC. These six components are loaded onto the corresponding nodes as shown in Figure 4.4.

4.2.3.2 RMS-only Attack Simulation Results

We conducted two different experiments (denoted 1a and 1b) to see how the frequency of system adaptation would impact attack success. Within each experiment, we included a control scenario where no adaptation occurred. Attacks were launched from the Internet towards the TargetDB. Each attack consisted of *single step attacks* from the Internet to the Authorizer, the Authorizer to the Planner, and from the Planner to the TargetDB. Once the TargetDB was compromised, the attack was counted as successful. If a single step attack failed, the attacker remained at its current VM and retried the attack until successful or the MTD system refreshed the VM. In each experiment, we performed 1000 single step attacks with a fixed Δt between each single step attack of 100 time intervals. We ran the 1000 single step attacks against an MTD system using 5 different time intervals (20, 50, 100, 200 and ∞) between each adaptation. Note that an ∞ adaptation interval corresponds to a static system.

In the experiment 1a, we assumed that in order to stop a single step attack from succeeding, either a randomly generated probability value is greater than the single step edge associated probability in CAG, or the MTD must refresh either the node under attack or the node from which the attack was launched during the attack (100 time intervals). Therefore, if there was a single step attack occurring from the Planner to the TargetDB, it could be stopped if either a randomly generated probability value is greater than 0.4, or the Planner, or TargetDB roles were refreshed by the MTD system during the attack. However, the attacker would remain on the network unless the actual VM it was residing on was refreshed. The green bars in Figure 4.6 show the ability of the MTD to deter a successful attack from the Internet through the Authorizer and the Planner to the TargetDB. When the configuration is static, the number of successful attacks (of each round of 1000 single step attacks) is 183. Essentially, since no refreshing was going on, this is the maximum number of successful attacks given the probabilities of single step attack success. Once the MTD system is activated, the number of successful attacks decreases. With an adaptation interval of 200, the number of successful attacks is reduced to 123, while an interval of 100 reduces it to 57, and an interval of 20 eliminates all successful attacks against the TargetDB. Figure 4.6 clearly shows that as the adaptation interval is reduced, the effect of the MTD defense is clearly visible.

In the experiment 1b, we assumed that in order to stop an attack from succeeding, the MTD could refresh any node on the path to the node being attacked during the attack (100 time intervals). Thus in this version, if there was a single step attack occurring from the Planner to the TargetDB, it could be stopped if either the Authorizer, Planner, or TargetDB roles were refreshed during the attack. The gold bars in Figure 4.6 shows the ability of the MTD to deter a completed attack from the Internet through the Authorizer and the Planner to the TargetDB. When the configuration is static, the number of completed attacks (out of 1000) is 168, while an adaptation interval of 200 reduces that number to 107, 100 reduces it to 41, and an adaptation interval of 20 again eliminates all successful attacks against

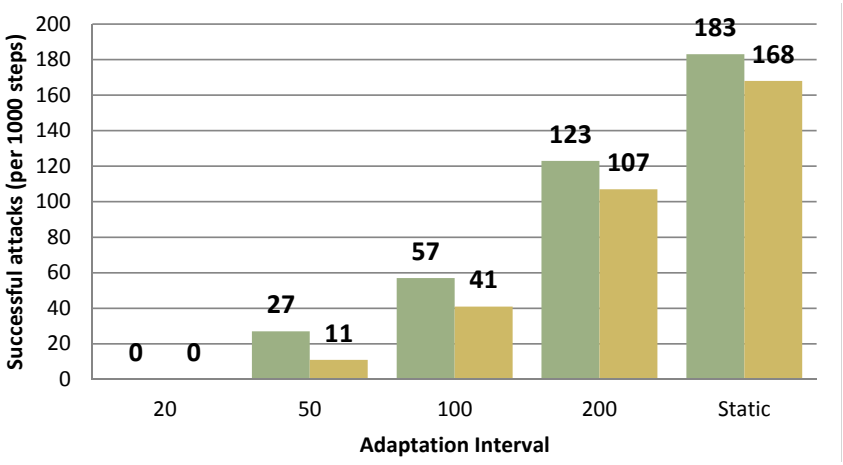


Figure 4.6: Attack Success Against TargetDB (experiment 1a are shown by green while experiment 1b are shown by gold bars)

the TargetDB. Again, Figure 4.6 clearly shows that as the adaptation interval is reduced, the effect of the MTD defense is obvious. Experiment 1a and 1b actually represents two different types of attacks. For 1a, the attack can be considered as worms, which can infect machines by the program itself, while for 1b, it's corresponding to the situation where the attacker tries to compromise the target machine through the intermediate nodes that are compromised as springboard.

4.2.3.3 Broad Attack Simulation System

In the broad attack simulation, the TargetDB is again the attacker's goal. However, we assume a more aggressive attacker who automatically attacks each available node in the network from each compromised VM using either the RMS or by guessing an address and port of an available node. Thus, the attacker is not limited to the RMS routes and the attack routes form a completely bidirectionally connected graph (except for the Internet node) as shown in Figure 4.7. However, since we assume that the RMS is designed to *not* respond to standard requests for mapping information, this eliminates the attacker's ability to automatically map the address space.

The probabilities associated with each attack depend on the node from which the attack

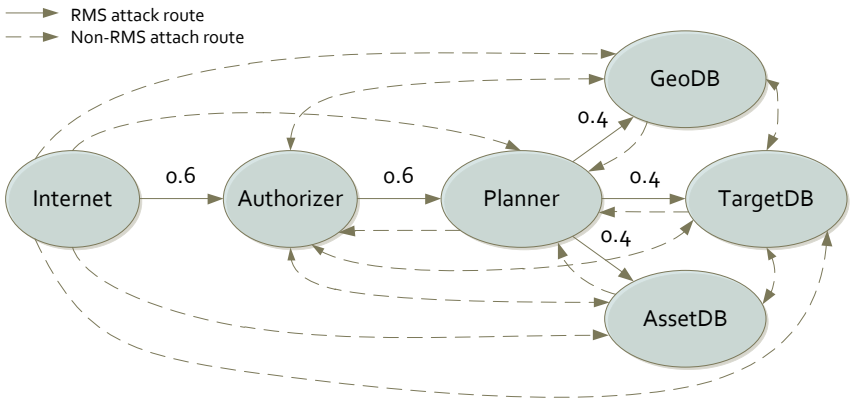


Figure 4.7: Attack Success Probabilities in Broad Attack Simulation

originates and the node being attacked. All attacks along the RMS maintain their probabilities as shown in Figure 4.7. However, the dashed lines, which denote attacks outside the RMS, have a much lower probability due to the fact that the attacker must guess the appropriate port for the attack to even have a chance to succeed. Therefore, each dashed line has an attack success probability of $p/65,536$ where p is the probability of successfully attacking that node through the RMS and 65536 is the port number space. Thus, all attacks against the TargetDB from any node but the Planner would have a $0.4/65,536$ probability of success. While this might seem like a very low probability, we believe that it is actually the upper bound for such an attack. Since the VMs addresses are being modified over time, the attacker will also have to guess the VM address. However, since it is hard to determine the specific range over which the addresses be assigned, we assume the attacker can guess that in some way (again giving the benefit to the attacker as opposed to the MTD system).

The simulation starts with the attacker at the Internet node. From the Internet node, the attacker attempts to attack each node in the network. The success of each attack is determined based on the probability of success of the attack and whether either the node being attacked or the node from which the attack originated was refreshed during the attack. If any of the attacks were successful, the newly compromised nodes are used to mount new attacks. Again, we assume we try to attack all uncompromised nodes from each newly

compromised node. This process continues until the TargetDB becomes compromised, or the attacker has no compromised nodes in the network (other than the Internet).

4.2.3.4 Broad Attack Simulation Results

We conducted 1000 runs (as opposed to 1000 single step attacks used in the RMS only experiments) of the broad attack simulation against various frequencies of MTD adaptation to determine its impact against attack success. Each run consisted of a sequence of attacks starting with the initial attack from the Internet to the Authorizer node and continuing until either (1) the attacker did not have access to a compromised node in the network or (2) the attacker successfully compromised the TargetDB. As with the previous experiments, we included a *static* control scenario where no adaptation occurred. In each experiment, we again assumed a fixed time interval $\Delta t = 100$ between each single step attack, and we ran the 1000 runs using 5 different adaptation intervals (20, 50, 100, 200 and ∞).

Figure 4.8 shows the ability of the MTD to deter an attack from the Internet through the network to the TargetDB. When the configuration is static, the number of completed attacks (out of 1000) is 588, which is close the expected 60% rate given that the probability of compromising the Authorizer node from the Internet is 0.6. This is due to the fact that if the attacker compromised the Authorizer node on the first attack, with a static network, the attacker will remain on the Authorizer node attacking various network nodes until the TargetDB is eventually compromised. We also noted that no attacks outside the RMS actually succeeded, which was expected given the extremely low probability of success. When we introduced our random adaptations, we found that an adaptation interval of 200 reduced the number of successful attacks against the TargetDB to 421, an adaptation interval of 100 reduced that number to 57, an adaptation interval of 50 allowed only 24 successful attacks, and an adaptation interval of 20 totally eliminated the ability of the attacker to compromise the TargetDB. Once again, Figure 4.8 clearly shows that as the adaptation interval is reduced, the effect of the MTD defense is clearly visible.

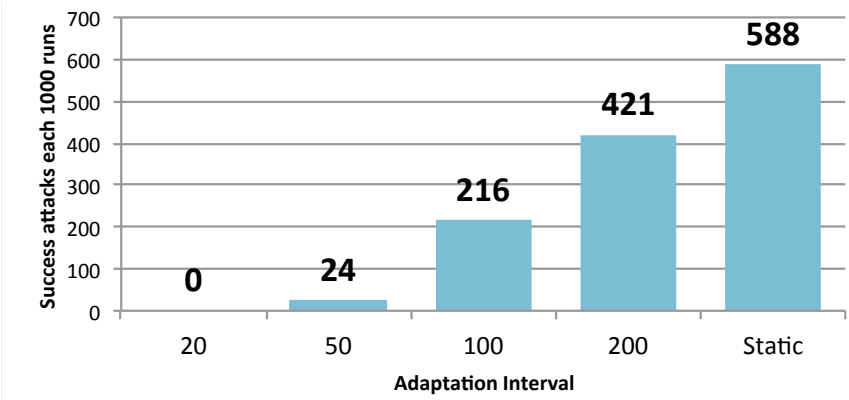


Figure 4.8: Attack Success Against TargetDB for Broad Attack Simulation Against Simple MTD

4.2.3.5 Intelligent MTD Simulation System

To help determine the effect of an intelligent MTD system, we again used our broad attack simulation where the attacker attempts to compromise the TargetDB. In fact, the experimental setup was the same as for the broad attack simulation presented above with one exception. To simulate an intelligent MTD system, we assumed that whenever the attacker attempted an attack outside the RMS, that such an attack could trigger an alert based on some probability of detection, p_d. Since the RMS is set up to allow only communication from known nodes on exactly one port, we believe the implementation of such detectors would be both practical and efficient. When detected, alerts would be sent directly to the Adaptation Engine, which would request that Configuration Manager immediately refresh the VM from which the detected attack originated. In addition, random adaptations continued to occur at the same predetermined intervals Δt as used in the previous experiments.

4.2.3.6 Intelligent MTD Simulation Results

The result of the intelligent MTD simulation is shown in Figure 4.9; note that the graph is logarithmic to show proper detail. Since the attacker indiscriminately attacks all nodes in the network without necessarily attempting to go through the RMS system, thus raising many alerts, the success rate of the attacker is reduced significantly. At a 100% probability

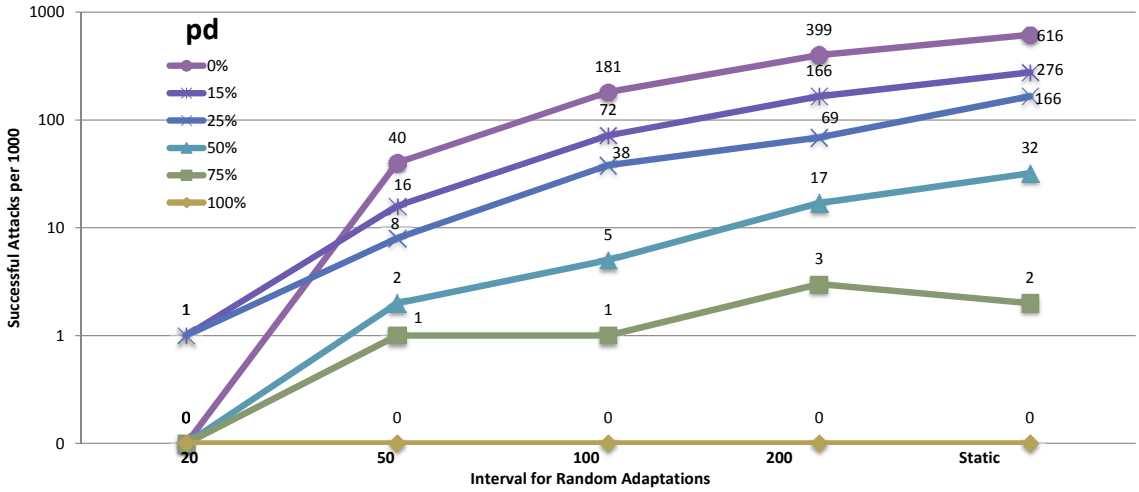


Figure 4.9: Attack Success Against TargetDB for Broad Attack Simulation Against Intelligent MTD

of detection, the attacker is always immediately detected and removed from the system, thus the attack success rate is 0%. However, even with lower p_d values, the reduction in attack success is significant. Even in the static case, with a p_d of 50%, the number of successful attacks is reduced from 616 (61.2%) to 32 (3.2%). We believe this shows the power of using an RMS with an intelligent MTD system. The RMS minimizes the attack surface to such a degree that attacks outside the RMS are easily detected and significantly decreases the attacker's likelihood of success.

When compared to the attack success rate of the simple MTD (shown by the $p_d = 0\%$ line in Figure 4.9), the intelligent MTD performs significantly better. A slight data anomaly is evident at adaptation interval 20 when p_d is 15% and 25%; there is one successful attack while there are none when $p_d = 0\%$. With more runs, we believe the data would normalized. This does show that while the probability is extremely low, the attacker can succeed. While not conclusive, this clearly shows the need for further research into the costs and benefits of intelligent MTD systems.

4.3 Conclusion

This chapter presents a high-level moving target defense system design. Based on this design, simulated MTD testbeds are developed with the core algorithms given. Then several simulation experiments are conducted to study the effects of randomly adapting one aspect of the system (role to VM mapping) in reducing attacker's success likelihood. The results show a reduction in attack success as the rate of adaptation increased. In addition, more simulations also have been conducted by incorporating the knowledge of when and where to adapt based on detecting attacks outside the RMS. Even with less than perfect detectors, significant improvements in network security can be made.

Simulations help us understand the effect of MTDs with intuitive results, however, simulations cost time and does not mathematically reveal the hidden relationships between key involved parameters, such as T_a and T_r. If we could have a model that captures these relationships clearly, then based on that, more useful predictions can be made to help the settings of various MTD system operational parameters. Targeted at this goal, the next chapter will present an scalable analytical model.

Chapter 5

A Scalable Analytical Model

5.1 Motivation

As shown in Chapter 4, more frequent system adaptation and more accurate intrusion detection leads to a reduced likelihood of intrusion. This brings up the question of whether an underlying model exists that can be used to predict the intrusion success likelihood. Models like this will be invaluable for MTD systems because they can explicitly reveal the key relationships between the important parameters involved and is the key to help understanding why and how adaptation can improve security.

In this chapter, offensive and defensive modeling is used along with simulations to create an analytical model that can answer these questions. The event-driven simulation from the last chapter was extended and used to investigate how well the results predicted by this proposed analytical model match the simulation results.

5.1.1 Extended Simulation

The extended event-driven simulator used in this chapter still has three components : the Attack component, the Defense component and the GroundTruth component. In this extended event-driven simulator, a more complex scenario is considered and can be represented

by the *conservative attack graph* as shown in Figure 5.1. As we see, there are eight roles and each is assigned to a VM to play it. The intrusion path could be from as long as four hops from the original attack node to the target node, such as the path $i \to a \to c \to f \to h$, to as short as only one hop, such as the path $i \to a$. With this setting, experiments can be set up to evaluate the different impact of MTD adaptation to these varied length attacks.

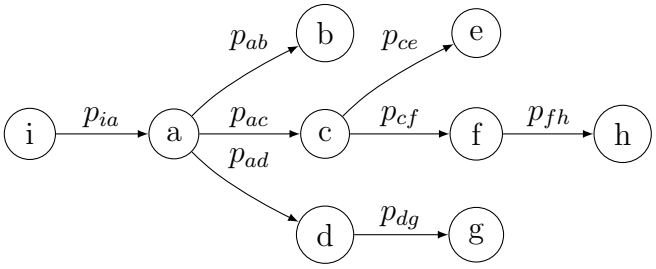


Figure 5.1: sample conservative attack graph

The CAG is a graph, $g = (V, E)$, where V is the set of vertices and E is the set of edges in this graph. Figure 5.1 shows a sample conservative attack graph. In this CAG, node set, $V = \{a, b, c, d, e, f, g\}$, are roles that are defined in the Defender. Each role specifies the configuration. During runtime, when a role is played by a VM instance, the VM instance implements these specified configurations, such as a unique ID and IP address. The configurations are not necessarily restricted to just VMID and IP address, they can be extended to all the identified adaptable configuration units discussed in Chapter 3. The edge set $E = \{i \to a, a \to b, a \to c, a \to d, c \to e, c \to f, d \to g, f \to h\}$, represents the communication rules between different roles defined by the RMS and contains the success likelihood of intrusion when both nodes connected by this edge are static. The intrusion starts from node i where the attacker is located.

The GroundTruth component maintains a list of CAGs, $G = \{g_1, g_2, g_3, \ldots, g_t\}$, where t represents the time tick. These graphs contain key configuration information that is used to compare the attacker's belief to judge whether the intrusion is successful or not.

The first set of experiments, which focused on investigating the impact of pure random adaptation to the intrusion success likelihood, is based on the *broad attack simulation* as

discussed in chapter 4. This means that the Attacker owned CAG not only contains the topology as the GroundTruth CAG defined it, but it also adds all the potential intrusion paths from the current compromised node to all other nodes (except i). These added paths could deviate the path enforced by RMS. However, based on the experience of the *broad attack simulation* experiments conducted in Chapter 4, the probability of succesfully following these deviated edges is extremely low and can be neglected. Thus, in this set of experiments, we drop these deviated edges in the Attacker's CAG, which gives the Attacker's CAG the same topology as the GroundTruth's CAG. The attacker will keep attacking until it either compromises the target or it gets totally pushed back to the original node i.

5.2 The Model

Based on the algorithms and MTD simulation presented in Chapter 4, one straightforward way to estimate the success likelihood of an intrusion to a target node is to use Monte Carlo simulation. One can run multiple complete attacks and count the fraction of times that the Attacker successfully acquires the target node. However, simulation takes time, especially when the size of CAG becomes large and the target node is deep in the system. This method is good for offline precalculation if we know all the possible network configurations, but it's not suitable for online estimation of success likelihood when new network configurations are deployed.

In this section, an analytical model is presented based on the attacker and defender modeling and can be used to estimate the intrusion success likelihood. This model not only has important characteristics, such as efficiency and scalability, but also helps to clearly reveal the relationships between the key parameters involved in an MTD system. Ideally, this model will provide deeper understanding about the key MTD settings, such as diversification, randomization and time as discussed in Chapter 3, that impact various types of attacks against an MTD system. As we will see, the insights gained facilitates the develop-

ment of a theory of MTD in Chapter 6, which will greatly benefit and guide MTD system design and development.

5.2.1 Suitable Structure

The analytical model presented in this section is suitable for graph structures where no cycles exist. However, for cyclic graphs, we can take advantage of techniques for unfolding cyclical graphs into non-cyclical graphs[108] that allow this analytical model to still apply. Thus it can be easily applied to large number (hundreds or thousands) of computing nodes that are common in today's enterprise network environment.

5.2.2 Model Parameters

This section defines the input and output parameters as well as the assumptions that are used in our mathematical analysis model.

Input parameters:

- *A conservative attack graph*

  The conservative attack graph not only reflects the role relationships in the system, but also contains security information such as vulnerability and success likelihood for static intrusion. This graph should follow the structure discussed in the previous section. Figure 5.1 shows the sample CAG that used to conduct the simulations. Note that the Attacker will launch intrusion from node i, and this node doesn't belong to the Defender. The edges in this CAG define the communication rules between the roles (except $i \rightarrow a$). When the system runs, each role will be played by a VM instance. The actual movement happens when we change the mapping between the roles and VMs.

- T_a : *attack interval*

  The time required for an attack along an edge from one node to an adjacent node. For

now, we assume each edge in a CAG has the same attack interval. This assumption is relaxed later.

- T_r : *adaptation interval*

  The time interval between each system adaptation. For now, we assume this value will not change once system starts running. This assumption is relaxed later.

- n : *number of active VMs in the system*

  Each virtual machine is assigned to a specific role and is involved in sytem adaptation.

- k : *number of VMs refreshed in each adaptation*

  Each adaptation randomly picks k out of n VMs for refreshing.

- p_{ij} : *success likelihood of intrusion*

  p_{ij} reflects the success likelihood of intrusion from node i to node j assuming no adaptation occurs on both nodes, which is a typical static situation. For example, in Figure 5.1, p_{ia} represents the success likelihood of intrusion from node i to the node a when neither node gets refreshed. And p_{ab} represents the success likelihood of intrusion from node a to node b when neither a and b gets refreshed.

Output parameters:

- $P_{success\_x}$

  Using the input parameters, the model predicts the success likelihood of intrusion from node i to each other node x in the given CAG. For example, in Figure 5.1, $P_{success\_a}$ represents the success likelihood of intrusion from node i to node a. And $P_{success\_c}$ represents the success likelihood of intrusion from the node i to node c.

5.2.3 Challenges

The key challenge in defining an analytical model for MTD systems lies in its nonmonotonicity. In the MTD context, the typical asymmetrical *time* advantage of the attacker

(such as having enough *time* to perform reconnaissance, choosing the best *time* to launch an intrusion and maintaining obtained privileges for a long *time* without being discovered) is eliminated. In an MTD system, the attacker might constantly lose any privileges gained during an attack due to adaptation and be required to regain them to achieve the final target. MTD systems not only add time pressure for an attacker but also can reduce the impact of a compromise by proactively adapting to eliminate the compromise. Thus, when facing an MTD system, an attacker will have to be diligent if they want to succeed. However, the defender's consistent adaptation will frustrate them again and again and hopefully force them give up. Thus, modeling a diligent attacker that works inexhaustibly until either compromise or elimination from the system is required. Inspired by this, the analytical model captures the extreme situation where the attacker keeps attacking the next node as long as he/she can stay in the system. The intrusion success likelihood estimated under such situation should capture the upper bound of $P_{success}$.

The first attempt to get the output parameters was to create a Markov Chain and derive the success likelihood when this Markov Chain converges. One way to model the Markov Chain is to consider the set of nodes currently obtained by the attacker as a Markov Chain state. Unfortunately, it turns out that in this approach, the state space is exponential when the CAG size becomes large due to the nonmonotonicity of MTD system. In addition, the state transition probabilities become extremely difficult to derive (if not impossible) as the transition possibility can be from one state that contains several compromised nodes (for example, 5) to another state that can contain any number of compromised nodes (for example, 0-8). An alternative approach is to consider each node as a state. The forward transition from one state to a deeper node represents the probability that the attacker can successfully obtain the next node. The self transition represents the probability the attacker can stay at current node. The backward transition from a deeper node state to a previously obtained node represents the probability the attacker gets pushed back to previous node due to adaptation. However, this approach also turns out to be poor because as the CAG size

becomes large, the backward transition becomes extremely hard to derive due to adaptation. The attacker can be pushed back to any previously obtained node in one or more adaptation. Also the nonmonotonicity totally breaks the important assumption that exists in Markov Chain model that state X_i only depends on X_{i-1} and is independent of all previous states of X_{i-1}.

Nevertheless, these attempts proved to be benefitial in that they provided the key insight that the analytical model should only consider forward and self transitions, with backward transitions being ignored. Although we do not directly consider backward transitions, they can be modeled indirectly by using forward and self transitions as described below.

5.2.4 An Original Model

In this section, the original analytical model is presented. As an example, we use the system in Figure 5.1 and assume that the attacker is trying to compromise node b from node a. The following analysis for b can be applied to node c and d similarly. The key idea of this original model is illustrated by Figure 5.2. The values of p_1, p_2, p_3 represent the transition probabilities from i to a, from a to a and from a to b under the condition that adaptations may occur during the attack.

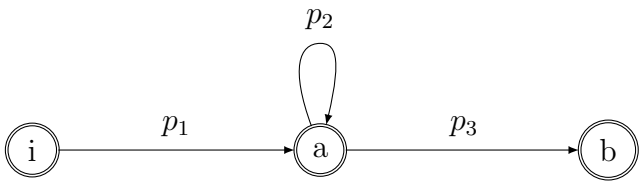


Figure 5.2: Original Transition Model for $i \rightarrow a \rightarrow b$

The forward transition probability from node i to node a is obtained as follows. The probability that node a gets refreshed in one adaptation is $\frac{k}{n}$, thus the probability of it not being refreshed in one adaptation is $1 - \frac{k}{n}$. The intrusion from i to a takes time interval T_a and during this time period, $\frac{T_a}{T_r}$ adaptations occur. Each adaptation is independent of each other and during T_a, the probability that node a does not get refreshed is $(1 - \frac{k}{n})^{\frac{T_a}{T_r}}$. In the

static situation, the transition probability from i to a is p_{ia}, thus the final success likelihood from i to a (which is p_1) can be derived as shown in Equation (5.1).

For the self transition from a to a, there are two possible ways for an attacker to remain at the node a during a time period T_a. First, the attacker may fail to penetrate from a to b with probability $(1 - p_{ab})$ assuming node a doesn't get refreshed during this time period, which gives us the probability $(1 - p_{ab}) \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}}$. The second probability is that node a does not get refreshed, the attacker successfully follows the edge from a to b with probability p_{ab}, but node b does get refreshed during time period T_a. This has the probability of $(1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times p_{ab} \times (1 - (1 - \frac{k}{n})^{\frac{T_a}{T_r}})$. Summing these two possible transition probabilities, the final probability of remaining at a is p_2 as shown in Equation (5.2).

Similarly the attacker can successfully compromise b from a when both a and b do not get refreshed and the attacker can follow the edge from a to b with probability p_{ab}. This gives us the probability p_3, which is shown in Equation (5.3).

$$p_1 = p_{ia} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \tag{5.1}$$

$$\begin{aligned} p_2 &= p_{ab} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times (1 - (1 - \frac{k}{n})^{\frac{T_a}{T_r}}) + \\ &\quad (1 - p_{ab}) \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \\ &= (1 - \frac{k}{n})^{\frac{T_a}{T_r}} - p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \end{aligned} \tag{5.2}$$

$$p_3 = p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.3}$$

Based on p_1, p_2, p_3, the following equations define the final probability of compromising b from i. As shown in Equation (5.2), p_2 can be simplified as $(1 - \frac{k}{n})^{\frac{T_a}{T_r}} - p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}}$.

Thus, the final probability of compromising b can be derived as Equation (5.5):

$$P_{success\_b} = p_1 \times [p_2^0 + p_2^1 + \ldots + p_2^\infty] \times p_3 \tag{5.4}$$

$$= p_{ia} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times [p_2^0 + p_2^1 + \ldots + p_2^\infty] \times p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.5}$$

As $0 < p_2 < 1$, then $p_2^0 + p_2^1 + p_2^2 + \ldots + p_2^\infty = \frac{1}{1-p_2}$, thus:

$$P_{success\_b} = p_{ia} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times \frac{1}{1-p_2} \times p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}}$$

$$= \frac{1}{1-p_2} \times p_{ia} \times p_{ab} \times (1 - \frac{k}{n})^{\frac{3T_a}{T_r}} \tag{5.6}$$

This derivation actually shows that the probability of compromising the ultimate target node is the summation of the probabilities associated with all possible intrusion paths the attacker could take that lead to the target. This illustrates the key idea of the analytical model. As long as the attacker can stay on a system node, new attacks will be launched as long as the target has not been obtained.

For example, consider a deeper penetration path, such as attempting to compromise node h from i in Figure 5.1. The attacker might first compromise a, then c and later f, but then lose the privilege of f. In this situation, as the attacker still has c, thus they must restart from c again to regain f. Now assume that the attacker successfully obtains f again and then h. In this case, the overall intrusion path is $i \rightarrow a \rightarrow c \rightarrow f \rightarrow c \rightarrow f \rightarrow h$, where $f \rightarrow c$ actually represents a backward transition. However, as discussed, the analysis model does not model backward transition directly but uses forward and self transitions to indirectly model it. Thus, the overall intrusion path can be indirectly captured as $i \rightarrow a \rightarrow c \rightarrow c \rightarrow c \rightarrow f \rightarrow h$.

The key point here is that in the process of a complete attack to target node h, the attacker can gain and loose intermediate nodes such as c and f multiple times. However, how often and with what probability the attacker moves forward and gets pushed back is

not important any more, as it can be either pushed back to c from f or pushed back directly to a from f. As long as the attacker can stay at a or c or f, he/she will keep attacking the next node, trying to get to h. Obviously, if the attacker loses access to a, c and f, a whole new attack will need to be started. This way, one can avoid modeling the huge number of backward transition probabilities when CAG size becomes large and focus only on the analysis of forward and self transitions.

To verify whether this analytical model can be used to estimate the success likelihood, the values predicted by this model are compared to the success likelihood calculated through Monte Carlo simulations. The CAG used in these experiments is as shown in Figure 5.1. In these experiments, the attack interval T_a is set as 100, the adaptation interval T_r has 17 different values, they are 20, 30, 50, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 1,200,000. In these values, 1,200,000 represents the static situation where no adaptation occurs. In addition, every edge $x \rightarrow y$ is assumed to be associated with the static transition probability $p_{xy} = 0.6$. In each T_r setting, 20,000 complete attacks have been executed. In these experiments, the attacker tried to compromise b, e, g and h. For example, in each complete attack where the attacker tried to compromise b, the attacker tried all edges in the CAG until either the attacker obtained b or got totally pushed back to i. In each of the 20,000 complete attacks attempting to compromising b under a specific T_a and T_r, the success likelihood of compromising b can be calculated by dividing 20,000 by the number of times that b is successfully compromised. Similarly, experimental results can also be used to calculate the success likelihood of compromising e, g and h.

Figure 5.3 shows the comparison between the model probabilities calculated by Equation (5.5) and the success likelihood values obtained from the experiments.

As Figure 5.3 shows, the model matches the trend of the experimental results. However, the left part of the red curve that represents the model's prediction overestimates the results from the Monte Carlo simulations. The first row of Table 5.1 shows that the maximum deviation between the model and experiments is about 8% and standard deviation between

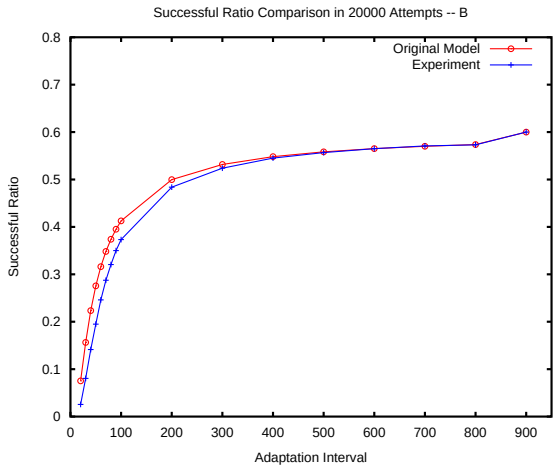


Figure 5.3: Model vs Experiment – Compromise b

the model and experiments is around 4.6%.

To further investigate this, consider node e (Figure 5.1) as the target. Again, all the backward transitions are discarded and only forward and self transitions are considered as shown in Figure 5.4. The following analysis also applies to node f and g. Notice this time, p_1 is still the same, but p_2 represents the self transition probability from a to a when trying to compromise c instead of b. Likewise, p_3 represents the forward transition from a to c, instead of a to b.

Table 5.1: Original Model versus Experiment

	max deviation	variance	std deviation
b	0.082141	0.0021680	0.046562
e	0.074130	0.0015741	0.039675
g	0.073930	0.0016396	0.040493
h	0.065965	0.0016256	0.040319

The probabilities p_4 and p_5 represent the transition probabilities from $c \rightarrow c$ and $c \rightarrow e$

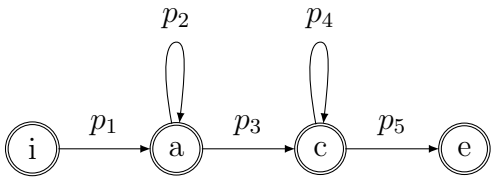


Figure 5.4: Original Transition Model for $i \rightarrow a \rightarrow c \rightarrow e$

and can be calculated as follows:

$$
\begin{aligned}
p_2 &= p_{ac} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times (1 - (1 - \frac{k}{n})^{\frac{T_a}{T_r}}) + \\
&\quad (1 - p_{ac}) \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \\
&= (1 - \frac{k}{n})^{\frac{T_a}{T_r}} - p_{ab} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} && (5.7) \\
p_3 &= p_{ac} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} && (5.8) \\
p_4 &= p_{ce} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times (1 - (1 - \frac{k}{n})^{\frac{T_a}{T_r}}) + \\
&\quad (1 - p_{ce}) \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} && (5.9) \\
&= (1 - \frac{k}{n})^{\frac{T_a}{T_r}} - p_{ce} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} && (5.10) \\
p_5 &= p_{ce} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} && (5.11)
\end{aligned}
$$

Thus, the final success likelihood of compromising e is shown in Equation 5.12,

$$
\begin{aligned}
P_{success\_e} &= p_{ia} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \times [p_2^0 + p_2^1 + \ldots + p_2^\infty] \\
&\quad \times p_{ac} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \times [p_4^0 + p_4^1 + p_4^2 \ldots + p_4^\infty] \\
&\quad \times p_{ce} \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} && (5.12)
\end{aligned}
$$

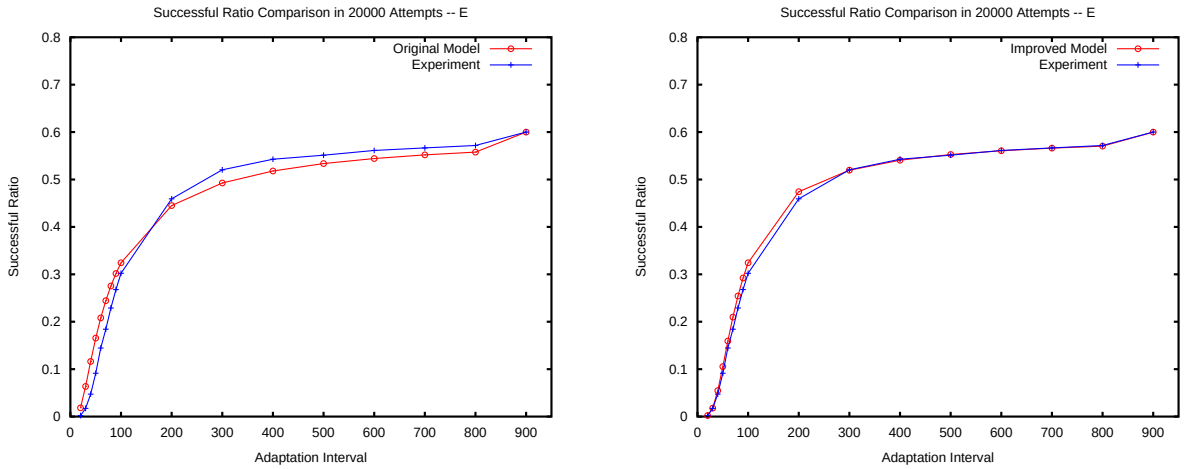
which can be simplified to:

$$P_{success\_e} = \frac{1}{1-p_2} \times \frac{1}{1-p_4} \times p_{ia} \times p_{ac} \times p_{ce} \times (1 - \frac{k}{n})^{\frac{5T_a}{T_r}} \tag{5.13}$$

Figure 5.5a shows the comparison between the probability values calculated through Equation 5.13 and the values computed from experiment results. As shown, the two curves doesn't match exactly. The second row of Table 5.1 shows that for node e, the model predicted results have a maximum deviation around 7.4% and a standard deviation around 4% as compared to the Monte Carlo simulation results. A careful comparison shows us that the model underestimates the experimental results when $T_r > T_a$, but overestimates the experimental results when $T_r < T_a$. This observation reveals a missing aspect of this original model. When we derive the probability formula for each transition, the original model tries to model each transition edge as a single step attack with time interval T_a. However, when $T_a < T_r$, a diligent attacker can actually launch multiple single step intrusion attempts in time period T_r if the first one fails, thus increasing the possibility of success. Also when $T_a > T_r$ the attacker could be kicked out without even being able to finish a single step attack, thus reducing the probability of success. However, the original model does not consider this and treats it as a single step intrusion, which is why it overestimates the success likelihood when $T_a > T_r$. Based on this analysis, the next section presents an improved model that takes into consideration these missing aspects and shows that it improves the accuracy of the analytical model.

5.2.5 An Improved Model

In this section, an improved model is proposed to better estimate the success likelihood of compromising each node. As analyzed above, the main problem with the original model is that when T_r becomes larger than T_a, we ignore the fact that the attacker can actually



(a) Original Model vs Experiment (b) Improved Model vs Experiment

Figure 5.5: Comparison of Compromise e

launch multiple attacks during T_r. When T_r becomes smaller than T_a we ignore the fact that the attacker can be kicked out and can not spend the whole T_a attack interval. The model proposed in this section is enhanced to compensate for these situations in both forward and self transitions.

Using Figure 5.4 as an example, the static probability associated with the edge from a to c is not p_{ac} any more, but $1-(1-p_{ac})^{\frac{T_r}{T_a}}$. In time period T_r, attacker can launch $\frac{T_r}{T_a}$ intrusions. The probability that all these attacks are fail is $(1-p_{ac})^{\frac{T_r}{T_a}}$, thus the probability that the attacker can succeed in this attack is $1-(1-p_{ac})^{\frac{T_r}{T_a}}$. In the improved model, this probability is used to replace all the p_{ac} probabilities in the original model. Similarly, $1-(1-p_{ce})^{\frac{T_r}{T_a}}$ is used to replace all the p_{ce} probabilities in the original model.

Thus the improved transition model for $i \rightarrow a \rightarrow c \rightarrow e$ is shown in Figure 5.6 where p'_1, p'_2, p'_3, p'_4 and p'_5 represent the new transition probability.

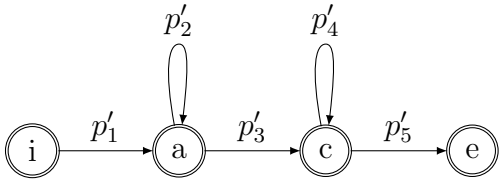


Figure 5.6: Improved Transition Model for $i \rightarrow a \rightarrow c \rightarrow e$

Table 5.2: Improved Model versus Experiment

	max deviation	variance	std deviation
b	0.039401	5.6117e-04	0.023689
e	0.025617	1.8209e-04	0.013494
g	0.025922	1.7676e-04	0.013295
h	0.027851	1.5973e-04	0.012639

$$p'_1 = p_{ia} \times (1 - \frac{k}{n})^{\frac{T_a}{T_r}} \tag{5.14}$$

$$p'_2 = (1 - \frac{k}{n})^{\frac{T_a}{T_r}} - (1 - (1 - p_{ac})^{\frac{T_r}{T_a}}) \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.15}$$

$$p'_3 = (1 - (1 - p_{ac})^{\frac{T_r}{T_a}}) \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.16}$$

$$p'_4 = (1 - \frac{k}{n})^{\frac{T_a}{T_r}} - (1 - (1 - p_{ce})^{\frac{T_r}{T_a}}) \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.17}$$

$$p'_5 = (1 - (1 - p_{ce})^{\frac{T_r}{T_a}}) \times (1 - \frac{k}{n})^{\frac{2T_a}{T_r}} \tag{5.18}$$

Equation 5.19 gives the final improved model for calculating the probability of compromising e in the CAG of Figure 5.1. The derivation process is similar as the one used in Equation (5.12). The formula used to compute the success likelihood of compromising f and g can be derived similarly as e.

$$P_{succsess\_e} = \frac{1}{1 - p'_2} \times \frac{1}{1 - p'_4} \times p_{ia} \times (1 - (1 - p_{ac})^{\frac{T_r}{T_a}}) \times (1 - (1 - p_{ce})^{\frac{T_r}{T_a}}) \times (1 - \frac{k}{n})^{\frac{5T_a}{T_r}} \tag{5.19}$$

Figure 5.5b shows the comparison between the probability values calculated from Equation 5.19 and the experimental results. As we can see, the improved model now has a better match with the simulation results than the original model. The second row of Table 5.2

also clearly indicates that the maximum deviation has been reduced from 7.4% to 2.56% and the standard deviation has been reduced from 0.039675 to 0.013494.

Figure 5.7 shows the comparison of original model versus the experiments as well as a comparison of the improved model versus the experiments when the target node is b, g and h. All the figures and statistical data collected in Table 5.2 demonstrates that the improved model more closely matches the Monte Carlo simulation results.

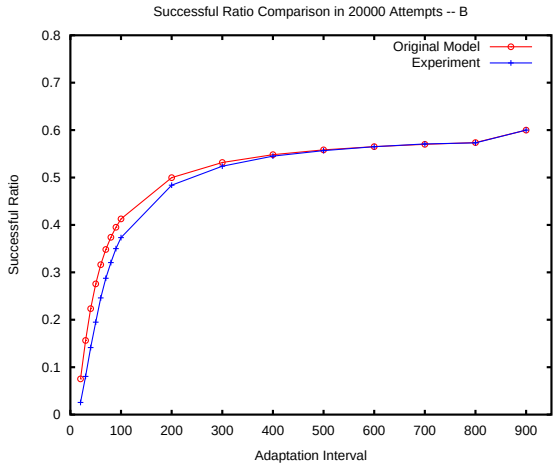
Another finding is that as the target hides deeper, the success likelihood becomes smaller. For example, in Figure 5.7, at adaptation interval $T_r = 100$, the success ratio for compromising b is around 0.4, for compromising g is reduced to around 0.3, and for h is further decreased to around 0.27.

5.2.6 General Form

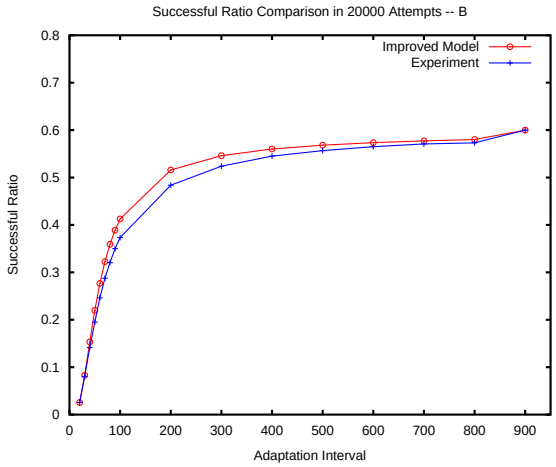
In the previous section, a model is derived based on the assumption that the attack interval T_a associated with each edge is the same. This assumption directly follows the assumption made in the simulator that all VMs assigned to play the given *role* has the same configuration except ID and IP address. However, in reality, different VM will be assigned to play different roles and will have different configuration. Thus, the time spend on compromising different nodes will usually not the same.

Based on the knowledge of our previous analysis and the simulation results, this section proposes a new model to relax this assumption. The following defines several notations used in the model.

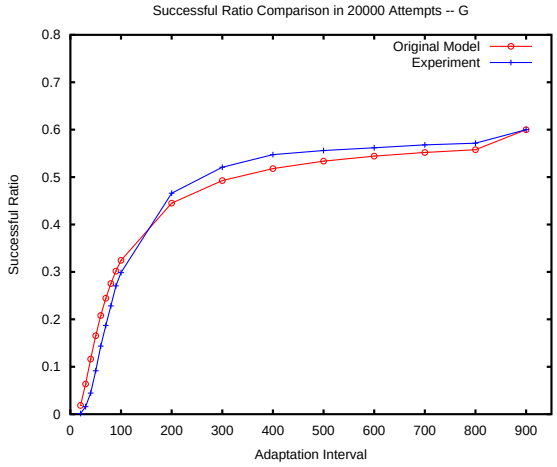
- i – the node where the attacker launches the intrusion.
- t – the target node.
- $x \rightarrow y$ – an edge from node x to y.
- $i \rightarrow a \rightarrow b \rightarrow \ldots \rightarrow t$ –the path the attacker must follow to compromise t.



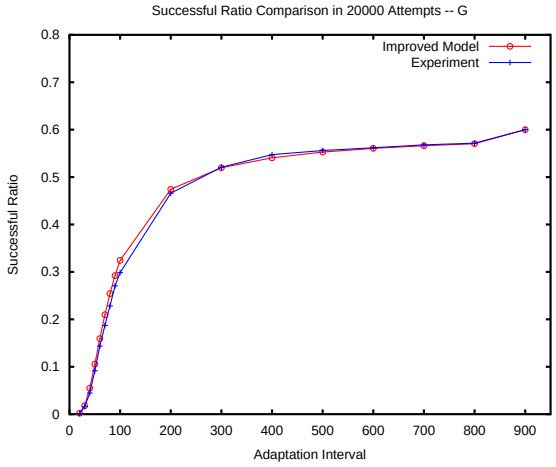
(a) Original vs Experiment – b



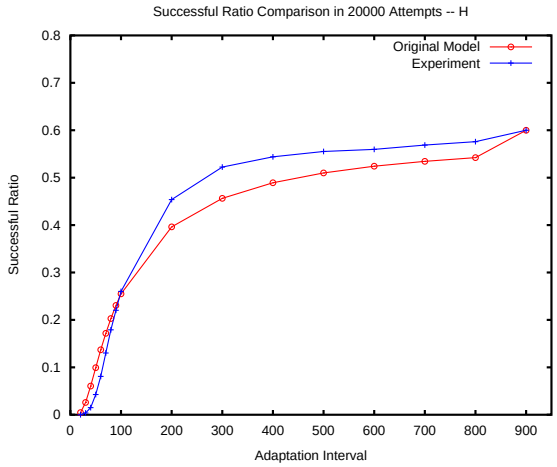
(b) Improved vs Experiment – b



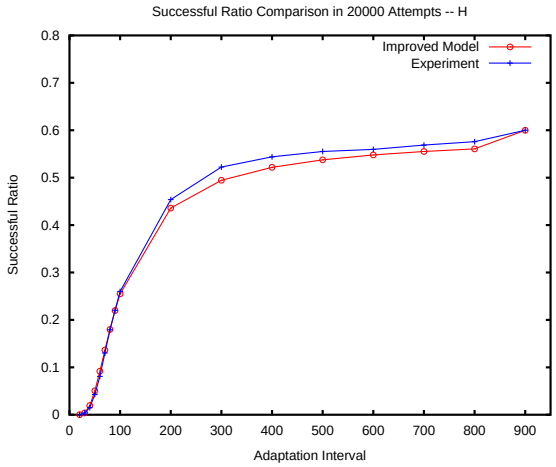
(c) Original vs Experiment – g



(d) Improved vs Experiment – g



(e) Original vs Experiment – h



(f) Improved vs Experiment – h

Figure 5.7: Model Comparisons

- V_p – nodes in the intrusion path.
- E_p – edges in the intrusion path.
- p^r_{xy} – forward transition probability of $x \to y$ where adaptation might occur.
- p^s_{xy} – static transition probability from x to y.
- T^{xy}_a – attack time interval associated with edge $x \to y$.
- T_r – adaptation time interval.

The improved model with the time assumption relaxed can be summarized as follows where Equation (5.20) represents the forward transition probability, Equation (5.21) represents the self transition probability and Equation (5.22) represents the more success likelihood from i to target t.

$$p^r_{xy} = \begin{cases} p^s_{xy} \times (1 - \frac{k}{n})^{\frac{T^{xy}_a}{T_r}} & \text{if } x = i; x, y \in V_p; x \to y \in E_p \\ (1 - (1 - p^s_{xy})^{\frac{T_r}{T^{xy}_a}}) \times (1 - \frac{k}{n})^{\frac{2T^{xy}_a}{T_r}} & \text{if } x \neq i; x, y \in V_p; x \to y \in E_p \end{cases} \tag{5.20}$$

$$p^r_{xx} = (1 - \frac{k}{n})^{\frac{T^{xy}_a}{T_r}} - (1 - (1 - p^s_{xy})^{\frac{T_r}{T^{xy}_a}}) \times (1 - \frac{k}{n})^{\frac{2T^{xy}_a}{T_r}} \text{ if } x \neq i; x \to y \in E_p \tag{5.21}$$

$$\begin{aligned} P_{successs\_t} &= \prod_{\substack{x \in V_p \\ x \neq i,t}} \frac{1}{1 - p^r_{xx}} \times \prod_{x \to y \in E_p} p^r_{xy} \\ &= \prod_{\substack{x \in V_p \\ x \neq i,t}} \frac{1}{1 - p^r_{xx}} \times \prod_{\substack{x \to y \in E_p \\ x \neq i}} (1 - (1 - p^s_{xy})^{\frac{T_r}{T^{xy}_a}}) \times p^s_{iy} \times (1 - \frac{k}{n})^{\frac{T^{ix}_a + \sum\limits_{\substack{x \to y \in E_p \\ x \neq i}} 2T^{xy}_a}{T_r}} \end{aligned} \tag{5.22}$$

Here, T_a has been replaced with T^{xy}_a to relax the assumption that all values of T_a are identical. Thus, attack time associated with different edge could be different. Another

important assumption is that all VMs assigned to play the given role has the same configuration as well as a vulnerability. Here this assumption is relaxed such that VMs assigned to play different role could have different configuration, but all VMs assigned to play the same role still have same configuration, this is captured by p^s_{xy}. However, in reality, MTD could adopt different movement mechanisms, thus VMs assigned to play the same *role* would have different configuration states with different – or no – vulnerabilities.

Relaxing this limitation requires the MTD theory in Chapter 6 that helps understanding the interactions between attacker and MTD system. In fact, as will see in the Cyber Attack Theory, T^{xy}_a and P^s_{xy} are important characteristics of an attack instance. Definitions in MTD System Theory, such as diversification and randomization, also play a key role as they formally describe the configuration state space and the probability that each configuration state shows up. Clearly, whether a vulnerability exists or not is closely related with a configuration state. We will see in Chapter 6 that once the theory is in place, solve this limitation is straightforward.

5.3 Conclusion

In this chapter, an efficient and scalable analytical model that can be used to analyze and estimate the success likelihood of multi-hop remote attacks in an MTD context has been presented. This chapter motivates the need for an analytical model and describes the challenges of such a model. By focusing on forward and self transitions (thus indirectly capturing backward transitions), an analytical model suitable for non-cyclic network topology structures is presented.

Chapter 6

A Theoretical Framework for Moving Target Defense

If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.

-Sun Tzu, The Art of War.

6.1 Overview

Security is a critical concern that exists in computer systems as well as in the physical world. Chapter 3 presented a taxonomy of the state-of-the-art *moving target defense* research in a cyber-security context. However, as with security in general, many intuitive examples of MTDs are found in physical world as well.

Example 6.1. In an air battle, two opposing aircraft often end up in a one-on-one situation, which is typically called a *dogfight*. In dogfights, one aircraft is the attacker while the other becomes the defender. It is up to the pilot of the defending aircraft to maneuver his or her

aircraft to avoid being shot down by the attacker. In this way, a dogfight is similar to cyber-security.

In a dogfight, performing tactical maneuvers gives the defender a greater chance to avoid being shot down as opposed to simply doing nothing. If we think of the defender as an MTD system, the target is the defender that is trying to maneuver the aircraft to dodge an incoming missile. In this case, the pilot may maneuver in three dimensional physical space using *tactics* of defensive aircraft maneuvers (such as a high yo-yo defense, unloaded extension, high-g barrel roll, defensive spiral, etc.) in order to change the state of the two aircraft (location, speed, yaw, pitch, roll, etc.) while not exceeding the physical limitations of the pilot.

This movement can increase the attacker's uncertainty regarding the defender's location and direction. The pilot chooses the time to trigger a maneuver either proactively, based on pilot's training and intuition, or reactivity after detecting an incoming missile. Other examples in the physical world are animals running fast and zig-zagging to avoid being shot by a hunter or soldiers moving quickly to find the cover to evade gunfire. These are simple yet intuitive real world examples where targets use movement to protect themselves. More complex examples include transportation surveillance, airport security patrolling, etc.

Example 6.2. The ARMOR system[109,110,111] (Assistant for Randomized Monitoring over Routes) has been successfully deployed at Los Angeles International Airport (LAX) since 2007. ARMOR uses a game-theoretic approach to randomly place vehicle checkpoints on roads entering airport terminals and to schedule canine patrol routes inside terminals for sniffing for bombs. Both applications are constrained by available resources such as police units, canine units and inspection devices, etc. The challenge is to optimally allocate limited resources to increase the effectiveness of airport security, all while avoiding patterns that may be discovered by persons intent on evading security. ARMOR considers different characteristics of airport terminals such as physical size, passenger loads, flight schedules in the risk assessment of the eight LAX terminals. ARMOR models different types of attacks

using different payoff functions and optimizes the allocation of limited security resources using a Bayesian Stackelberg game solver[112].

In this example, ARMOR can be viewed as using an real-world MTD approach to increase security by randomizing the allocation of limited defensive resources. The *adaptable aspects* are the resources, such as police units and canine units, while the *tactic* used is a Bayesian Stackelberg game solver that determine the optimal movement and time to *trigger* that movement. Here the *movement* is not directly applied to the potential targets themselves (e.g., flights), but instead is applied to defensive elements of the system to obscure them from potential attackers. By doing so, it increases the difficulty of locating and successfully compromising the targets by increasing the possibility of detection along the routes required to reach the targets.

From these real world examples, we see that MTD systems can increase the security through movement by either increasing the attacker's uncertainty of locating the target or by increasing the attacker's probability of detection along the intrusion path, all while minimizing the cost to normal system operations. The *movement* does not necessarily need to be directly applied to the target, but also can be applied to the path that leads to the target. These ideas apply to cyber world as well. In either case, solving these problems using an MTD approach requires us to first identify the attack type to defeat, and then consider a suitable combination of adaptable aspects with strategies and tactics.

6.1.1 General MTD Adaptation Effect

Figure 6.1 illustrates the effect of adaptation on the attacker. At time t_1, an adaptation just finishes and the system has configuration, s_g. Suppose now an attacker starts to investigate the system configuration, as time elapses, the attacker knows more information about the system, thus the system configuration uncertainty for attaker decreases. As shown at time, t_2, the exploration space shrinks and the attacker gets closer to reaching the potential vulnerability. However, if at t_2 another adaptation launches and produces a new configu-

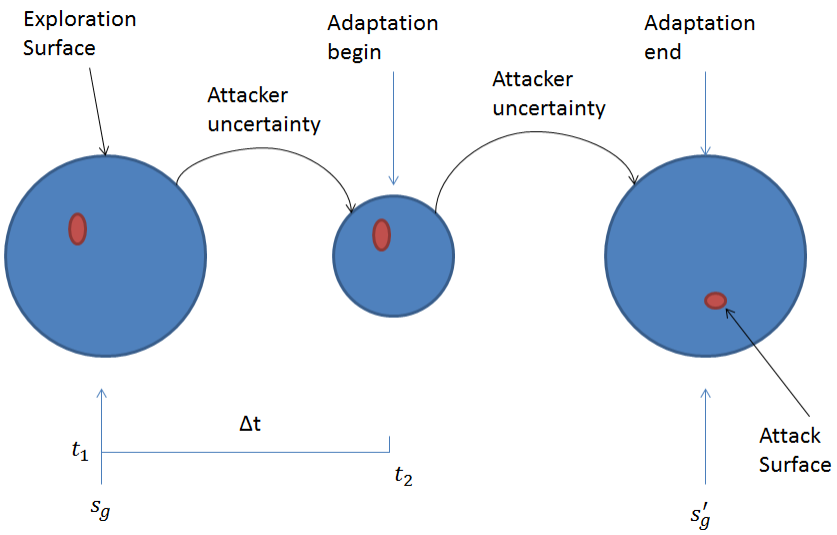


Figure 6.1: Adapatation Effect Intuition – Attacker Uncertainty

ration, s'_g, which should invalidate the attacker's previous penetration effort and pull the configuration uncertainty level back.

6.1.2 General MTD process

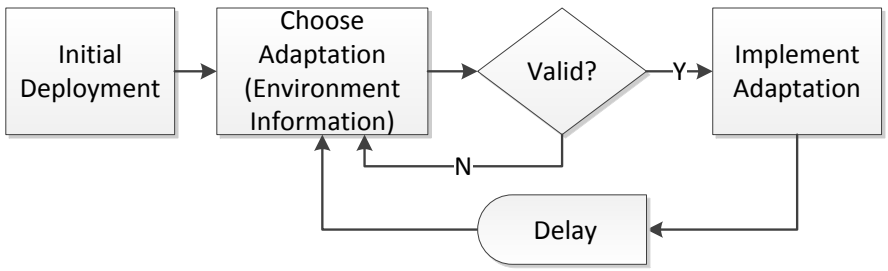


Figure 6.2: Overview of MTD process

The general workflow of an MTD system is shown in Figure 6.2. An MTD is first designed, implemented and deployed in its initial configuration. Once the system is executing, an MTD system will choose one or more adaptations to apply to its configuration. As discussed above, the adaptation can be triggered either proactively based on fixed or random time point or reactively by environment information, such as IDS alerts or operation failures. All operational systems have a set of constraints and resource limitations as demonstrated

in Example 6.2. The configuration resulting from the chosen adaptation must be checked against these constraints to ensure that the new configuration will be valid. If it is not valid, a new adaptation will have to be chosen. Once an adaptation has passed the validity test, it can be implemented. As will be discussed later, there are several key problems that must be solved in order to make an MTD system work.

6.1.3 Motivation for an MTD Theory

In the scientific world, a theory is generally something that defines a set of concepts, their relationships and principles to help shape the view of a field and clarify the essential problems. Theories are typically used to understand and explain things we observe and to predict things that have not been proven.

The past few years has seen a growing need within the research community to develop a science of security[113]. The motivation is to develop a systematic body of knowledge with strong theoretical and empirical underpinnings to inform the engineering of systems that can better resist known and unanticipated attack types. A theory for moving target defense will not only create a set of common and well defined terms, but also provide a framework and systematic way to think and analyze MTD problems and solutions.

Concretely, the theory should define key concepts such as adaptation, diversification, randomization, attack surface, and exploration space in a way that is both formal and appropriate to the dynamic nature of MTD system. To support the understanding and analysis of the interaction between MTD systems and the attacks to thwart, the theory should also define key concepts that support precise discussion of attacker knowledge, attack types, and attack instances. Because the implementation of an effective MTD mechanism only makes sense in the context of a specific threat model[114,115].

6.1.3.1 Motivation Questions to be answered in General

Pragmatically, the theory should be practical enough to inform design decisions during the design of MTD systems and should be able to answer general questions like these:

- Given an attack type or a sequence of attack types, what can an MTD system do to defeat it?
- Given an attack type and the MTD system setting, how to analyze whether this MTD system is potentially effective to thwart this given attack type?
- What are the conditions of, or what constitutes the success likelihood of intrusion under MTD?
- How does MTD impact attacker's intrusion?

6.1.3.2 Motivation Questions to be answered in Concrete Scenario

Ideally, the theory should also be straightforward enough to allow MTD system designers to decide how to use existing configuration choices and diversification to increase security. Given a concrete MTD system, it should allow MTD designers to analyze the effectiveness of adapting various combinations of configuration aspects to thwart different types of attacks. More specifically, it should be able to answer these questions in a concrete MTD system.

- If a given MTD system is potentially effective to thwart an attack type, how can we quantify this effectiveness in terms of success likelihood of intrusion?
- How are these important parameters, such as T_a, P_{static}, adaptation interval, diversification, randomization and configuration space, interrelated and impact the success likelihood of intrusion?
- Fixing T_a, P_{static}, can we change MTD settings, such as adaptation interval, configuration space, etc to see the effectiveness of MTD with different settings?

- Fixing MTD settings, can we tweak T_a, P_{static} to see when this setting will be effective or ineffective?

6.1.4 Approach

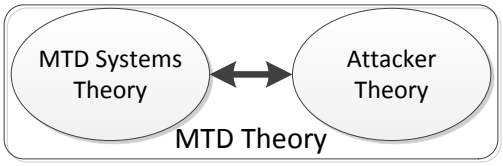


Figure 6.3: MTD Theory Overview

The high level approach to developing a theory for MTD is shown in Figure 6.3. The first step is to develop a theory of MTD Systems. This theory focuses only on the system itself and how it adapts over time to achieve its overall goals. The second step is to develop an Attacker Theory, which will describe an attacker's goals and actions they can take to reach their goals. The final step will be to combine the two into an overall MTD Theory. The objective of MTD Theory will be to define how elements of the MTD and Attacker theories interact. This is especially important in being able to understand the true effect of an MTD system as its effectiveness only makes sense in light of actions from an attacker for a specific type of attack.

6.1.5 Scenarios

This section presents two scenarios that will be used through out. Examples provided below to help understand definitions and theorems will be based on these two scenarios.

6.1.5.1 K-state Web Authentication

Kansas State University's web authentication requires each student to register an ID and password to login to use the university's online service and resources. Every six month each student is asked to update their password to a different one to keep their account active.

6.1.5.2 Mission Planning System

Another scenario is a simple military mission planning system. An overview of this system is shown in Figure 6.4. An authorized user can remotely access the mission planner to construct a specific mission. Through the Planner (a web server), which provides the web UI, authorized actions such as adding new military strategy, establishing plans or tactics, allocating owned resources, etc. could be performed. To support these actions, related information are stored in three different databases. An asset database provides the available resources information such as tanks, troops, aircraft, etc. A geographical database provides map and geographical information such as coordinates, satellite images, etc. A target database stores information about targets of interest such as name, location, features, etc. Such distributed information storage also enhances security in that one database compromise does not lose all sensitive data.

In this system, the data stored in the three databases – the AssetDB, the GeoDB and the TargetDB are most likely targets of interest for a serious attacker.

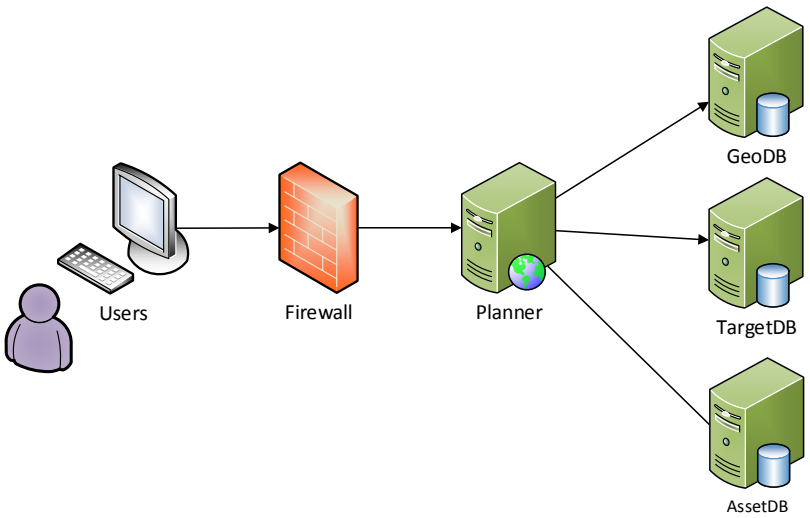


Figure 6.4: Mission Planning Scenario

6.2 MTD System Theory

This section defines the key concepts required to formally talk about MTD systems and their basic properties. The focus is on formally defining an MTD system, although more definitions about adaptation, configuration space, diversification, and randomization in light of the MTD system definition are given as well. Since the essence of an MTD system is adapting the configuration of the system over time, this section starts by defining what a configurable system is, borrowing key terms and concepts from configuration management theory[116].

6.2.1 Configurable System

When talking about configuring a system, we generally refer to the physical devices that are part of a system, the software installed on those devices, and the settings of that software. In the context of MTD, these configuration elements are defined as a configuration parameter (borrowing from[116]) that can take on various values to specify the specifics of the configuration.

6.2.1.1 Configuration Parameter

The definition of a configurable system starts from defining a configuration parameter as a variable with an associated type.

Definition 6.1. *A* configuration parameter*, π, is a unit of configuration information that can take on a value based on its type.*

Definition 6.2. *A* configuration parameter type*, Π, is a label identifiable with the domain of possible values that the parameter can assume[116]. The associated domain of a configuration parameter π_i is denoted as Π_i.*

In essence, a configuration parameter can be viewed as a variable to which we can assign values. These values can be used to describe a piece of hardware, the software installed on

that hardware, or the settings of the software itself, etc. Examples of configuration units can be a specific physical or virtual machine host, the amount of memory installed, the speed of the processor, the operating system installed, the IP address of the host, the ports that are open, the password of a user account, etc. While some configuration parameters are basically fixed (e.g., the size of memory in a physical host), MTD systems obviously are more interested in the configuration parameters that can be modified during execution (the size of memory in a virtual host or the IP address of a host). The configuration parameter that can be modified by MTD system is referred to as an adaptable configuration unit.

To capture the configuration of an entire system or a complex component of that system, we introduce the notion of a composite configuration parameters.

Definition 6.3. *A* composite configuration parameter*, π, is a configuration parameter that is composed of a set of sub configuration parameters, $\pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$. The domain of a composite configuration parameter π is derived from the sub configuration parameter domains, $\Pi = \Pi_1 \times \Pi_2 \times \ldots \times \Pi_n$.*

Thus, a configuration unit can be either large or small. In the K-State authentication scenario, a configuration parameter, $\pi_{account}$ can be used to capture a student's K-State account. This parameter, $\pi_{account}$, can be further divide to a set of sub configuration parameters, $\langle \pi_{eid}, \pi_{password} \rangle$. For the mission planning scenario, one can use a composite configuration parameter $\pi_{planner}$ to describe planner's corresponding (virtual) machine's overall configuration. Appropriate sub configuration parameters could include $\langle \pi_{memory\_size}, \pi_{disk\_size}, \pi_{cpu\_type}, \pi_{operating\_system}, \pi_{web\_application\_server}, \pi_{service\_implementation}, \pi_{ip\_address}, \pi_{service\_port} \rangle$. Here sub configuration parameter could be a composite configuration parameter too. For example, $\pi_{operating\_system}, \pi_{web\_application\_server}, \pi_{service\_implementation}$ are possible composite configuration parameters.

6.2.1.2 Configuration State

The process of reconfiguration, which is at the heart of MTD systems, is the process of moving from one configuration to another. To capture the notion of a specific configuration, a configuration state is introduced. Typically, a configuration state, by default, refers to the overall configuration of a system; however, it can also be used to refer part of that system as well.

Definition 6.4. *A* configuration state*, s, is a unique assignment of value(s) from Π to a configuration parameter π. An assignment of some value z in Π to π is denoted as $\pi \leftarrow z$. If π is a composite configuration parameter and $\pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$, then s is a configuration state of π if $s = \langle s_1, s_2, \ldots, s_n \rangle$ where $\forall i \in [1, n], s_i \in \Pi_i \wedge \pi_i \leftarrow s_i$.*

Based on this, the current configuration of a system becomes clear. Using the example of the K-State account configuration parameter above, a tuple that combines a student's eid and password such as $\langle john, Abc1234 \rangle$ is a configuration state of $\pi_{account}$. An assignment of valid values to each of the sub configuration parameters of $\pi_{planner}$, for instance, $\langle \pi_{memory\_size} = 4GB, \pi_{disk\_size} = 100GB, \pi_{cpu\_type} = intel\_i7, \pi_{operating\_system} = Ubuntu\_14.04\_LTS\_64bit, \pi_{web\_application\_server} = apache, \pi_{service\_implementation} = PHP,$ $\pi_{ip\_address} = 192.168.10.18, \pi_{service\_port} = 80 \rangle$ is an example configuration state of $\pi_{planner}$. Notice here, $\pi_{operating\_system}, \pi_{web\_applicationserver}, \pi_{service\_implementation}$ are composite parameters, and the assigned value for these composite parameters can be understood as the tag or key of a concrete state. For example, by assign $\pi_{operating\_system} = Ubuntu\_14.04\_LTS\_64bit$, then use this tag the program will be able to find the linked OS installation templates.

6.2.1.3 Action

To change one configuration into another configuration requires actions to be taken on the part of the system administrator. Traditionally, these actions are performed manually. Although recently new configuration management tools have automated much of the tedious

nature of these actions, however, ultimate control of what to change still resides with the administrators.

Definition 6.5. *A configuration* action*, α, is an operation that can modify a value of an existing configuration parameter, π, or add/delete a configuration sub parameter from a composite configuration parameter. When adding a sub parameter, we assume the parameter is initialized with a valid value. An action can also be composed of a sequence of sub configuration actions.*

Operations such as powering machines on and off, assigning IP addresses, installing Operating systems, deploying services, etc. are all typical actions. In addition, operations such as adding/removing computers to a system, adding/removing deployed services to a computer, etc. are also actions.

6.2.1.4 Configurable System

The configuration of a system is defined as a set of configuration parameters that can be modified by a set of configuration actions. By combining these concepts with configuration states, configurable systems can be defined, upon which the definition of an MTD system is built.

Definition 6.6. *A* configurable system *is a labeled transition system, $\Gamma = (S, \Lambda, \tau)$, where $S = \{s_1, s_2, \ldots\}$ is a finite or recursively enumerable set of configuration states the system can be in, $\Lambda = \{\alpha_1, \alpha_2, \ldots\}$ is a finite or recursively enumerable set of actions, and $\tau : S \times \Lambda \rightarrow S$ is the state transition function.*

This configurable system definition is based on labeled transition systems as the essence of configuration management is change. Here the set of allowable changes is captured in the state transition function, τ.

For the K-State account scenario, the account authentication can be viewed as a configurable system. Usually, the eid is not changed once it is set, so an eid generally has one

state. For passwords, however, state, S, is the set of all possible combinations of characters that a user with eid could choose. Actions consists of editing the account information to set the password. Transition functions capture the transition from old password to a new password after the action applied.

The airbattle example can also be considered a configurable system where the state, S, is the physical locations of the aircraft, actions are the manuevers that the plane could take, and the state transition function captures the location transition from old position to new based on the action performed. Similarly, the mission planning scenario can also be thought as a configurable system. More details that show how this theory maps to the mission planning scenario will be provided in the validation section after Cyber Attack Theory and MTD Theory is developed.

Until now, the notion of configuration consistency and validity have been ignored. These are addressed in the next section using system goals and constraints.

6.2.2 System Goals

The heart of the MTD paradigm is adapting the system to keep an attacker from taking the time required to successfully attack and compromise the system. However, major problems associated with this constant adaptation is that this could keep the system from achieving its intended mission. It has recently been recognized by several in the adaptive systems community that the key to *effective* adaptive systems is explicitly modeling the requirements or objectives of the system[117,118]. Understanding the objectives of the system is critical in making the determination of what is a *valid* adaptation and what will simply lead to chaos. Thus, the notion of system goals is introduced here, which will be used later to define the notion of valid adaptations.

Definition 6.7. *A* goal*, g, captures an intended function of a computer system. There are two types of goals of interest: operational goals and security goals. Each system has a set of goals, G that is captured by a tuple $\langle G_o, G_s \rangle$, where $G_o = \{g_{o1}, g_{o2}, \ldots g_{oj}\}$ represents the*

operational goals of the system and $G_s = \{g_{s1}, g_{s2}, \ldots\ldots g_{sk}\}$ *represents the security goals of the system.*

Operational goals capture the mission the system was built to support. An operational goal is a high-level business goal (e.g., K-State login system, blog website, mission planning website, etc.) whose purpose is to organize the IT elements of the system around business objectives. The operational goals are guideposts used to ensure that any adaptations to the system configuration still support the overall mission. Security Goals, on the other hand, define the critical parts of the system that should be protected. If an mission plan website relies on a database of commander's information, then protecting that database becomes an important security goal.

While a system can be in a variety of configuration states, only some of those states actually provide the capabilities required to achieve the operational goals of the system. For instance, a mission planning system may be in a configuration that does not include a database server that stores commander's information (thus ensuring the security goal not to allow the database to be compromised). However, this configuration is not really helpful since one of the key goals of the system is to allow commanders to log in and plan missions through the website. Therefore, a relation needs to be defined between configurations of the system and the goals that those configurations achieve. This relation is used later to ensure that as system adapts, it is capable of achieving the overall operational goals.

Definition 6.8. *If a goal,* g*, can be realized in a system in configuration state,* s*, we say that* s achieves g *and define a relation* achieves: $G \times S$ *to capture this relationship. If* $\forall g \in G, \langle s, g\rangle \in$ *achieves, we say* s *achieves a set of goals,* G*, denoted as* $s \vdash G$*.*

As discussed in Section 6.1.4, the objective is to define an Cyber Attack Theory as well as an MTD Systems Theory and then show they are related. Attacker is envisioned to have their own set of goals and thus it seems obvious that when the goals of the attacker and the MTD system conflict, then the MTD system must be able to take steps to counterattack that attack.

6.2.3 System Policies

To ensure the parts of a system can function together efficiently and effectively, every system has a set of policies that define how the system can and cannot be structured. For example, the K-state authentication scenario requires students to change their password every 180 days. To set up a new one, the password chosen needs to fullfil some requirements such as: 10-30 characters (5 must be different), contains characters from 3 of these categories: uppercase letters, lowercase letters, numbers and special characters. These are all typcial policy examples that one would encounter.

Usually, these policies are implicit, which leads to significant problems when changes are made to the system without understanding all the pertinent policies. Ideally, an MTD system will require system designers to explicitly state these policies so that the MTD system can reason over them before making adaptions.

Definition 6.9. *A* policy*, p, defines a restriction on the configuration state of a system. Each system has a finite or recursively enumerable set of policies, $P = \{p_1, p_2, \dots\ p_l\}$. The aggregated restrictions of system policies, P, on system configuration states, S, define a relation* satisfies*: $S \times P$. We say a state, s,* satisfies *a set of policies, P, denoted as $s \succ P$, if $\forall p \in P, \langle s, p \rangle \in satisfies$.*

If a configuration policy is violated, the system will not operate as intended. Thus, as the system adapts, it is critical that it does not fall into an *inconsistent state* where it violates any policies. Next we define a consistent configuration state.

Definition 6.10. *A system is said to be in a* consistent state*, s, if $s \succ P$, where P represents the current system policies. The set of all consistent states of a system is denoted as $S_c = \{s | \forall s \in S \wedge s \succ P\}$. Any state that is not a consistent state is an inconsistent state.*

While being in a consistent state is necessary, the system still needs to ensure it is in a state so that it can achieve its intended goals. Thus we use the definition of achieves from Definition 6.8 to define a valid state.

Definition 6.11. *A* valid state *s is a consistent state that is capable of achieving all of the existing system operational goals, G, i.e., $s \vdash G$. We define the set of* all valid states *of a given system, S_v, as $S_v = \{s | \forall s \in S, s \succ P \wedge s \vdash G\}$.*

Knowing a state is consistent and valid is very useful. However, when dealing with an entire system, MTD needs to be able to reason about the configurations that are really of interest – those that define a complete system for the purposes of achieving the operational goals of the system. Thus, we define a complete configuration as one that has all the configuration parameters required to configure a system that is actually capable of achieving the goals of the system.

Definition 6.12. *A* complete configuration *is a configuration parameter whose value can be a valid state. Formally, this is stated as $\exists s \in S_v, \pi \leftarrow s$.*

Since a complete configuration might also contain unnecessary configuration parameters, to restrict it to the configuration information that contains only necessary parameters to achieve the overall goals of the system, we define a minimum complete configuration. Again, it should be noted that this is a minimum complete configuration whose valid states can achieve the system goals G.

Definition 6.13. *A* minimum complete configuration *is a complete configuration parameter that has the minimal required parameters. Formally, this is stated as $\nexists \pi_i \in \pi$, such that $\exists s \in S_v, \pi - \{\pi_i\} \leftarrow s$.*

6.2.4 Adaptation

While actions have been defined above as operations that take a system from one configuration state to another, what MTD is really interested in is transforming a system from one configuration state to a *valid* configuration state. This will be the basis of defining the kind of MTD systems of interest and will be discussed further in the next section. To handle this

specific type of transformation operation, an adaptation is now defined as a restriction on the set of actions to capture this.

Definition 6.14. *An* adaptation *is a sequence of actions* $A = \{a_1, a_2, \ldots, a_k\}$ *that transforms a system from a state,* s*, to a valid state,* s_v*.*

Now we have all the definitions to define an MTD system, which is done in the next section.

6.2.5 MTD System

An MTD system is a configurable system that can adapt its configuration during execution. As discussed in the previous section, an MTD system should also be able to configure itself so that it is always in a consistent state that can achieve the overall goals of the system.

Definition 6.15. *A* moving target defense (MTD) system*,* Σ*, is a tuple* $\langle \Gamma, G, P \rangle$*, where* Γ *is a configurable system,* G *is the set of goals which includes both operational goals and security goals and* P *is the set of polices.*

Notice that although S in Γ will be restricted to S_v by G and P, an MTD system is still defined using S, the set of all configuration states of Σ, instead of S_v. The reason is that an MTD system does not always have to be in a valid state. Operational failures or system errors could lead the MTD system to an invalid state. However, as defined above, when applying an adaptation to MTD system, it needs to result in a valid state.

In order to reason and discuss the configuration of an MTD system, one must be able to refer to the *current* configuration of a system at any given point in time. Since it is necessary to reason about the entire configuration, the configuration must be complete as defined in Definition 6.12. Since no extraneous configuration information should be included, it must be minimum as defined in Definition 6.13. As configuration actions may add or remove configuration parameters from a composite configuration parameter, the configuration state

of an MTD system can change in terms of the configuration parameters as well as their values. The current configuration of an MTD system is defined as its configuration state.

Definition 6.16. *At any point in time, each MTD system, Σ, has a minimum complete configuration denoted Σ_π. Σ_π also has a unique value, $s \in S$, which is called the* configuration state *of Σ.*

6.2.6 Configuration Space

The set of all valid states, S_v, captures the overall *configuration space* of an MTD system's valid configuration. This can also be derived from the domain of Σ_π, which has been defined in Definition 6.3.

Definition 6.17. *The* configuration space *of an MTD system, Σ, with configuration parameter Σ_π is the domain of Σ_π, which is S_v. S_v can be computed as $S_v = \prod \Pi_i$, where the cross product operation should obey constraints such that $\forall s \in S_v, s \succ P \wedge s \vdash G$.*

As defined above, S_v can be computed as the cross product from Π_i of each configuration parameter, π_i, where the cross product operation should obey the constrains such that the result is a valid state. Each Π_i actually captures the configuration space of the configuration parameter, π_i.

As discussed earlier, a critical part of a cyber attack is the reconnaissance or exploration of the target system's configuration. To understand the effect of an MTD system, we must be able to characterize the space, or *exploration space* that the attacker must explore before attacking. Clearly, the exploration space is related to the *configuration space* of the MTD system. If we assume the attacker knows the exact domain of the configuration parameter, Σ_π, in a MTD system, then the exploration space equals the configuration space. But obviously, this assumption is often not the case. For instance, if an attacker is looking for a specific target host in an IPv4 system, the attacker must check each possible IP address, or 256 different addresses (assume a /24 network). However, if the configuration

constraints of the MTD limit the possible IP address of the host to a range of $100 \ldots 255$, the configuration space as defined above does not truly reflect the exploration space of the attacker. As the exploration space is associated with ongoing attacks, we leave its definition until after defining the Cyber Attack Theory.

6.2.7 Diversification

As summarized in Chapter 3, diversification typically has two related concepts. The first refers to the configuration choices already available in the system, while the second refers to techniques used to increase the number of configurations available to the MTD system to use to confuse attackers. Based on the definitions above, we can see that these both refer to the configuration space. The first definition relates to the size of the configuration space while the second concern a variety of techniques to increase the size of the configuration space. To eliminate this confusion of terms, we propose to use the term *artificial diversification* to refer to the second definition.

Definition 6.18. *The* diversification *of an MTD system is the cardinality of its configuration space, or* $|S_v|$*. And the diversification of any configuration parameter,* π*, is simply the cardinality of its domain* $|\Pi|$*.*

Definition 6.19. Artificial Diversification *is a function to increase the configuration space,* S_v*, of an MTD system. Formally, it is denoted as,* $f_d : S_v \rightarrow S'_v$*, where* $|S_v| < |S'_v|$*.*

The four artificial diversity techniques summarized in Section 3.3.1.2 of Chapter 3 can be viewed as examples of this diversification function. As presented in Chapter 1, Kant[45] points out that attack surface modification can be aided by introducing diversity. However, quantitative models for guiding the design of good diversification techniques and assessing their effectiveness remain largely unexplored. Christodorescu *et,al*[29] also indicated that a fundamental challenge in understanding the impact of diversification is to introduce a precise, computationally-meaningful way to measure the increase in difficulty for the attacker.

As shown in the definitions, diversification provides the potential to measure security of an MTD system through configuration space. In an MTD system, a system with a higher diversification is likely to be more difficult to compromise than one of lower diversification, given that all else is equivalent. Artificial diversification seeks to enhance the security provided by an MTD system by introducing functionality equivalent alternatives for a configuration parameter. Diversification determines the size of configuration space, which then provides the space for adaptation. Again, measure and assess the effectiveness of diversification requires Cyber Attacker Theory, which will be presented in the next section.

6.2.8 Randomization

Randomization is another term often used in MTD literature as shown in Chapter 3. Of course, there is nothing in the general understanding of MTD systems that requires random choices, although there are good arguments for this use. The overall goal of randomization is to make full use of the available configuration space introduced by diversification, where a larger configuration space provides more space for adaptation.

In MTD systems, randomization typically refers to choosing random configurations in order to make full use of configuration space while adding the notion of non-predictability. Random choices are usually understood as making selections assuming that the probability of each choice forms a uniform distribution. In this thesis work[32,34], Chapter 4 proposed both a purely random MTD system, which selects the next configuration via a uniform distribution, as well as an *intelligent* MTD system, which also considers potential vulnerabilities and attack alerts when choosing configurations.

In Chapter 3, I suggest to view randomization as a decision making process that chooses the next valid state based on a specific probability distribution over states in S_v. This way, choosing the next configuration from a uniform distribution of states as well as considering alerts become specific instances of randomization. Indeed, given S_v and a specific set of environmental information, if one want to ensure that state $s_i \in S_v$ is chosen, it is simply

define a probability distribution that says the probability of picking s_i is 1. This way, randomization as a decision making process generalizes so that all kinds of AI techniques, such as genetic algorithms, machine learning, game theory, etc. all are just different ways to select the appropriate probability distributions over states S_v. Thus, randomization is defined as follows.

Definition 6.20. Configuration randomization *is a decision making process of selecting the next valid system configuration value* $s \in S_v$ *for* Σ_π. *If* P_j *represents the probability that* s_j *is chosen and* p_j *represents a specific probability value assigned to* P_j *through randomization, then* $\forall s_j \in S_v, 1 \leq j \leq |S_v|$, *we have* $P_j = p_j \wedge 0 \leq p_j \leq 1 \wedge \sum_j p_j = 1$.

6.2.9 Problems

Using the terminology and concepts developed in Section 6.2, this section discusses the implications of these concepts in the MTD process as shown in Figure 6.2. First this section extracts and summarizes several problems which are common to any MTD system that follows from MTD Systems Theory. Then it indicates the next steps towards a theory for moving target defense.

To carry out a process similar to the one described in Figure 6.2, there are three essential problems:

1. How to select the next configuration state of the MTD system.
2. How to select the adaptations to take to get to the next configuration state.
3. When to carry out the adaptations to actually change the state of the system.

Each of these interrelated problems are discussed below.

6.2.9.1 MTD Problem

The essential problem of an MTD system is to move the system from current state to a valid state in such a way as to make an attackers job of compromising the system more difficult.

Thus, the key problem will be deciding what state to move to.

Definition 6.21. *Given the current state, s, of an MTD system, the* MTD problem *is how to choose the next configuration state of the system, s', subject to $s' \in S_v$, to increase the effectiveness of the MTD system.*

The definition makes it clear that only a valid configuration state will be chosen. In addition, measuring effectiveness requires additional theory to define the attacker as well as the relationships as discussed in Section 6.1.4. As seen from the taxonomy, there are several approaches that could be taken including random selection, intelligent selection based on environmental information such as IDS alerts, or cost-based strategies. I believe this to be a prime area of future MTD research.

6.2.9.2 Adaptation Selection Problem

The solution to the MTD problem will lead to a valid next state s' being chosen. However, moving the system to this state requires solving the *adaptation selection problem*, which can be stated informally as finding an adaptation that can transition the system from s to s'.

Definition 6.22. *Given current state of an MTD system, s and a valid next state, s', the* adaptation selection problem *is how to synthesize a sequence of actions $A = \{a_1, a_2, \ldots, a_k\}$ such that $\tau : s \times A \rightarrow s'$. This problem may also consider constraints such as time and costs.*

This problem is analogous to a planning problem and thus future research will likely lead in that direction for the general case. The complexity lies in that there could be multiple sequence of actions that could result in the same configuration state, the optimal solution would require to take constraints such as time and costs into consideration.

6.2.9.3 Timing Problem

The final problem to consider is the MTD Timing Problem, or "when to adapt". Timing is a critical factor in the success of an MTD system. Chapter 6 provides some insight into the

relationship between several key MTD system factors which include the adaptation time interval T_r and attack time interval T_a [119].

Definition 6.23. *In an MTD system, the* Timing Problem *is at what point should the MTD system launch an adaptation to increase the effectiveness of the MTD system, while maintaining a reasonable cost, c.*

Here, reasonable means that the decision of when to launch the adaptation should be made based on the trade off between operation and security. After the Cyber Attack Theory is in place, we will be able to reason over the interactions between an MTD system and an attacker to make such decisions.

6.3 Cyber Attack Theory

Cyber attack success relies on information possessed by an attacker when the attack is launched and is often measured by the information gained or modified as a result of the attack. Thus, information must be an essential element of any theory of cyber attacks. MTD Systems Theory included the notion of a configuration parameter that captures information about the configuration of a computer system, or more generally a target device. This configuration information is clearly part of the information of interest in an attack. However, the information of interest to an attacker goes beyond simple configuration information. For example, a target system's execution status and data are not configuration information, but are critical to many types of attacks. To capture all information of interest, we introduce a new concept called an *information parameter*, where the system's configuration parameters is a subset of the system's information parameters. Thus, a target device is described by a set of information parameters and an attacker expends effort to gain or modify a target device's information parameters. Figure 6.5 highlights this relationship. In addition, an attacker generally has a lot of information that is not necessarily related to the target, which consists of knowledge about other target systems, specialized skills, or general purpose

knowledge.

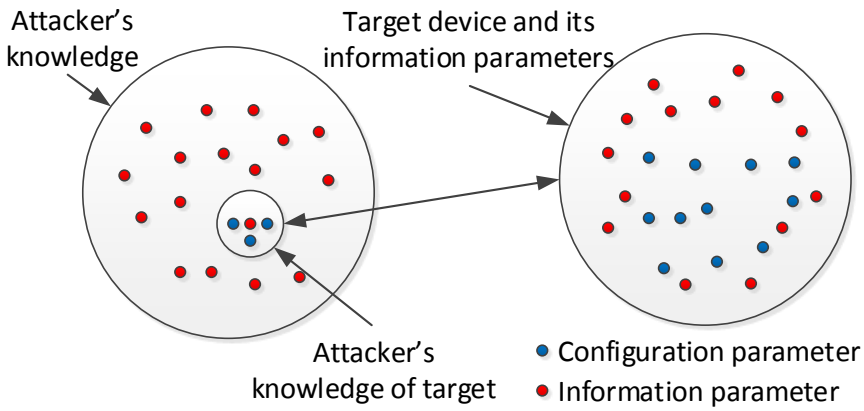


Figure 6.5: Attacker and Target System Overview

In Cyber Attack Theory, we formally define attacks and attacker's knowledge in terms of information parameters. By reasoning over the relationship between an MTD system's configuration parameters and an attack's information parameters, we can formally describe and analyze interactions between attackers and MTD systems. We use the mission planning scenario to help motivate and explain Cyber Attack Theory. In this scenario, attackers try to exploit a vulnerability in the Planner. The vulnerability details are unimportant, but we assume that if the vulnerability exists, the attacker can exploit it. A second scenario, featuring a more concrete code reuse attack and an address space layout randomization (ASLR) MTD, is provided in Section 6.5.2.

6.3.1 Targets

We start our presentation of Cyber Attack Theory by defining the concept of a target, which is based on the concept of information parameters introduced above. We formally define a target and a target system from the attacker's perspective and then build on them by defining concepts associated with attackers and cyber attacks. When we talk about a target, we view it as a device or system of devices that can be described by all kinds of information, such as configuration parameters, execution state and various other information

types. Thus, we start by formally defining an information parameter.

6.3.1.1 Information Parameter

We define an information parameter as a name value pair with an associated type, which defines the domain of possible values an information parameter can assume.

Definition 6.24. *An* information parameter*, $\psi = \langle n, v \rangle$, is a unit of information that can take on a value based on its type, where n is the name and v is a value.*

Definition 6.25. *An* information parameter type*, Ψ, represents the domain of possible values that an information parameter value can assume. We denote the domain of information parameter ψ_j as Ψ_j. An assignment of some value z in Ψ to ψ is denoted as $\psi.v \leftarrow z$.*

An information parameter is basically a variable to which we can assign values from its domain. This is essentially the same definition as that of a configuration parameter. The difference is that a configuration parameter captures information only about the configuration of a device, types of software on that device, specific software settings, etc. Note also that a device's the configuration parameters are a subset of its information parameters. In order to aggregate all the information parameters of a device or system, we define the notion of a composite information parameter.

Definition 6.26. *A* composite information parameter*, ψ, is an information parameter that is composed of a set of sub information parameters, $\psi = \langle n, \{\psi_1, \psi_2, \ldots, \psi_n\} \rangle$, where n is the name, and $\psi_1, \psi_2, \ldots, \psi_n$ are sub information parameters. The domain of a composite information parameter, ψ, is derived from the sub information parameter domains, $\Psi = \Psi_1 \times \Psi_2 \times \ldots \times \Psi_n$.*

For example, the Planner in Figure 6.8 has a set of information parameters that might include $\psi_1 = \langle pageviews,\ 15/d \rangle$, $\psi_2 = \langle visitors,\ 250 \rangle$, $\psi_3 = \langle memory,\ 8GB \rangle$, $\psi_4 = \langle cpu,\ intel\ i5 \rangle$, $\psi_5 = \langle eax,\ 0x65442224 \rangle$, $\psi_6 = \langle os,\ Ubuntu\ 14.04 \rangle$, and $\psi_7 = \langle ip,\ 222.20.22.22 \rangle$.

Here, ψ_1 and ψ_2 are information parameters but not configuration parameters because they are statistically determined by visitors. ψ_5 is an information parameter but not a configuration parameter because the content of EAX reflects the execution status of a program. The remaining information parameters are also configuration parameters.

6.3.1.2 Target

Targets are generally thought of as devices on a computer network that an attacker may compromise, modify or gain information from. However, we also include humans as potential targets because users such as administrators, developers, and clients are also prime targets of attackers.

Definition 6.27. *A* target*, d, is any device an attacker may try to obtain information from, break into, alter or destroy. In cyber space, the targets are computer network devices and communication channels, such as machines (either physical or virtual), humans, routers, switches, cellphones, cables, optical fiber, etc.*

In our example system, the Planner, AssetDB, GeoDB, TargetDB and all users of the mission planning system are potential targets for an attacker. To link the information parameters to the target they represent, we define the predicate *describes*.

Definition 6.28. *If an information parameter, ψ, represents some aspects of the configuration, data or execution state of a device, d, we say ψ describes d, which we denote as the predicate describes(ψ, d).*

In the mission planning example ψ_1 through ψ_7 can all be used to *describe* the Planner. If we have a composite information parameter that captures *all* the relevant information that describes a specific device, we say that composite information parameter is complete as defined below.

Definition 6.29. *Each device d has a unique* complete information parameter ψ_d *whose value captures all the information that describes d and only information that describes d. Formally, this is stated as*

$$\forall d, \exists \psi_d, describes(\psi_d, d) \wedge (\forall \psi, \psi \in \psi_d \Rightarrow describes(\psi, d)) \wedge (\neg \exists \psi, \psi \notin \psi_d \wedge describes(\psi, d))$$

It should be noted that complete information parameters refer to the information parameters *defined* for that device. Obviously, the set of information parameters for a device could be very large, of which only a subset are of practical use as we will see later. Using the notion of a complete information parameter, we define the state of a particular target device or system as the current values associated with each information parameter in the device's complete information parameter.

Definition 6.30. *The* state*, s_d, of target d, is current value of the complete information parameter of d, ψ_d. We can also refer to the state of d at time t.*

Obviously, the state of a target captures the value of ψ_d at a given point in time. This fact will become important when we discuss the effect of attacks in Section 6.3.3.

To judge whether an information parameter exists in a target, a predicate called *exists* is defined. This definition will be used to capture the precondition of an attack type in Section 3.1.

Definition 6.31. *If an information parameter,ψ, can be used to describe a target, d, such that $describes(\psi, d)$ is true, then we say $exists(\psi)$.*

For example, ψ_1 through ψ_7 all can be used to describe the Planner, thus, $exists(\psi_1)$, $exists(\psi_2)$,... $exists(\psi_7)$ are all evaluated as true.

6.3.1.3 Target System

Now that we have defined a target, a target system simply becomes the composition of a set of targets. This essentially allows us to capture the large, complex computer systems that are the target of many attackers today.

Definition 6.32. *A* target system*, D, consists of a set of targets, $D = \{d_1, d_2, \ldots, d_k\}$.*

Definition 6.33. *Each target system, D, has a* system information parameter*, ψ_D, which is the set of each target device's complete information parameter, which is defined as $\psi_D = \{\psi_{d_i} | d_i \in D\}$ and assumes each target's complete information parameter name is unique.*

Combining the Definitions 6.33 and 6.28 allows us to define *describes* for a target system as $describes(\psi_D, D) \Leftrightarrow \forall\ d \in D,\ describes(\psi_d, d)$. The mission planning system (MP) can be viewed as the target $D = \{d_{Planner}, d_{AssetDB}, d_{TargetDB}, d_{GeoDB}\}$ with a system information parameter ψ_{MP} where $describes(\psi_{MP},\ MP)$ is true. In addition, since each target device has a complete information parameter ψ_{d_i}, each device also has a complete information parameter such as $\psi_{Planner}$ where $describes(\psi_{Planner},\ Planner)$. Like the information parameter, the state of the target system is simply a combination of each target device's state.

Definition 6.34. *The* target system state *S_D of target system D is the current value of the system information parameter ψ_D. We also refer to the state of D at time t.*

At this point, we have defined key concepts related to targets and target systems, which in some ways overlaps the definitions in MTD Systems Theory. (MTD systems are also target systems, etc.) However, it is important to note that targets are defined from the attacker's perspective, which captures the fact that the attacker may not know the policies and constraints associated with the system. These details are often the objective of preliminary attacks on the system. As we define attackers and attacks, we show how this information can be gained via attacks.

6.3.2 Attackers

To begin to understand attacks, including how and why they are launched, we must start with the attacker. While an attacker is usually interpreted as an individual person, we extend that notion slightly.

Definition 6.35. *An* attacker*, x, represents a single intruder or team of intruders, where an intruder can be either a human or an automated program.*

Thus, an attacker is not limited to an individual person. Attackers may be groups of people where each is responsible for a part of a coordinated attack. Attackers may also be software programs that automate the intrusion process. As discussed above, for an intrusion to be successful, an attacker must make an effort to investigate the target, which leads to the attacker possessing additional or updated knowledge about the target. We also represent this knowledge as information parameters.

Definition 6.36. *A* knowledge unit *is an information parameter possessed by an attacker. We say attacker x possesses knowledge ψ^x, which includes all the attacker's knowledge units x. If $\psi_1, \psi_2 \ldots \psi_n$ are the knowledge units x possesses, then $\psi^x = \{\psi_1, \psi_2 \ldots \psi_n\}$. Note: attacker's knowledge may not be true; it only represents what the attacker believes to be true.*

Here, we see ψ^x represents all the knowledge an attacker can use to attack a target. If the attacker's knowledge is not sufficient to attack a specific target to achieve the attacker's objective, the attacker will be forced to perform preliminary attacks to gain that knowledge. To specifically talk about an attacker's knowledge about a given target, d, or target system, D, we define ψ_d^x and ψ_D^x.

Definition 6.37. *Attacker, x, has* knowledge*, ψ_d^x, about target, d, where*

$$\psi_d^x = \{\psi | \psi \in \psi^x \wedge \psi \overset{n}{\in} \psi_d\}$$

Similarly, attacker, x, has knowledge, ψ_D^x, about target system, D, where

$$\psi_D^x = \{\psi | \psi \in \psi^x \wedge \psi \overset{n}{\in} \psi_D\} = \bigcup_{d \in D} \psi_d^x$$

The operator, $\overset{n}{\in}$, is used to capture the fact that an attacker's knowledge ψ_d^x and ψ_D^x includes the information parameters that have the *same name* as those in the target's complete information parameter, although their values associated may be different. Formally,

we recursively define $\overset{n}{\in}$ as

$$\psi \overset{n}{\in} \psi' \Leftrightarrow \exists \psi_i \in \psi'.v \;\; s.t.(\psi.n = \psi_i.n \vee \psi \overset{n}{\in} \psi_i)$$

Capturing attacker knowledge supports reasoning about what an attacker knows, what information is gained during an attack, etc. This reasoning will be the key to understanding the affect of MTD adaptation on an attacker's attempt to penetrate or compromise specific targets since MTD adaptation works by invalidating attackers' knowledge of their targets. To know when an attacker's knowledge of a target is valid, we define the predicate *holds*.

Definition 6.38. *If during a time period,* $[t_1, t_2]$*, a logical statement,* l*, defined over* ψ^x *and* ψ_D *is true, we say* $holds(l, [t_1, t_2])$*. If* $t_1 = t_2 = t$*, it simplifies as* $holds(l, t)$*.*

For example, an attacker might have these knowledge about the Planner: $\langle memory, 8GB \rangle$, $\langle cpu, intel\ \ i5 \rangle$, $\langle os, Windows\ \ 8.1 \rangle$, and $\langle ip, 222.20.22.22 \rangle$. However, as defined in Section 6.3.1.1, the true value of the operating system for the Planner is $\langle os, Ubuntu\ 14.04 \rangle$ and thus the attacker's knowledge does not *hold*. Obviously, any attacks against the Planner that assumes a Windows OS will fail.

6.3.3 Attacks

Now that we have defined the concepts of attackers and targets, we turn to defining the attacks themselves. Attacks are a key aspect of our theory as they define the effect of an attacker's interaction with the target system. We define attacks in terms of their affect on system information parameters or attacker knowledge. To help us capture the modification of information, we start by defining an assignment of values between two information parameters.

Definition 6.39. *An* assignment*,* o*, is a tuple of information parameters* $\langle \psi_1, \psi_2 \rangle$*, that*

when executed, copies the value of ψ_2 *into* ψ_1*, which is denoted as* $\psi_1.v \leftarrow \psi_2.v$*. Formally, we define the* execute *operation as* $execute(o) \Leftrightarrow o.\psi_1.v \leftarrow o.\psi_2.v$*.*

Essentially, the execution of an assignment affects only the value associated with the first information parameter. The information parameters themselves do not need to have the same name, even though it is often used to denote the copying of information from a target to an attacker's knowledge or vice versa. From the atttacker's perspective, when an assignment $o = \langle \psi_1, \psi_2 \rangle$, where ψ_2 is a target system information parameter and ψ_1 belongs to attacker's knowledge, it is called a *gain* assignment. In contrast, when ψ_2 belongs to an attacker's knowledge and ψ_1 is a target system information parameter, it is called a *modify* assignment, which implies that the attacker has successfully modified the target system, and the attacker's knowledge of the target system is updated accordingly. For example, a successful intrusion that obtains the IP address of the Planner can be captured by an assignment $\langle \psi^x_{Planner}.ip, \psi_{Planner}.ip \rangle$[1], which sets the Planner's IP address to the address the attacker's has in its IP address information parameter for the Planner. We can also define an intrusion that modifies the system time of the Planner via two assignments $\langle \psi_{Planner}.time, \psi^x_t \rangle$, and $\langle \psi^x_{Planner}.time, \psi^x.t \rangle$, where the first assignment sets the Planner's system time to $\psi^x.t$ (some time the attacker wants to set the system time to) while the second assignment updates the attacker's own knowledge of the Planner's system time.

6.3.3.1 Attack Goals

Based on the previous discussion, we see the *gain assignment* or *modify assignment* actually capture what an attack tries to achieve or the intention behind an attack. In other words, we could use *gain assignment* or *modify assignment* to define the attack goal.

Definition 6.40. *An* attack goal*,* g_a*, captures the attacker's intention to gain or modify*

[1]Here we use the '.' notation to refer to the ip information parameter in the attackers knowledge about the Planner.

the target system information parameter, ψ_D. It can be represented by the gain *or* modify *assignments.*

6.3.3.2 Attack Types

Next, we use assignments to define the post-conditions of attacks, which are defined over ψ_D and ψ^x. These post-conditions can be viewed as the attack goal of corresponding attack. We start by defining an attack type, which is a template for actual attack instance, which we define later.

Definition 6.41. *An* attack type*, ϕ, is a tuple $\langle \Omega_{pre}, \Omega_{post} \rangle$ where Ω_{pre} is a logical statement defined over the target system's information parameter ψ_D and the attacker's knowledge ψ^x, and Ω_{post} is a set of assignments over ψ_D and ψ^x.*

To simplify our discussion, we assume that the logical statements are valid and that the names of the information parameters in ψ^x and ψ_D are unique. In the mission planning system example, assume the attacker, x, has the goal to exploit the Planner to obtain root privileges. To achieve this objective, x considers a sequence of attack types, $\phi = \{\phi_1, \phi_2, \ldots, \phi_6\}$, where the effects of the attacks are as follows:

- ϕ_1 - gains the IP address of the Planner
- ϕ_2 - gains the port number of a specific app
- ϕ_3 - gains the operating system type
- ϕ_4 - obtains an exploitable vulnerability of the app
- ϕ_5 - deploys an exploit agent on the Planner
- ϕ_6 - connects to the agent (e.g., via reverse shell) and gain the root privilege

Table 6.1 shows the specification of these attack types. Each attack type's precondition is a logic statement that explicitly reflects the relationship between an attacker's knowledge of the target and the target's true information. Also, notice that each attack type's precondition depends on the previous attack type's post-condition. For example, ϕ_3 requires that the attacker's knowledge about the target's IP address and the port number is correct. This means the attacker must have a way to gain prior knowledge before the actual attack, which is done via the post-conditions of ϕ_1 and ϕ_2. Also note that the post-condition of ϕ_5 can be viewed as a *modify* assignment while $\phi_1, \phi_2, \phi_3, \phi_4$ and ϕ_6 are all *gain* assignments. This approach not only allows us to explicitly define an attack type based on the relationship between an attacker's knowledge and target system, but it also provides insight into the key information parameters associated with specific attacks and targets. This relationship will be instrumental when we tie Cyber Attack Theory to existing MTD System Theory in order to analyze which configuration parameters can be modified to thwart different types of attacks and to formally define the attack surface for specific types of attacks. We also note that a precondition's logical statement only captures necessary information-based conditions for an attack to succeed. It does not include all the sufficient conditions. We discuss this in more details in Section 6.3.3.3. While the attack specifications in Table 6.1 provide a precise description of individual attacks, attackers generally combine a sequence of low-level attacks to achieve some higher-level objective. To capture this reality, we need to provide the ability to analyze the composition of a set of low-level attack types. However, before defining attack type composition, we define three helper functions: *transform*, *union replace*, and *substitution*.

Definition 6.42. *We define two* transform functions*, ξ_x and ξ_d, that compute a set of information parameters from the assignments in Ω_{post}. $\xi_x()$ extracts information parameters belonging to ψ^x while $\xi_d()$ extracts information parameters that describe target d as defined below*

$$\xi_x(\phi.\Omega_{post}) = \{\langle n, v\rangle | \langle \psi_1, \psi_2\rangle \in \phi.\Omega_{post} \wedge \psi_1 \in \psi^x \wedge n = \psi_1.n \wedge v = \psi_2.v\}$$

$$\xi_d(\phi.\Omega_{post}) = \{\langle n, v\rangle | \langle \psi_1, \psi_2\rangle \in \phi.\Omega_{post} \wedge \psi_1 \in \psi_D \wedge n = \psi_1.n \wedge v = \psi_2.v\}$$

The purpose of the transform function is to extract information parameters contained in Ω_{post}. For example, if we take Ω_{post} from ϕ_1, $\xi_x(\phi_1.\Omega_{post}) = \{\langle ip, \psi_{d_1}.ip.v\rangle\}$, and $\xi_d(\phi_1.\Omega_{post}) = \emptyset$. Similarly, for ϕ_5, $\xi_x(\phi_5.\Omega_{post}) = \{\langle Exa,\ \psi^x.Exa.v\rangle\}$, and $\xi_d(\phi_5.\Omega_{post}) = \{\langle Exa, \psi^x.Exa.v\rangle\}$. To define the union replace function, we first define an operator to determine that an information parameter name does not exist in a set of information parameters or assignments. If $\hat{\psi}$ is a set of information parameters, then we recursively define the operator $\overset{n}{\notin}$ as

$$\psi \overset{n}{\notin} \hat{\psi} \Leftrightarrow \nexists \psi_i \in \hat{\psi},\ s.t.(\psi.n = \psi_i.n \vee \psi \overset{n}{\in} \psi_i.v)$$

And when $\hat{o}$ is a set of assignments, $\overset{n}{\notin}$ becomes

$$o \overset{n}{\notin} \hat{o} \Leftrightarrow \nexists o_i \in \hat{o}, s.t.(o_i.\psi_1.n = o.\psi_1.n \vee o.\psi_1 \overset{n}{\in} o_i.\psi_1.v).$$

Using the $\overset{n}{\notin}$ operator, we now define the union replace function, which updates one set of information parameters based on a second set of information parameters. Essentially, the union replace function replaces the information parameter values in the first set with those of the second set if the names match. Additionally, if information parameters exist in the second set but not the first, these new information parameters from the second set are added to the first set. We also overloaded the union replace operator to work on two sets of assignments as well.

Definition 6.43. *To update one set of information parameters, $\hat{\psi}_1$, based on a second set, $\hat{\psi}_2$, we define a* union replace *function, $\hat{\psi}_1 \uplus \hat{\psi}_2 = \{\psi | (\psi \in \hat{\psi}_1 \wedge \psi \overset{n}{\notin} \hat{\psi}_2) \vee \psi \in \hat{\psi}_2\}$. Likewise, union replace over assignments is defined as $\hat{o}_1 \uplus \hat{o}_2 = \{o | (o \in \hat{o}_1 \wedge o \overset{n}{\notin} \hat{o}_2) \vee o \in \hat{o}_2\}$*

Next, we define a *substitution* function, σ, that substitutes the values from a set of

information parameters into a logical statement. We use σ to substitute the information parameters values from an attacker's knowledge into the corresponding information parameter in the precondition of a given attack type, ϕ.

Definition 6.44. *Given a logical statement, l, and a set of information parameters, $\hat{\psi}$, we define the* substitution *function, $\sigma(l, \hat{\psi})$, as a mapping from names in l to values of information parameters in $\hat{\psi}$ such that the name in l matches the name of the information parameter in $\hat{\psi}$.*

In general, we use the σ function to substitute the values of information parameters in the attacker's knowledge to variable names in Ω_{pre}. This mapping allows us to evaluate the precondition, Ω_{pre}. Using these helper functions, we now formally define a composite attack type. Intuitively, a composite attack type is a sequence of sub attack types. The preconditions of the composite attack type is the conjunction of all the preconditions from the sub attack types that are not satisfied by previous sub attacks. Likewise, the post-condition is the union of the sub attack post- conditions where an assignment to an information parameter later in the sequence takes precedence over assignments to the same information parameter earlier in the sequence.

Definition 6.45. *A* composite attack type*, ϕ, is a sequence of attack types, $\phi = [\phi_1, \phi_2, \ldots, \phi_n]$, where each sub attack type's pre and post-conditions ($\phi_i.\Omega_{pre}$ and $\phi_i.\Omega_{post}$) are defined over a target system's complete information parameter, $\psi_{D(i)}$, and the attacker's knowledge, $\psi^{x(i)}$. The composite attack type's pre and post-conditions are defined as*

$$\phi.\Omega_{pre} = \bigwedge_{1 \leq i \leq n} \sigma(\phi_i.\Omega_{pre}, \psi^{x(i-1)})$$

$$\phi.\Omega_{post} = \biguplus_{1 \leq i \leq n} \phi_i.\Omega_{post}$$

$$where : \psi_{D(0)} = \{\}, \psi^{x(0)} = \{\}$$

$$\psi^{x(i)} = \psi^{x(i-1)} \uplus \xi_x(\phi_i.\Omega_{post})$$

$$\psi_{D(i)} = \psi_{D(i-1)} \uplus \xi_x(\phi_i.\Omega_{post})$$

Note, we use a sequence above to reflect the relationship that the subsequent attack types depend on previous attack types. If two attack types are independent with each other, they should be analyzed individually and no composition is required. While not specifically defined, there is nothing in our theory to limit the analysis of parallel attacks. To simply the discussion, we also define the concept of *minimal composite attack type*. A composite attack type is *minimal* if it only contains the minimum number of necessary sub attack types to achieve the post-conditions defined by an attack goal. Thus, from now on, when talk about the composite attack type, we assume it's minimal and don't consider the situation where redundant attack types being involved and make no contribution to the attack goal defined post-conditions.

For completeness, we define an *atomic attack type* as an attack type that cannot be further decomposed into sub attack types. To demonstrate how pre and post-conditions for composite attack types are computed, we show how are ϕ_1 and ϕ_2 are composed into ϕ in Figure 6.6. Generally speaking, an attack type acts as a template for an actual attack. This relationship is similar to that of an object-oriented class and an object or instance of that class. One attack type can be implemented by many different attacks. For example, attack type ϕ_1 attempts to gain the IP address of the Planner. To implement ϕ_1, an attacker might use automated IP scanning tools, guess the IP address, or obtain it through social engineering. Although these are different attacks, they all implement a same attack type.

6.3.3.3 Attack Instances

Definition 6.46. *An* attack *is a process performed by attacker, x, against target, d, implementing attack type, ϕ, during time period, $[t_s, t_f]$. We denote this attack as $\oint_{t_s}^{t_f}(x, d, \phi)$. Each attack, has a success likelihood against static systems of P_{static} and a duration of $T_a = t_f - t_s$.*

The development of an Cyber Attack Theory will greatly benefit our understanding of

Initialization:

$$\psi^x(0)\{\}, \psi_{D(0)} = \{\}, \Omega_{pre} = \{\}, \Omega_{post} = \{\}$$

compose ϕ_1 :

$$\begin{aligned}
\psi^x(1) &= \psi^x(0) \uplus \xi_x(\phi_1.\Omega_{post}) \\
&= \{\} \uplus \{\langle ip, \psi_{d_1}.ip\rangle\} \\
&= \{\langle ip, \psi_{d_1}.ip\rangle\} \\
\psi_D(1) &= \psi_D(0) \uplus \xi_D(\phi_1.\Omega_{post}) \\
&= \{\} \uplus \{\} \\
&= \{\} \\
\Omega_{pre} &= \sigma(\phi_1.\Omega_{pre}, \psi^x(0)) \\
&= \sigma(\phi_1.\Omega_{pre}, \{\}) \\
&= exists(\psi_{d_1}.ip) \\
\Omega_{post} &= \{\} \uplus \phi_1.\Omega_{post} \\
&= \{\langle \psi^x_{d_1}.ip, \psi_{d_1}.ip\rangle\}
\end{aligned}$$

compose ϕ_2 :

$$\begin{aligned}
\psi^x(2) &= \psi^x(1) \uplus \xi_x(\phi_2.\Omega_{post}) \\
&= \{\langle ip, \psi_{d_1}.ip\rangle\} \uplus \{\langle port, \psi_{d_1}.port\rangle\}) \\
&= \{\langle ip, \psi_{d_1}.ip\rangle, \langle port, \psi_{d_1}.port\rangle\} \\
\psi_D(2) &= \psi_D(1) \uplus \xi_D(\phi_2.\Omega_{post}) \\
&= \{\} \uplus \{\} \\
&= \{\} \\
\Omega_{pre} &= \sigma(\phi_2.\Omega_{pre}, \psi^x(1)) \wedge \Omega_{pre} \\
&= \sigma((\psi^x_{d_1}.ip = \psi_{d_1}.ip \wedge exists(\psi_{d_1}.port)), \\
&\quad \{\langle ip, \psi_{d_1}.ip\rangle\}) \wedge \Omega_{pre} \\
&= exists(\psi_{d_1}.port) \wedge exists(\psi_{d_1}.ip) \\
\Omega_{post} &= \Omega_{post} \uplus \phi_2.\Omega_{post} \\
&= \{\langle \psi^x_{d_1}.ip, \psi_{d_1}.ip\rangle, \langle \psi^x_{d_1}.port, \psi_{d_1}.port\rangle\}
\end{aligned}$$

Figure 6.6: Composition of attack types, ϕ_1 and ϕ_2, into attack type, ϕ.

the interaction between the attacker and MTD system. This will include an understanding of the cost factors related to the attacker and MTD actions, which we believe are closely tied to the duration of the attacks and the impact on the attacker's intrusion success likelihood. Other cost factors, such as attacker effort, are directly related to the time and intrusion success likelihood. However, explicitly including T_a and P_{static} in the definition of the attack does not necessarily mean that we will assign specific values to them. Coming up with real numbers for these factors is hard[46], although there has been work trying to estimate the

mean time-to-compromise[120] and to measure P_{static}[121]. Quantifying T_a and P_{static} is out of the scope of our work. However, we do believe that T_a and P_{static} can be impacted by MTD designers by manipulating MTD system parameters such as the diversification of the configuration space and the adaptation interval. One of our future goals is the development of an analytical model that can inform designers as to how particular parameter settings will impact the effectiveness given attack parameters, such as T_a and P_{static}. Conversely, MTD designers will also be able to judge how effective a given MTD system will be based on various values of T_a and P_{static}.

Definition 6.47. *An* atomic attack *is an attack that implements an* atomic attack type *and cannot be decomposed into sub attacks. An* atomic attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, *is successful with probability* P_{static}, *if and only if its precondition,* Ω_{pre}, *is true from* t_s *to* t_f. *If successful* $\oint_{t_s}^{t_f}(x, d, \phi)$ *ensures execute*(Ω_{post}) *is true precisely at* t_f.

If P_{static} is true, that indicates that all the sufficient conditions for the attack to be successful in a static system, with the exception of those specified in Ω_{pre} are true. However, Ω_{pre} captures those necessary conditions that can be impacted by the MTD system. As long as Ω_{pre} remains true from t_s to t_f and P_{static} is true, the attack will be successful and the attack's post-conditions will be executed at t_f. Like attack types, attacks themselves are generally composed of a sequence of smaller attacks to achieve a larger purpose. We now formally define a composite attack.

Definition 6.48. *An* composite attack *is an attack that implements a composite attack type. Given composite attack type,* ϕ, *that is composed of a sequence of attack types* $[\phi_1, \phi_2, \ldots, \phi_n]$, *a composite attack,* $\oint_{t_s}^{t_f}(x, d, \phi)$, *that implements* ϕ *is composed of a sequence of attacks where each attack,* $\oint_{t_{i-1}}^{t_i}(x, d, \phi_i)$, *implements* ϕ_i *and* $t_0 = t_s \wedge t_n = t_f$. *Formally, this is captured as:*

$$\oint_{t_s}^{t_f}(x, d, \phi) = [\oint_{t_s}^{t_1}(x, d, \phi_1),\ \oint_{t_1}^{t_2}(x, d, \phi_2),\ \ldots,\ \oint_{t_{n-1}}^{t_f}(x, d, \phi_n)]$$

Thus, a composite attack is simply implemented by a sequence of attacks where each attack implements a corresponding sub attack type. Using the attacks defined in Table 6.1, an attack, $\oint_{t_s}^{t_f}(x, d_1, \phi)$, that implements the composite attack type, $\phi = [\phi_1, \phi_2]$, requires the composition of two sub attacks, $\oint_{t_s}^{t_1}(x, d_1, \phi_1)$, that implements ϕ_1 and, $\oint_{t_1}^{t_f}(x, d_1, \phi_2)$, that implements ϕ_2.

6.3.4 Exploration Space

So far, we have introduced two properties of attacks, beside the attack type definition itself, that are critical to analyzing attacks, the attack interval, T_a, and the static likelihood of success, P_{static}. Next, we introduce a third concept that is important to the analysis of attacks and their interactions with MTD systems called the exploration space. Essentially, the exploration space captures the set of possible values an attacker must search in order to find the correct value of a specific information parameter or parameters in order to carry out specific attacks. Figure 6.7 shows an overview of the relationships between an information parameter, ψ's, exploration space, its *configuration space* (as discussed in Section 6.2.6), and the attacker's effort to ascertain ψ's actual value [2]. The effort spent on gaining knowledge through preliminary attacks can be viewed as actions that reduce the attacker's uncertainty about ψ's value from the exploration space down to a single value. For a static system, this uncertainty can be safely assumed to monotonically decreasing with each additional attack. However, with MTD systems, this assumption is invalid. Instead, MTD systems make the attacker's uncertainty non-monotonic. An exhaustive search of the entire exploration space Ψ to identify the correct value of ψ is not the preferred approach. However, there are times when an exhaustive approach are applicable. For example, in the mission planning example, an attack implementing ϕ_1 may scan all possible IP addresses in an IPv4 subnet to obtain

[2] If the information parameter in question is a target's complete information parameter or any other set of information parameters the values are simply tuples of values corresponding to the information parameters in the set.

the correct IP address of the Planner. However, attackers can also use *a priori* knowledge to reduce the search space as well. For example, while knowing that port numbers must be in the range of 0-65535 is of some use in searching $\psi_{Planner}$ for the website port number, knowledge that public facing websites usually use port number 80 may immediately reveal the correct value of $\psi_{Planner}.port$. An attacker can use social engineering to gain required knowledge. For example, an administrator might be fooled into leaking important system information such as IP addresses, operating systems, passwords, etc. No matter which approach is leveraged by attackers, gaining knowledge definitely requires effort on their part to reduce the size of the exploration space. If an information parameter ψ is a configuration

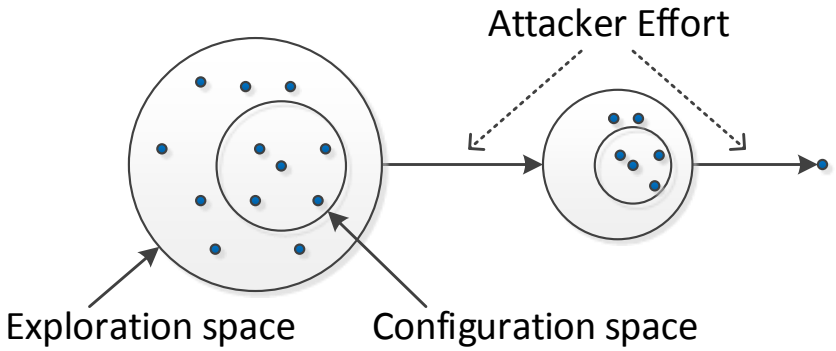


Figure 6.7: Exploration Space Overview (dots are possible values of the information parameter)

parameter of the MTD system as well, the exploration space of ψ may actually be larger than configuration space of ψ. Although a configuration parameter's valid values are typically limited based on system constraints and policies, attackers usually have no way of knowing what these constraints are. For example, constraints internal to the system may require $\psi_{AssetDB}.port$ to be either 43, 53, or 63. However, since attackers would not typically know this information, they would likely be forced to scan the entire range from 0 to 65535. Thus, in general, the exploration space of information parameter ψ equals its domain ψ.

Definition 6.49. *Given information parameter ψ with domain Ψ, the* exploration space *of*

ψ is

$$ESpa_{\psi} = \Psi$$

and the size *of the exploration space is* $|\Psi|$.

Generally speaking, the exploration space is the maximum set of potential values attackers need to investigate in order to obtain the correct value. This concept itself is objective and comes with no specific assumptions about an attacker's capability, skill level, or knowledge about the related target system. Similarly, the exploration space of a target system, D, is simply the domain of Ψ_D, which is the cross product of all ψ_D's sub information parameter domains.

Definition 6.50. *Given a target system D, with its complete information parameter ψ_D with domain of Ψ_D, the* exploration space *of D is defined as:*

$$ESpa_D = \Psi_D$$

Similarly, the size of this exploration space is $|\Psi_D|$.

Although theoretically, this definition gives us an intuition about the exploration space of a target system, it provides little insight to help us understand the exploration space for each individual attack. To do that, we need to define the exploration space of both atomic and composed attack types. To facilitate these definitions, we first define a function to extract the information parameters in an attack type whose value must be gained by an attacker.

Definition 6.51. *Given atomic attack type, $\phi = \langle \Omega_{Pre}, \Omega_{Post} \rangle$, the function δ extracts all information parameters from Ω_{Post} whose value the attacker must gain. Formally, this is*

defined as

$$\delta(\Omega_{Post}) = \{\psi | \forall o \in \Omega_{Post}, o.\psi_1 \in \psi^x \wedge o.\psi_2 \in \psi_D \wedge \psi \leftarrow o.\psi_1\}$$

Based on this function, we define the exploration space of an attack type.

Definition 6.52. *The* exploration space *of attack type,* $\phi = \langle \Omega_{Pre}, \Omega_{Post} \rangle$*, is the cross product of the domain of each information parameter,* $\psi \in \delta(\phi.\Omega_{Post})$*.*

$$ESpa_\phi = \prod_{\psi \in \delta(\phi.\Omega_{Post})} \Psi$$

The size of this exploration space is $\prod_{\psi \in \delta(\phi.\Omega_{Post})} |\Psi|$*.*

This definition requires that for an atomic attack type, ϕ, if there are multiple information parameters in $\delta(\phi.\Omega_{Post})$, attacks implementing ϕ must attempt to gain the value of each of those information parameter simultaneously. Thus, the exploration space is the cross product of its information parameter domains. However, atomic attack types tend to be very simple and usually only attempt to gain the value of a single information parameter. Because a composite attack is made of a set of sub attacks, one might think that the exploration space of the composite attack would not be as large as the cross product of all the information parameters for which it attempts to gain a value. However, this is untrue. Although the exploration space shrinks as sub attacks are successfully completed, as shown in Figure 6.7 as *attacker effort*, the overall exploration space remains the same. Actually, the concept of sub attacks clearly demonstrates the argument that an attacker's effort is actually linear[46] instead of exponential as would be suggested by the cross product operation. Instead of finding all information parameter values simultaneously, a composite attack breaks that down into a series of steps, whose effort is generally small. Thus, if no changes occur to the system configuration, each atomic attack type can be viewed as an attempt to break into a single layer of defense. And, because the effort required to break

into each layer is relatively low, once a layer is penetrated, the next layer is exposed and the attacker has almost unlimited time to attack it. Bellovin[46] claims that what is really needed for system security is an approach that makes the effort expended by the attacker exponential as opposed to linear. Clearly, by constantly adapting the values of the appropriate information parameters, MTD systems could eliminate that brittleness. Attackers can no longer assume that they can ascertain the value of each information parameter one at a time, but will effectively need to learn, and potentially relearn them all in a very short time frame, which pushes the attackers effort towards the exponential.

6.4 MTD Theory

Enabled by MTD System Theory and Cyber Attack Theory, we are able to start formally defining the MTD Theory. The objective of MTD Theory is to define how elements of the MTD Systems and Cyber Attacks theories interact. This step is especially important in being able to understand the true effect of an MTD system as its effectiveness only makes sense in light of actions from an attacker for a specific attack type[122,115].

Recall from Chapter 1 that a major challenge to understand moving target defenses is the need for new metrics. Existing metrics for attack surface areas are not suitable for evaluating a moving attack surface because two basic assumptions of the existing metrics have been broken. One is that the attack surface remains unchanged, while the other is that the target attack surface is always reachable by attackers. Thanks to the MTD System Theory and Cyber Attack Theory, we are able to relax these two assumptions by introducing a new definition of *attack surface* based on a specific attack type. In addition, this section also introduces two new concepts, the *adaptation surface*, which captures all of the information parameters adapted by an MTD system, and the *engagement surface*, which captures the information parameters adapted by an MTD system that can be *potentially effective* to

thwart an attack type. Based on these new concepts, several metrics, such as *coverage*, *potential effectiveness* and *success likelihood of intrusion*, will be introduced. These metrics will greatly benefit the interaction analysis between the attacker and MTD System, and help quantify the effectiveness of MTD. Based on these new definitions, serveral theorems can be derived. These theorems provides fundamental guidelines for MTD system design, analysis and parameter setting. Lastly, this section discusses attack effort indicators and how different MTD system parameter settings would impact the attack effort.

6.4.1 Attack Surface

Previous definitions of attack surface[3,4,31] suffer from its inability to capture the dynamic and changing nature of MTD, plus the potential state and action space explosion when trying to capture it from the whole system perspective. To solve these problems, a new attack surface definition is proposed in this section. As it defined based on information parameters, we can formally talk about the different state of an attack surface at different time point. As it only talks about the attack surface in terms of a set of information parameters related to a specific attack, it avoids the explosion issue and encourages the MTD designers to focus on the most critical information that could be used to thwart an attack type.

The δ function introduced in Definition 6.51 can be used to extract target system information parameters from Ω_{post}, we overload the δ function and define it to also extract system information parameters from Ω_{Pre} to help define the attack surface.

Definition 6.53. *Given atomic attack type, $\phi = \langle \Omega_{Pre}, \Omega_{Post} \rangle$, the function δ also extracts all system information parameters from Ω_{Pre}. Formally, this is defined as*

$$\delta(\Omega_{Pre}) = \{\psi | \forall \psi \in \Omega_{Pre}, \psi \in \psi_D\}$$

Once how to extract system information parameters from an attack type is in place, the

attack surface for an attack type is simply the set of these system information parameters.

Definition 6.54. *Given an attack type ϕ, the* attack surface *of ϕ, $S_{attack}(\phi)$, equals the union of $\delta(\phi.\Omega_{Pre})$ and $\delta(\phi.\Omega_{Post})$. Formally, this is defined as*

$$S_{attack}(\phi) = \delta(\phi.\Omega_{Pre}) \cup \delta(\phi.\Omega_{Post})$$

As an information parameter ψ is a name value pair, that can take on various values at different times. This enables us to formally talk about the state of an attack surface at different times. In addition, as the attack surface is defined in terms of a set of information parameters related to a specific attack, it avoids the explosion issue and encourages the MTD designers to focus on the most critical information that could be used to thwart an attack type. Concrete examples of attack surfaces are given in Section 6.5.

6.4.2 Adaptation Surface

From attacker's perspective, the attack surface describes the system information parameters involved in an attack that an attacker needs to gain or modify. While knowing this information is very helpful, from defender's perspective, it doesn't mean all the information parameters can be adapted due to various constraints, such as the balance between security and operation, the labor of development or the cost of maintenance. Thus a new concept, called *adaptation surface*, is introduced to capture only those system information parameters that actually adapted. Thus, we eliminate information parameters whose domain contains only one value.

Definition 6.55. *Given an MTD system, Σ, and its minimum complete configuration parameter, Σ_π, the* adaptation surface *of Σ, $S_{adapt}(\Sigma)$, is a set of configuration parameters*

whose domain contain more than one value. Formally, this is defined as

$$S_{adapt}(\Sigma) = \bigcup_{\pi_i \in \Sigma_\pi}^{n} \pi_i, where\ |\Pi_i| > 1$$

The adaptation surface clearly captures which configuration parameters are being adapted by an MTD system and which are not. We provide concrete examples of adaptation surface in Section 6.5.

6.4.3 Engagement Surface

Once the attack surface of an specific attack type and the adaptation surface of an MTD system are defined, we can derive which information parameter adapted by an MTD system can be used to thwart the specific attack type. We call these information parameters the *engagement surface*, inspired by military terminology where the engagement of attacker and a defender occurs.

Definition 6.56. *The* engagement surface *between the attack surface of an attack type, ϕ, and adaptation surface of an MTD system, Σ, contains all the configuration parameters an MTD system uses to thwart that attack type. Formally, this is defined as*

$$S_{engage}(\phi, \Sigma) = S_{attack}(\phi) \cap S_{adapt}(\Sigma)$$

Clearly an empty engagement surface, $S_{engage}(\phi, \Sigma)$, indicates there is no MTD-enabled protection against ϕ. An MTD system, Σ, can only impact ϕ when the engagement surface is not empty. Based on these concepts, we develop metrics which can quantify the effectiveness of MTD in the following. Concrete examples of engagement surfaces are provided in Section 6.5.

6.4.4 Coverage

Concepts like *attack surface*, *adaptation surface* and *engagement surface* enable us to formally talk about the interaction between a specific attack type and an MTD system. However, the ultimate goal of having these formal definitions is to formally define useful metrics that can measure the effectiveness of an MTD as well as to derive useful theorems that guide the design and implementation of an MTD system.

Starting from this section, several metrics will be introduced. The first one describes the *coverage* of an MTD system, Σ, against an attack type, ϕ, which measures the percentage of information parameters that are part of the attack surface of ϕ but also belong to the adaptation surface of Σ. Again, concrete examples of these metrics are provided in Section 6.5.

Definition 6.57. *The* coverage *of an MTD system, Σ, verses an attack type, ϕ, is the number of information parameters contained in $S_{engage}(\phi, \Sigma)$ divided by the number of information parameters contained in $S_{attack}(\phi)$. Formally, this is defined as*

$$Coverage(\phi, \Sigma) = \frac{|S_{engage}(\phi, \Sigma)|}{|S_{attack}(\phi)|}$$

6.4.5 Potential Effectiveness

Coverage, as a quantified value based on the engagement and attack surfaces, provides a preliminary measurement of an MTD system's effectiveness, which we call the *potential effectiveness*.

Definition 6.58. *We define the* potential effectiveness *of an MTD system, Σ, against an attack type, ϕ, in terms of coverage. Clearly, the* potential effectiveness *has a value range [0, 1]. A zero value means Σ has no impact to ϕ, a one value only means Σ is potentially effective for thwarting ϕ.*

According to the definition, a zero potential effectiveness of ϕ means there is no engagement surface between an MTD system, Σ and a specific attack type, ϕ. Thus, for attack type, ϕ, Σ provides no extra protection benefit when compare to a static system. On the other hand, a higher coverage value indicates a higher potential effectiveness, but does not mean that the *actual* effectiveness is also better. For example, a potential effectiveness of one means the MTD system, Σ, covers all of the information parameters contained in the attack surface of ϕ. However, if Σ takes an extremely long time to actually adapt, then the *actual* effectiveness is no better than an static system. After all, the effectiveness of an MTD system depends on many factors, while the coverage is only one of these factors. But a coverage of zero indeed provides a clear indication of the non effectiveness of an MTD against a particular attack type.

To analyze and quantify the MTD effectiveness when its coverage is greater than zero, we introduce the success likelihood of intrusion.

6.4.6 Success Likelihood of Intrusion

Success likelihood reflects uncertainty, which is a good fit in a cyber security context where the success of intrusion depends on many factors. However, as discussed in Chapter 1, measuring security is hard, if not impossible[46]. Although we start to define the success likelihood of intrusion in this section, the goal is not to quantify and obtain the absolute value, as this is still a major challenge. Instead, we show that the success likelihood of intrusion under MTD depends on several conditions, and part of these conditions can be quantified nicely. This quantification allows us to analyze relative success likelihood of intrusion as compared to static systems.

Recall from section 6.3.3.3, that the Ω_{pre} of an attack type captures the necessary conditions of an attack that can be impacted by the MTD system. While P_{static} captures all the sufficient conditions for the attack to be successful in a static system, with the exception of

those specified in Ω_{pre}. In this section, we show how the uncertainty impacted by Ω_{pre} can be quantified, while leaving the quantification of P_{static} as future work.

Before giving the formal definition of success likelihood of intrusion, we first define a predicate, $unchanged(\hat{\psi}, [t_1, t_2])$, which captures the fact that the value of an information parameter does not change.

Definition 6.59. *If during a time period, $[t_1, t_2]$, for a set of information parameter, $\hat{\psi} = \{\psi | \psi \overset{n}{\in} \psi_D\}$, the value of each $\psi \in \hat{\psi}$ keep unchanged, we say $unchanged(\hat{\psi}, [t_1, t_2])$.*

Together with *holds* as given in Definition 6.38, we define the success likelihood of an attack instance.

Definition 6.60. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, that implements an attack type, $\phi = \langle \Omega_{pre}, \Omega_{post} \rangle$, the success likelihood of intrusion of this attack under MTD can be defined as:*

$$P_{success}(\oint) = P(holds(\Omega_{pre}, [t_s, t_f])) \times P_{static}$$

where :

$$P(holds(\Omega_{pre}, [t_s, t_f])) = P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f])) \times P(holds(\Omega_{pre}, t_s))$$

Thus, for an attack instance to be successful in attacking an MTD system, not only does Ω_{pre} need to hold during the attack interval, but all the other conditions captured by P_{static} also needs to be true. In addition, it turns out $P(holds(\Omega_{pre}, [t_s, t_f]))$ can be further decomposed into two parts, $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ and $P(holds(\Omega_{pre}, t_s))$, which means for Ω_{pre} to be hold during the whole attack interval, $[t_s, t_f]$, we only need to analyze whether Ω_{pre} is true at the begining of intrusion, plus whether all the system information parameters involved in Ω_{pre} is changed or not during the attack interval, $[t_s, t_f]$. The failure of either part will lead to an unsuccessfull intrusion. Conversely, if both parts are

fulfilled, P_{static} also needs to be true for $P_{success}(\oint)$ to be true. Concrete examples that show how to quantify $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ and $P(holds(\Omega_{pre}, t_s))$ will be provided in Section 6.5.

6.4.7 Theorems

As discussed, the ultimate goal of having an MTD theory is to formally define useful metrics that help analyze and quantify the effectiveness of an MTD, and to derive useful theorems that could guide the design and implementation of an MTD system. Theorems provide intuitive conclusions that can be used by MTD system designers to decide how to use existing configuration choices and diversification to increase security.

Before presenting theorems, we first introduce what constitutes the *characteristics* of an MTD system and an attack. These *characteristics* represent the key factors that impact the effectiveness of an MTD system against a specific attack type. We will use these characteristics when describing the conditions under which the theorems are satisfied.

Definition 6.61. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, that implements an attack type, $\phi = \langle \Omega_{pre}, \Omega_{post} \rangle$, and an MTD system, Σ, the* characteristics *of Σ and ϕ include $S_{attack}(\phi)$, $S_{engage}(\phi, \Sigma)$, $Coverage(\phi, \Sigma)$, the attack interval T_a, the adaptation interval T_r, and an algorithm that controls how each configuration parameter in $S_{engage}(\phi, \Sigma)$ is adapted.*

These characteristics captures the most important factors involved in analyzing the effectiveness of an MTD system against a specific attack type. These factors are closely related and in the following theorems, we will see how each factor will impact the effectiveness in terms of $P_{success}(\oint)$ when fixing other characteristics.

Theorem 6.1. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, against an MTD system, Σ, if all other characteristics in Definition 6.61 hold during the attack from t_s to t_f, a smaller T_r leads to a lower $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. Conversely, a larger T_r leads to higher $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$.*

Proof. For a given T_r and attack interval, $T_a = [t_s, t_f]$, where t_s could be any time point, there are $\frac{T_a}{T_r}$ times of MTD adaptation could occur. A reduction in T_r will increase $\frac{T_a}{T_r}$, which means more times of MTD adaptation could occur during $[t_s, t_f]$, thus $unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f])$ is less likely to be hold, which means the probability $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ is decreased. According to Definition 6.60, the success likelihood of intrusion is proportional to $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$, thus a decreased $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ leads to a lower $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. Using a similar argument, we can prove that a larger T_r leads to a higher $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. □

Theorem 6.2. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, against an MTD system, Σ, if all other characteristics in Definition 6.61 hold during the attack from t_s to t_f, a smaller T_a leads to a higher $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. Conversely, a larger T_a leads to a lower $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$.*

Proof. For a given T_r and attack interval, $T_a = [t_s, t_f]$, where t_s could be any time point, there are $\frac{T_a}{T_r}$ times of MTD adaptation could occur. A reduction in T_a will decrease $\frac{T_a}{T_r}$, which means less times of MTD adaptation could occur during $[t_s, t_f]$, thus $unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f])$ is more likely to be hold, which means the probability $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ is increased. According to Definition 6.60, the success likelihood of intrusion is proportional to $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$, thus an increased $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ leads to a higher $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. Using a similar argument, we can prove that a larger T_a leads to a lower $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$. □

Theorem 6.3. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, against an MTD system, Σ, if all other characteristics in Definition 6.61 hold during the attack from t_s to t_f, adding more configuration parameters into $S_{engage}(\phi, \Sigma)$, increases $Coverage(\phi, \Sigma)$ and will not increase $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$.*

Proof. According to Definition 6.57, $Coverage(\phi, \Sigma) = \frac{|S_{engage}(\phi,\Sigma)|}{|S_{attack}(\phi)|}$, thus adding more configuration parameters into $S_{engage}(\phi, \Sigma)$ increases $|S_{engage}(\phi, \Sigma)|$ which in turn increases

$Coverage(\phi, \Sigma)$. If there are more configuration parameters contained in $Coverage(\phi, \Sigma)$, it means more configuration parameters could be changed during t_s to t_f, thus the probability of $P(unchanged(\delta(\phi.\Omega_{pre}), [t_s, t_f]))$ is going to decrease or remain unchanged, as such, $P_{success}(\oint_{t_s}^{t_f}(x, d, \phi))$ will not increase. □

Lemma 6.1. *Given a configuraton parameter, π, a larger configuration space of π, $CSpa_\pi$, leads to non smaller exploration space of π, $ESpa_\pi$.*

Proof. Because $CSpa_\pi = \Pi_\pi$, where $\forall s \in \Pi_\pi, s$ is valid. On the other hand, $ESpa_\pi = \Psi_\pi$, where Ψ_π is the domain of π and $\forall s \in \Psi_\pi$, s can be either valid or invalid, then $|ESpa_\pi| \geq |CSpa_\pi|$. Thus increasing configuration space of π leads to a non-smaller exploration space. □

Lemma 6.2. *Given an information parameter, ψ, if an attacker needs to spend a finite amount of time, t_i, on each value of ψ in order to determine the true value, a larger exploration space of ψ leads to a longer mean time of compromise, T_a.*

Proof. As the size of the exploration space of ψ is $|ESpa_\psi|$, if an attacker needs to spend time t_i on each value, then the total time spend on investigating all the values is $\sum_{i=1}^{i=|Espa_\psi|} t_i$, then the mean time of compromise will be $T_a = \frac{1}{2} \times \sum_{i=1}^{i=|Espa_\psi|} t_i$. Thus T_a is monotonically increasing when the size of the exploration space of ψ increases, hence increasing exploration space of ψ will increase the mean time of compromise, T_a. □

Note, the lemma doesn't apply to situations where attacker don't need to try all possible values because of the behavior of the user. There are situations where a larger exploration space could result in less time of compromise when the attacker doesn't needs to try each possible value. Examples include social engineering attacks and man-made errors. In our analysis of MTD systems, we assume people behave properly.

Theorem 6.4. *Given an attack, $\oint_{t_s}^{t_f}(x,d,\phi)$, against an MTD system, Σ, then if all other characteristics in Definition 6.61 holds, if $\forall\psi_i \in S_{engage}(\phi,\Sigma)$, the attacker spends a finite amount of time t_{ij} on each state of ψ_i, then increasing the configuration space of each $\psi_i \in S_{engage}(\phi,\Sigma)$ leads to non-increased $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$.*

Proof. If $\forall\psi_i \in S_{engage}(\phi,\Sigma)$, the attacker spends a finite amount of time, t_{ij}, on each state of ψ_i, then the mean time of compromise $T_a = \sum_{i=1}^{i=|S_{engage}(\phi,\Sigma)|}(\frac{1}{2}\times\sum_{j=1}^{j=|ESpa_{\psi_i}|}t_{ij})$. Based on lemma 1, increasing configuration space of each $\psi_i \in S_{engage}(\phi,\Sigma)$ leads to non-smaller exploration space of ψ_i, $|ESpa_{\psi_i}|$. As $T_a = \sum_{i=1}^{i=|S_{engage}(\phi,\Sigma)|}(\frac{1}{2}\times\sum_{j=1}^{j=|ESpa_{\psi_i}|}t_{ij})$, thus a non-smaller $|ESpa_{\psi_i}|$ will leads to a non-smaller overall T_a. Then according to Theorem 6.2, a larger T_a reduces $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$, thus a non-smaller T_a leads to a non-increased $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$. □

Theorem 6.5. *Given an attack, $\oint_{t_s}^{t_f}(x,d,\phi)$, against an MTD system, Σ, then if all other characteristics in Definition 6.61 holds, if $\forall\psi_i \in S_{engage}(\phi,\Sigma)$, the attacker spends a finite amount of time t_{ij} on each state of ψ_i, then increasing the exploration space of each $\psi_i \in S_{engage}(\phi,\Sigma)$ will reduce $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$.*

Proof. Based on lemma 6.2, a larger exploration space of ψ_i leads to a longer mean time compromise of each ψ_i, thus the overall T_a will be increased. Then according to Theorem 6.2, a larger T_a reduces $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$, thus increasing the exploration space of each $\psi_i \in S_{engage}(\phi,\Sigma)$ will reduce $P_{success}(\oint_{t_s}^{t_f}(x,d,\phi))$. □

6.4.8 Attack Effort

Generally speaking, attack effort is hard to define and quantify. There are many factors, such as attacker skill level, attack time, available resources, or network bandwidth, which can be used to indicate the attack effort. Unfortunately, these factors are also usually hard to know. As such, the goal of this section is not to define and quantify attack effort, but

instead to propose a potential attack effort indicator based on the expected number of attack trials to first achieve a certain success likelihood of intrusion.

Before defining attack effort indicator, we first present an theorem about the number of trials until first success. Interested users could find more information about the proof of theorem in[123,124].

Theorem 6.6. *Let V be an event that occurs in a trial with probability, p. Mathematical expectation, E, of the number of trials to first occurrence of V in a sequence of trials is $E = 1/p$.*

As discussed, it's hard to define attack effort as there are many unknown factors. However, we do believe the number of attack trials can be part of all possible attack effort indicators and we define the attack effort indicator as follows.

Definition 6.62. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, against an MTD system, Σ, an attack effort indicator can qualitatively indicate the relative amount of effort spent on conducting this attack until success. In this thesis, we focus on the number of attack trials until the first success of $\oint_{t_s}^{t_f}(x, d, \phi)$, and a larger number of attack trials leads to a higher attack effort.*

Based on Theorem 6.6 and Definition 6.62, we derive another theorem that states the relationship between success likelihood of intrusion and attack effort.

Theorem 6.7. *Given an attack, $\oint_{t_s}^{t_f}(x, d, \phi)$, against an MTD system, Σ, a lower success likelihood of intrusion, $P_{success}(\oint)$, leads to a higher effort of the first success of $\oint_{t_s}^{t_f}(x, d, \phi)$.*

Proof. According to Theorem 6.6, the lower success likelihood of intrusion, $P_{success}(\oint)$ leads to larger number of trials to first occurance of successful $\oint_{t_s}^{t_f}(x, d, \phi)$. Based on Definition 6.62, a larger number of attack trials leads to a higher attack effort. □

6.4.9 Relationships between MTD system Parameters and Attack Effort

Based the theorems proved in Section 6.4.7, we know larger T_a, less T_r, and a larger exploration space $Espa_\phi$ can reduce the success likelihood of intrusion. In addition, a larger $Coverage(\phi, \Sigma)$ and larger configuration space of $S_{engage}(\phi, \Sigma)$ will not increase the success likelihood of intrusion. Then from the previous section we know that larger T_a, less T_r and larger exploration space $Espa_\phi$ lead to a higher attack effort, and that larger $Coverage(\phi, \Sigma)$ and larger configuration space of $S_{engage}(\phi, \Sigma)$ will not reduce attack effort.

In short, we can define the following relationships between MTD system parameter settings and attack effort.

- Larger T_a, higher attack effort.
- Less T_r, higher attack effort.
- Larger $Coverage(\phi, \Sigma)$, non-decreased attack effort.
- Larger configuration space of $S_{engage}(\phi, \Sigma)$, non-decreased attack effort.
- Larger exploration space of $Espa_\phi$, higher attack effort.

By introducing attack effort indicator, we connect the success likelihood of intrusion with attack effort, which provides further insight into how various MTD parameters impact attack effort. These relationships provide MTD designers a clear understanding about the effect of moving target defense and gives them powerful theoretical insights about how MTD will impact the success likelihood of intrusion as well as attack effort. Armed with these guidelines, MTD designers will be able to make more confident trade-off decisions when designing and implementing MTD systems.

6.5 Validation

Until now, we have developed a theory of moving target defenses. However, no complete examples have been provided to show how things work together and why this theory would benefit design and analysis when implementing an MTD system. While a formal and full validation of the theory is out of the scope for this thesis, this section starts the validation process by introducing concrete scenarios and showing examples of how the concepts defined in the theory are instantiated and how this theory can be used to analyze and quantify the effectiveness of an MTD system.

Specifically, there will be two examples provided. The first example describes an attack that targets a mission planning system. It provides concrete specifications for the attack as well as MTD system. These specifications are based on the definitions in Cyber Attack Theory and the MTD System Theory. Then, by taking advantage of MTD Theory, we show how to analyze the potential effectiveness of this MTD system against the given attack as well as quantify the success likelihood of intrusion. A concrete discussion of how various parameter settings would impact the effectiveness of MTD system in terms of success likelihood is also given.

The first example focuses on analyzing an MTD system that only adapts one configuration parameter (IP address) invovled in the attack surface. To see how more than one configuration parameters being adapted impacts the given attack type, the second example extends the first MTD system to enable address space layout randomization (ASLR) where the memory address is also adapted. Again the second example provides concrete specifications for the attack as well as the MTD system based on the definitions in Cyber Attack Theory and MTD System Theory. We also analyze the coverage as well as its potential effectiveness based on MTD Theory. In addition, by adopting two adaptation mechanisms, we show how different adaptations will impact the success likelihood of intrusion. By enabling

more than one configuration parameter in the second example, we provide a more complete example to show how to analyze and quantify MTD system's potential effectiveness and success likelihood of intrusion. It also provides a more complete view and understanding about what an MTD system can do to thwart a given attack type.

6.5.1 Attack Mission Planning System

We start with the simple military mission planning system, which is shown in Figure 6.8. Authorized users can remotely access the mission planner to construct military type missions. The Planner (a web server with a user interface) allows users to carry out authorized actions, such as adding new strategies, establishing plans/tactics or allocating resources. To support these actions, the Planner accesses three associated databases – the AssetDB, GeoDB and TargetDB. In this scenario, attackers try to exploit a vulnerability in the Plan-

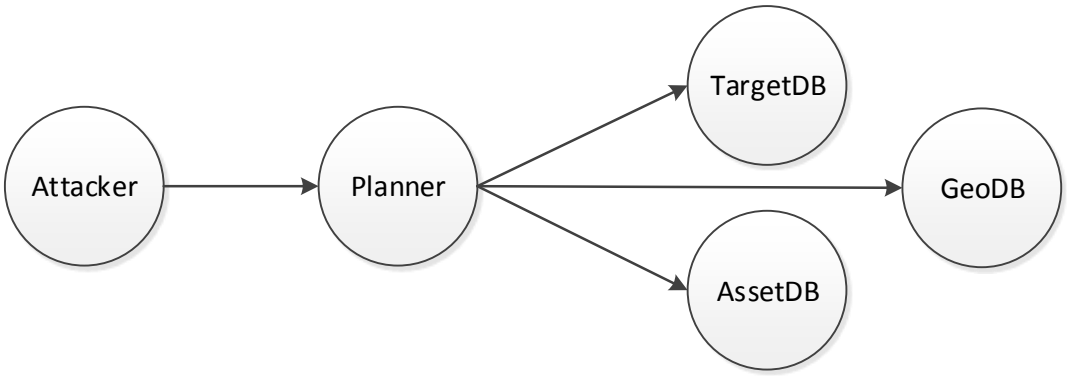


Figure 6.8: Motivating Attack Scenario

ner. The vulnerability details are unimportant here, but we assume that if the vulnerability exists, the attacker can exploit it. A more detailed vulnerability and realistic attack will be provided in the next example.

6.5.1.1 MTD System Specification

The mission planning system has the following settings:

The Planner:

- Runs a C++ web application based on Apache and Ubuntu.
- Web application has port number 80.
- Adaptation contains actions to replace the old Planner machine with a new machine and notify the changes to the AssetDB, GeoDB and TargetDB.
- New Planner is configured with the same C++ web app, port and OS.
- New Planner's IP address is picked from 192.168.10.100–192.168.10.199.
- Adaptation occurs in every time interval T_r.
- Every T_r, the Planner gets refreshed in a probability of p_r.

As an example, here we assume $p_r = 1/4$. The AssetDB, GeoDB and TargetDB remain unchanged and only update related configuration when the Planner is refreshed.

Configuration Space Because IP address is the only configuration parameter that is changed in this system, the configuration space has the size of 100. Later, when discuss the ASLR attack example, we will show that this configutaton space can be enlarged by enabling the address space layout randomization.

Diversification The adaptation in this mission planning system takes advantage of the IP address space. No artificial diversification technique is used in this example.

Randomization When adaptation occurs, the new machine's IP address is randomly picked from the IP pool, except for the previous IP address, thus each IP address has the probabiliy of $p_s = 1/99$ to show up and the old IP address has probability of $p_s = 0$ to be choosen. In addition, for every T_r, there is a probability of $p_r = 1/4$ the Planner actually gets

refreshed. Thus, the current selected IP address has the probability of $(1 - p_r) = \frac{3}{4}$ to remain unchanged.

MTD Problem In this concrete example, the solution to the MTD problem is to replace the Planner in every T_r with a probability of $p_r = 1/4$. After refreshing, the new Planner will have the same configuration as before except for the IP address, which is randomly picked from the IP pool.

Adaptation Problem In this concrete example, the solution to the adaptation problem is to synthesize a sequence of configuration actions to shutdown the old Planner machine, start and configure a new Planner, and then notify the IP address change of the Planner to the AssetDB, GeoDB and TargetDB machines.

Timing Problem In this example, the solution to the timing problem is to simply schedule the adaptation in every time interval, T_r. Notice here T_r does not need to be a constant.

6.5.1.2 Attack Specification

6.5.1.2.1 Attack Goal In this example, an attacker, x, has the goal to exploit the Planner to obtain root privileges. We will extend and analyze a more realistic attack that will try to steal data from a database server in the next example.

6.5.1.2.2 Target System The mission planning system in this example is the target system, D, with a complete information parameter, $\psi_{MissionPlanning}$, and the Planner, $d_{Planner}$ (or d_P for short), as the specific target of interest, which has its complete information parameter, $\psi_{Planner}$. Concrete information parameters that will be used in this example include $\psi_{d_P \cdot ip}$, $\psi_{d_P \cdot apache\_port}$, $\psi_{d_P \cdot os}$, $\psi_{d_P \cdot vul}$

6.5.1.2.3 Attacker To carry out this attack successfully, the attacker, x, must have correct knowledge of the Planner, $\psi^x_{d_P}$, including the Planner's IP address, Apache web server port number, operating system, and the vulnerability of the web server. Formally,

these are captured via attacker knowledge in the form of information parameters, such as $\psi^x_{d_P}.ip$.

6.5.1.2.4 Attack Type To achieve the objective, x considers a sequence of attack types, $\phi = [\phi_1, \phi_2, \ldots, \phi_6]$, where the effects of the attacks are as follows:

- ϕ_1 - gains the IP address of the Planner, $\psi_{d_P}.ip$
- ϕ_2 - gains the port number of a specific app, $\psi_{d_P}.apache\_port$
- ϕ_3 - gains the operating system type, $\psi_{d_P}.os$
- ϕ_4 - obtains an exploitable vulnerability of the app, $\psi_{d_P}.vul$
- ϕ_5 - deploys an exploit agent on the Planner, $\psi_{d_P}.exa$
- ϕ_6 - connects to the agent (e.g., via reverse shell) and gains the root privilege, $\psi_{d_P}.root$.

Table 6.1 shows the specification of these attack types.

Table 6.1: Attack Type Specification

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
ϕ_1	$exists(\psi_{d_P}.ip)$	$\langle \psi^x_{d_P}.ip, \psi_{d_P}.ip \rangle$
ϕ_2	$\psi^x_{d_P}.ip = \psi_{d_P}.ip \wedge exists(\psi_{d_P}.apache\_port)$	$\langle \psi^x_{d_P}.apache\_port \rangle$, $\langle \psi_{d_P}.apache\_port \rangle$
ϕ_3	$\psi^x_{d_P}.ip = \psi_{d_P}.ip \wedge \psi^x_{d_P}.apache\_port = \psi_{d_P}.apache\_port \wedge exists(\psi_{d_P}.os)$	$\langle \psi^x_{d_P}.os, \psi_{d_P}.os \rangle$
ϕ_4	$\psi^x_{d_P}.ip = \psi_{d_P}.ip \wedge \psi^x_{d_P}.apache\_port = \psi_{d_P}.apache\_port \wedge \psi^x_{d_P}.os = \psi_{d_P}.os \wedge exists(\psi_{d_P}.vul)$	$\langle \psi^x_{d_P}.vul, \psi_{d_P}.vul \rangle$
ϕ_5	$\psi^x_{d_P}.ip = \psi_{d_P}.ip \wedge \psi^x_{d_P}.apache\_port = \psi_{d_P}.apache\_port \wedge \psi^x_{d_P}.os = \psi_{d_P}.os \wedge \psi^x_{d_P}.vul = \psi_{d_P}.vul$	$\langle \psi_{d_P}.exa, \psi^x.exa \rangle$, $\langle \psi^x_{d_P}.exa, \psi^x.exa \rangle$
ϕ_6	$\psi^x_{d_P}.ip = \psi_{d_P}.ip \wedge \psi^x_{d_P}.apache\_port = \psi_{d_P}.apache\_port \wedge \psi^x_{d_P}.exa = \psi_{d_P}.exa \wedge exists(\psi_{d_P}.root)$	$\langle \psi^x_{d_P}.root, \psi_{d_P}.root \rangle$

6.5.1.2.5 Attack Instances Concrete attacks that implements attack type $\phi_1 - \phi_6$ are attack instances.

6.5.1.3 Interaction Analysis

In this section, we analyze the interactions between $\phi_1, \phi_2, \phi_3, \phi_4$ and the mission planning MTD system. With a concrete MTD system setting, We show the attack surface, adaptation surface and engagement surface, we also show how to quantify coverage and success likelihood of intrusion. In the next example, We will extend this analysis to ϕ_5, ϕ_6.

6.5.1.3.1 Attack Surface Each attack type has its own attack surface.

- $S_{attack}(\phi_1) = \{\psi_{d_P \cdot ip}\}$
- $S_{attack}(\phi_2) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}\}$
- $S_{attack}(\phi_3) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \}$
- $S_{attack}(\phi_4) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot vul}\}$

If we compose $\phi_1, \phi_2, \phi_3, \phi_4$, we get compositional attack type, $\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$, as specified in Table 6.2,

Table 6.2: Compositional Attack Type Specification

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
$\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$	$exists(\psi_{d_P}.ip) \wedge exists(\psi_{d_P}.apache\_port) \wedge exists(\psi_{d_P}.os) \wedge exists(\psi_{d_P}.vul)$	$\langle \psi^x_{d_P}.ip, \psi_{d_P}.ip \rangle$, $\langle \psi^x_{d_P}.apache\_port, \psi_{d_P}.apache\_port \rangle$, $\langle \psi^x_{d_P}.os, \psi_{d_P}.os \rangle$, $\langle \psi^x_{d_P}.vul, \psi_{d_P}.vul \rangle$

The compositional attack type, $\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$, also has its own attack surface, which can be viewed as the overall attack surface of $[\phi_1, \phi_2, \phi_3, \phi_4]$.

- $S_{attack}(\phi) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot vul}\}$

6.5.1.3.2 Adaptation Surface $S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$. Although refreshing the Planner means we remove the complete configuration parameter, π_{d_P}, and recreate it, our analysis focuses on concrete sub configuration parameters of π_{d_P} that change during runtime.

6.5.1.3.3 Engagement Surface Base on the definition, engagement surafce between the mission planning system and each attack type are,

- $S_{engage}(\phi_1, \Sigma) = S_{attack}(\phi_1) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_2, \Sigma) = S_{attack}(\phi_2) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_3, \Sigma) = S_{attack}(\phi_3) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_4, \Sigma) = S_{attack}(\phi_4) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$

The engagement surface between Σ and compositional attack type, $\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$, is,

- $S_{engage}(\phi, \Sigma) = S_{attack}(\phi) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$

6.5.1.3.4 Coverage Based on the definition, coverage between the mission planning system and each attack type are,

- $Coverage(\phi_1, \Sigma) = \frac{|S_{engage}(\phi_1, \Sigma)|}{|S_{attack}(\phi_1)} = 1$
- $Coverage(\phi_2, \Sigma) = \frac{|S_{engage}(\phi_2, \Sigma)|}{|S_{attack}(\phi_2)} = 1/2$
- $Coverage(\phi_3, \Sigma) = \frac{|S_{engage}(\phi_3, \Sigma)|}{|S_{attack}(\phi_3)} = 1/3$
- $Coverage(\phi_4, \Sigma) = \frac{|S_{engage}(\phi_4, \Sigma)|}{|S_{attack}(\phi_4)} = 1/4$

The coverage between Σ and compositional attack type, $\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$, is,

- $Coverage(\phi, \Sigma) = \frac{|S_{engage}(\phi, \Sigma)|}{|S_{attack}(\phi)} = 1/4$

6.5.1.3.5 Potential Effectiveness Clearly, as all the values of coverage are greater than 0, the mission planning MTD system is *potentially* effective to the given attack types ϕ_1 - ϕ_4. Notice, we emphasize it is only potentially effective. Whether this MTD system is really effective and how, depends on extra factors that we will proceed to the $P_{success}$ analysis in the next section.

On the other hand, if this value is zero against an attack type, ϕ, then according to Definition 6.58, this MTD system has no impact to ϕ at all. In this situation, it implies that MTD system designer should consider increasing the adaptation surface of Σ to improve its effectiveness against ϕ.

6.5.1.3.6 Success Likelihood of Intrusion We first analyze the attack instances that implement attack type, ϕ_1, and then extend the analysis to ϕ_2, ϕ_3 and ϕ_4.

Let an attack instance, $\oint(x, d_P, \phi_1)$, which implements ϕ_1, have a mean time of compromise, T_{a_1}, and success likelihood of intrusion P_{static_1}. According to Definition 6.60

$$P_{success}(\oint(x, d_P, \phi_1)) = P(holds(\phi_1.\Omega_{pre}, [t_s, t_f])) \times P_{static_1}$$

where :

$$P(holds(\phi_1.\Omega_{pre}, [t_s, t_f])) = P(unchanged(\delta(\phi_1.\Omega_{pre}), [t_s, t_f]) \times P(holds(\Omega_{pre}, t_s))$$

For this concrete example,

$$\begin{aligned} P_{success}(\oint(x, d_P, \phi_1)) &= P(unchanged(\psi_{d_P.ip}, [t_s, t_f])) \times P(holds(exists(\psi_{d_P.ip}), t_s)) \times P_{static_1} \\ &= (1 - p_r)^{\frac{T_{a_1}}{T_r}} \times P_{static_1} \end{aligned}$$

Note, here $P(holds(exists(\psi_{d_P.ip}), t_s)) = 1$, because we assume that there is an IP address

exists in the Planner at the beginning of this attack. In addition, $P(unchanged(\psi_{d_P.ip}, [t_s, t_f])$ equals $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ because during attack interval T_a, adaptation could occur $\frac{T_a}{T_r}$ times. While for each adaptation, the probability of the Planner's IP address does not change is $(1-p_r)$.

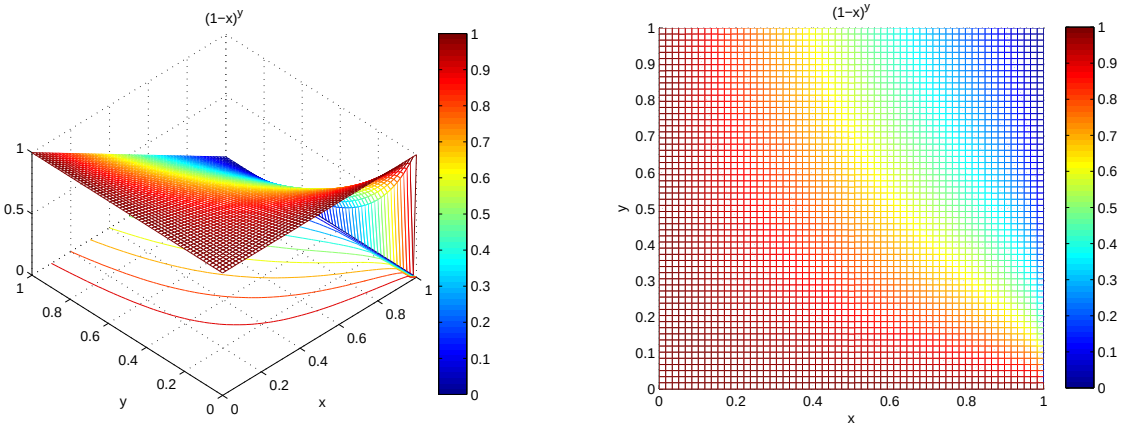
Table 6.3 shows the value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different values of p_r and $\frac{T_{a_1}}{T_r}$. As we can see, given a fixed p_r, higher $\frac{T_{a_1}}{T_r}$ leads to less success likelihood. Given a fixed $\frac{T_{a_1}}{T_r}$, higher p_r leads to the less success likelihood. When $p_r = 0$, $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ becomes 1 and this means when $p_r = 0$, a MTD system is degraded to a static system. When $p_r = 1$ and $\frac{T_{a_1}}{T_r} > 0$, $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ becomes 0. This special condition captures the fact that as long as MTD system ensures that the configuration parameter being adapted, the attack will eventually fail when adaptation happens.

Table 6.3: Value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different p_r and $\frac{T_{a_1}}{T_r}$

$p_r \setminus \frac{T_{a_1}}{T_r}$	0	0.1	0.3	0.7	1.0	3.0	6.0	9.0
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.1	1.0	0.9895	0.9689	0.9289	0.9	0.729	0.5314	0.3874
0.3	1.0	0.9650	0.8985	0.7791	0.70	0.343	0.1176	0.0404
0.6	1.0	0.9124	0.7597	0.5266	0.40	0.064	0.0041	0.0003
0.9	1.0	0.7943	0.5012	0.1995	0.10	0.001	1e-6	1e-9
1.0	1.0	0.0	0.0	0.0	0	0	0	0

To have a more complete understanding about $(1-p_r)^{\frac{T_{a_1}}{T_r}}$, we plot its value based on different p_r and $\frac{T_{a_1}}{T_r}$ values. Let $x = p_r$, $y = \frac{T_{a_1}}{T_r}$, Figure 6.9 plots the value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ when $0 \leq p_r \leq 1$, $0 \leq \frac{T_{a_1}}{T_r} \leq 1$. As we can see from Figure 6.9b, there is a large area that has $(1-p_r)^{\frac{T_{a_1}}{T_r}} \geq 0.5$, which means adapting the IP address does not provide much help under these situations. More specifically, when $p_r \to 0$, then for $0 \leq \frac{T_{a_1}}{T_r} \leq 1$, $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 1$, which means when $p_r \to 0$ and $0 \leq \frac{T_{a_1}}{T_r} \leq 1$, this mission planning MTD system is like a static system because $P_{success}(\oint(x, d_P, \phi_1))$ almost equals P_{static_1}. On the other hand, when $p_r \to 1$, $\frac{T_{a_1}}{T_r}$ needs to be small, for example $\frac{T_{a_1}}{T_r} \leq 0.1$, to make $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 1$, otherwise, $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 0$. This tells us that when the mission planning system has a high p_r, in order

to maintain $P_{success}(\oint(x, d_P, \phi_1))$ as high as P_{static_1} , the attacker needs to shorten its mean time of compromise, T_{a_1}, to maintain the $\frac{T_{a_1}}{T_r}$ as low as 0.1. Conversely, it clearly indicates that the mission planning MTD system should set a high adaptation probability, p_r, and maintain the ratio of $\frac{T_{a_1}}{T_r}$ larger than 0.1, in order to effectively reduce $P_{success}(\oint(x, d_P, \phi_1))$. When it comes the situation where T_r has to be configured relatively large, then the MTD designer should consider approaches that could increase the attack time, T_{a_1}.



(a) 3d view. The color represents different values of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different $(1-p_r)$ and $\frac{T_{a_1}}{T_r}$

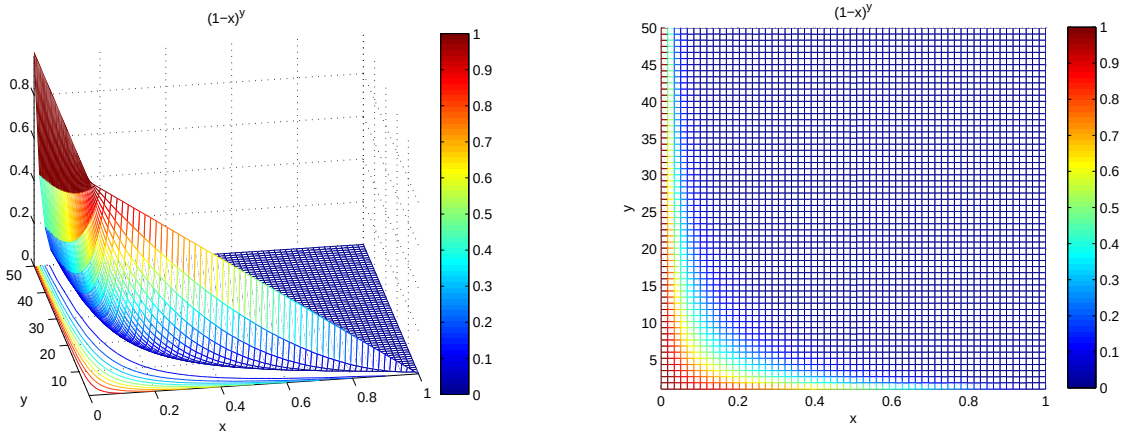
(b) 2d top down view of (a). The color represents diffrent values of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different p_r and $\frac{T_{a_1}}{T_r}$ values. Red: $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 1$. Blue: $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 0$.

Figure 6.9: Value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$, $0 \leq p_r \leq 1, 0 \leq \frac{T_{a_1}}{T_r} \leq 1$.

An interesting point to note here is that $\lim_{p_r \to 1, \frac{T_{a_1}}{T_r} \to 0} (1-p_r)^{\frac{T_{a_1}}{T_r}} = 1$, which means even when the system has a very high adaptation probability, $p_r \to 1$, but as long as the attack instance is fast enough to make $\frac{T_{a_1}}{T_r} \to 0$, the MTD system for this particular attack instance looks like a static system. This actually shows that intrusion speed divided by adaptation speed, $\frac{T_{a_1}}{T_r}$, dominate the adaptation probability, p_r, which perfectly matches a proverb – "There is no martial art is indefectible, while the speed defines the winner".

From a defensive perspective, the MTD designer should try to keep $\frac{T_{a_1}}{T_r}$ as large as possible. Figure 6.10 shows the value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ when $0 \leq p_r \leq 1$, $1 \leq \frac{T_{a_1}}{T_r} \leq 50$. As we

see, MTD becomes more effective under this situation. For example, in Figure 6.10b, when $0 \leq p_r \leq 1, 1 \leq \frac{T_{a_1}}{T_r} \leq 50$, there is a large portion that $(1-p_r)^{\frac{T_{a_1}}{T_r}} \leq 0.2$, which means the MTD system under this situation, could reduce the success likelihood of intrusion around 80% when compared to the static system. However, we should also notice that when p_r is small, such as $p_r \leq 0.05$, the MTD system doesn't help much even when $\frac{T_{a_1}}{T_r}$ is relatively high, for example $\frac{T_{a_1}}{T_r} = 50$. Actually, Figure 6.10b suggests that p_r should be at least 0.5 to effectively invalidate attack instances when $T_{a_1} \geq T_r$. This forces the attacker to finish the attack no longer than T_r to achieve a relatively high $P_{success}(\oint(x, d_P, \phi_1))$.



(a) 3d view. The color represents different values of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different $(1-p_r)$ and $\frac{T_{a_1}}{T_r}$

(b) 2d top down view of (a). The color represents diffent values of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$ based on different p_r and $\frac{T_{a_1}}{T_r}$ values. Red: $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 1$. Blue: $(1-p_r)^{\frac{T_{a_1}}{T_r}} \to 0$.

Figure 6.10: Value of $(1-p_r)^{\frac{T_{a_1}}{T_r}}$, $0 \leq p_r \leq 1, 1 \leq \frac{T_{a_1}}{T_r} \leq 50$.

If we want to consider the success likelihood of gaining a specific IP address, say 192.168.10.100, in the range of configuration space: 192.168.10.100 – 192.168.10.200, then $\psi_{d_P}.ip = 192.168.10.100$ should be part of $\phi_1.\Omega_{Pre}$. Thus according to the success likelihood

definition,

$$
\begin{aligned}
P_{success}(\oint(x, d_P, \phi_1)) &= P(holds(\phi_1.\Omega_{pre}, [t_s, t_f])) \times P_{static_1} \\
&= P(unchanged(\delta(\phi_1.\Omega_{pre}), [t_s, t_f])) \times P(holds(\phi_1.\Omega_{pre}, t_s)) \times P_{static_1} \\
&= P(unchanged(\psi_{d_P.ip}, [t_s, t_f])) \\
&\quad \times P(holds(\psi_{d_P.ip} == 192.168.10.100), t_s) \times P_{static_1} \\
&= (1 - p_r)^{\frac{T_a}{T_r}} \times \frac{1}{100} \times P_{static_1}
\end{aligned}
$$

Here, $P(holds(\psi_{d_P.ip} == 192.168.10.100), t_s) = \frac{1}{100}$ because based on the randomization adopted in this mission planning system, each IP address has the probability of $\frac{1}{99}$ to be selected, and 1 time of 0 probability being selected, thus each IP address in general has a $\frac{1}{100}$ chance of being selected. It seems this analysis only helps when we consider the situation where an attacker tries to attack a specific state. However, it actually provides very useful information for the MTD designer. If each state in the configuration space has an equal probability to be selected, then a larger configuration space leads to a smaller $P(\psi_{d_P.ip} = 192.168.10.100)$. In addition, a larger configuration space could reflect an even larger exploration space that costs the attacker more time to identify the true value. Thus, when the MTD system can not afford to change fast, which means keeping T_r relatively short or p_r relatively high, then the MTD designer should consider putting in more effort to diversify the configuration space to increase T_a. Approaches to diversifying IP address space that making gaining IP address harder, which could be adopted in this MTD example, are to consider switching from IPv4 to IPv6[20], or to take advantage of the spatio-temporal address mutation technique[125]. Chapter 5 discussed that the model presented has a limitation that VMs assigned to play the same role are assumed to have the same vulnerability. With MTD theory, this limitation can be eliminated by knowing the probability of each configuration state being selected.

Until now, we only analyzed the attack instance that implement ϕ_1 and discussed what an MTD system could do to thwart ϕ_1. Next, we analyze a sequence of attack instances that implement ϕ_2, ϕ_3 and ϕ_4.

Let's start by assuming when no adaptation happens, attack instances that implement ϕ_1 to ϕ_4 have the time interval, T_a and P_{static} as specified in Table 6.4.

Table 6.4: Attack Instances Specification

Attack Instances	$\boldsymbol{T_a}$	$\boldsymbol{P_{static}}$
$\oint_{t_0}^{t_1}(x, d_P, \phi_1)$	T_{a_1}	P_{static_1}
$\oint_{t_1}^{t_2}(x, d_P, \phi_2)$	T_{a_2}	P_{static_2}
$\oint_{t_2}^{t_3}(x, d_P, \phi_3)$	T_{a_3}	P_{static_3}
$\oint_{t_3}^{t_4}(x, d_P, \phi_4)$	T_{a_4}	P_{static_4}

Then for attack instance, $\oint(x, d_P, \phi_2)$, which implements attack type ϕ_2, we can derive its success likelihood based on the condition that $\oint(x, d_P, \phi_1)$ is success. According to the definition,

$$
\begin{aligned}
P_{success}(\oint(x, d_P, \phi_2) | \oint(x, d_P, \phi_1)) &= P(holds(\phi_2.\Omega_{pre}, [t_1, t_2])) \times P_{static_2} \\
&= P(unchanged(\delta(\phi_2.\Omega_{pre}), [t_1, t_2])) \\
&\quad \times P(holds(\phi_2.\Omega_{pre}, t_1)) \times P_{static_2} \\
&= P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}\}, [t_1, t_2]) \\
&\quad \times 1 \times P_{static_2} \\
&= (1 - p_r)^{\frac{T_{a_2}}{T_r}} \times P_{static_2}
\end{aligned}
$$

Here, $P(holds(\phi_2.\Omega_{pre}, t_1))$ is set to 1 because we analyze it based on the condition that $\oint(x, d_P, \phi_1)$ is successful and also assume that $exists(\psi_{d_P \cdot ip})$ is true. Notice the caveat that $exists(\psi_{d_P \cdot ip})$ is true, otherwise the attack will fail for sure. In addition, $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}\}, [t_1, t_2])$ is set to $(1 - p_r)^{\frac{T_{a_2}}{T_r}}$, because only the IP ad-

dress is changed and the port number remains unchanged, thus the result is similar as $P(unchanged(\psi_{d_P \cdot ip}, [t_0, t_1]))$ in ϕ_1 except the time interval switches from T_{a_1} to T_{a_2}. Using the Bayesian rule, we can derive the success likelihood when two attack instances, $\oint(x, d_P, \phi_1)$ and $\oint(x, d_P, \phi_2)$ are both successful.

$$\begin{aligned} P_{success}(\oint(x, d_P, \phi_1), \oint(x, d_P, \phi_2)) &= P_{success}(\oint(x, d_P, \phi_1)) \\ &\quad \times P_{success}(\oint(x, d_P, \phi_2) | \oint(x, d_P, \phi_1)) \\ &= (1 - p_r)^{\frac{T_{a_1}}{T_r}} \times P_{static_1} \times (1 - p_r)^{\frac{T_{a_2}}{T_r}} \times P_{static_2} \\ &= (1 - p_r)^{\frac{T_{a_1} + T_{a_2}}{T_r}} \times P_{static_1} \times P_{static_2} \end{aligned}$$

This formula actually provides another way to understand the intrusion. Attack instances $\oint(x, d_P, \phi_1)$ and $\oint(x, d_P, \phi_2)$ can be viewed as a single attack instance ϕ', which has time interval $T'_a = T_{a_1} + T_{a_2}$ and $P'_{static} = P_{static_1} \times P_{static_2}$.

This process can be extended to ϕ_3 and ϕ_4 as well to get the following results.

$$\begin{aligned} P_{success}(\oint(x, d_P, \phi_1), \oint(x, d_P, \phi_2), \oint(x, d_P, \phi_3)) =& (1 - p_r)^{\frac{T_{a_1} + T_{a_2} + T_{a_3}}{T_r}} \times P_{static_1} \\ & \times P_{static_2} \times P_{static_3} \end{aligned}$$

$$\begin{aligned} P_{success}(\oint(x, d_P, \phi_1), \oint(x, d_P, \phi_2), \oint(x, d_P, \phi_3), \oint(x, d_P, \phi_4)) =& (1 - p_r)^{\frac{T_{a_1} + T_{a_2} + T_{a_3} + T_{a_4}}{T_r}} \\ & \times P_{static_1} \times P_{static_2} \\ & \times P_{static_3} \times P_{static_4} \end{aligned}$$

Thus, this sequence of four attack instances can be viewed as a single attack instance, $\oint'$, which has a time interval $T'_a = T_{a_1} + T_{a_2} + T_{a_3} + T_{a_4}$ and $P'_{static} = P_{static_1} \times P_{static_2} \times P_{static_3} \times P_{static_4}$. And this attack instance, $\oint'$ can be viewed as an instance which implements the

compositional attack type $\phi = [\phi_1, \phi_2, \phi_3, \phi_4]$.

This also tells us that the success likelihood of intrusion analysis needs to be tied to concrete attacks and what the attacker tries to gain or modify. For example, when we consider attack instance, $\oint(x, d_P, \phi_1)$, we only focus on the IP address. However, the IP address itself is usually just a tiny step towards obtaining more valuable information from the target system, thus, if the MTD system cannot change as fast as T_{a_1}, it doesn't have to. If all an MTD system cares is that ϕ_4 is not successfully implemented, then it only needs to be change as fast as $T_{a_1} + T_{a_2} + T_{a_3} + T_{a_4}$ to decrease the $P_{success}$. However, as port number, OS and web application in this example all remain unchanged, $T_{a_2} + T_{a_3} + T_{a_4}$ could be very small. Imagine the case where attacker purchases a same system and studies it offline, then once the system is broken, the attacker could simply reuse exactly the same knowledge about port, OS and web application vulnerability and apply it to the target. Although adapting IP address will force the attacker to regain the IP address in each attack, but clearly we can diversify and change other configuration parameters as well to increase or maintain the overall attack time to avoid the situation mentioned.

In the next section, we show an MTD system where more than one configuration parameters are changed, instead of only the IP address, and analyze its effectiveness. We also show that different adaptation mechanisms will have notable varying impact to the intrusion.

6.5.2 Attack ASLR-enabled Mission Planning System

In this example, an attacker needs to correctly gain some memory locations in order to take adavantage of exploitable vulnerabilities. Thus, instead of only changing the IP address, the designer decides to enable the PaX address space layout randomization (ASLR) to add more protection. ASLR is a security technique that guards against code reuse attacks, which work by overwriting memory locations to point to potentially malicious code. By randomizing memory locations, ASLR makes it difficult to correctly guess the memory locations of specific

processes. More specifically, a processs address space contains three areas: the executable area, the mapped area and the stack area. Instead of fixing each areas base address, ASLR randomizes it by adding an extra variable to the base address when the process is created. For the Intel x86 architecture, PaX ASLR randomizes 16 or 24 bits for these areas.

For instance, the mapped data area variable *delta_mmap* has 16 bits of randomness, which means the attacker only needs to iterate from 0 to 65535 to determine its value. PaX ASLR has two properties. First, PaX ASLR randomizes only the base addresses of the three memory areas but not the *layout* within each area. Second, the *layout* is fixed throughout a process and all its children's lifetime.

Next, we provide more details to the attack types given in the previous example and consider a concrete return-to-libc attack instance that implements these attack types. The implementation of the return-to-libc attack first creates a memory hole in the Oracle 9 PL/SQL Apache module by creating an overflow buffer in the *ap_getline()* function in *http_protocol.c*. To conduct the attack, the base of the mapped area *mmap_base* and the offset of the *usleep()* function *usleep_offset* in *libc* are precomputed.(*libc* is the standard C-language library that is loaded into all Unix programs.). Then the value of *delta_mmap* is found by repeatedly overflowing the stack buffer with guesses for the absolute address of the *usleep()* function. An unsuccessful guess causes the child process to crash and be replaced by a new process with the same randomization offsets. A successful guess calls the *usleep()* function and hangs the connection for 16 seconds, which helps determine the value of *delta_mmap*. Once the value of *delta_mmap* is gained, the absolute locations of all functions in *libc* can be calculated. The final step is to smash the stack to point to another *libc* function, *system()*, which executes user supplied commands through the command shell. Shell commands are sent to *system()* as an argument.

6.5.2.1 MTD System Specification

In this new example, the mission planning system remains mostly the same as previous settings. The planner has all the prevous settings that are reproduced here as a reference.

The Planner:

- Runs a C++ web application based on Apache and Ubuntu.
- Web application has port number 80.
- Adaptation contains actions to replace the old Planner machine with a new machine and notify the changes to the AssetDB, GeoDB and TargetDB.
- New Planner is configured with the same C++ web app, port and OS.
- New Planner's IP address is picked from 192.168.10.100–192.168.10.200.
- Adaptation occurs once during each time interval T_r.
- Every T_r, the Planner get refreshed in a probability of p_r.

However, the Planner has one more configuration parameter being changed, which is *delta_mmap*.

- New Planner's *delta_mmap* is randomly picked from 0 - 65535

In addition, instead of analyzing the adaptation mechanism that the Planner gets refreshed (where both IP and *delta_mmap* changes) every T_r with a probability of p_r, we analyze a more general adaptation approach where every T_r, the IP address has a probability p_{r_1} to change and *delta_mmap* has a probability of p_{r_2} to change.

- Every T_r, the Planner's IP address is adapted with probablity p_{r_1} and *delta_mmap* is adapted with probability p_{r_2}.

6.5.2.1.1 Configuration Space The configuration space in this example has been greatly enlarged and has the size of 100 x 65536. The reason is that we can set the value of IP address and *delta_mmap* independently in each adaptation.

6.5.2.1.2 Diversification The adaptation in this mission planning system takes advantage of the IP address space as well as the memory space. No artificial diversification technique is used in this example.

6.5.2.1.3 Randomization For first adaptation option: new Planner machine's IP address is randomly picked from the IP pool except the previous IP address, thus each IP address has the probabiliy of $p_{s_{ip}} = 1/99$ to be selected and the old IP address has probability of $p_{s_{ip}} = 0$ to be chosen. Similarly, each *delta_mmap* has probability $p_{s_{delta\_mmap}} = 1/65535$ to be selected and the old *delta_mmap* has probability of zero to be chosen. In addition, for every T_r, there is a probability of $p_r = 1/4$ the Planner actually gets refreshed. Thus, the current IP and *delta_mmap* has the probability of $(1 - p_r) = \frac{3}{4}$ to remain unchanged.

For second adaptation option: new Planner machine's IP address is randomly picked from the IP pool except the previous IP address, thus each IP address has the probabiliy of $p_{s_{ip}} = 1/99$ to be selected and the old IP address has probability of $p_{s_{ip}} = 0$ to be chosen. Similarly, each *delta_mmap* has probability $p_{s_{delta\_mmap}} = 1/65535$ to be selected and the old *delta_mmap* has probability of zero to be chosen. However, the difference is that there is a probability of p_{r_1} that the IP address get changed and p_{r_2} that *delta_mmap* get changed. Thus, the current IP has probability $(1 - p_{r_1})$ and current *delta_mmap* has probability of $(1 - p_{r_2})$ to remain unchanged. As an example, one may set $p_{r_1} = 0.5$ and $p_{r_2} = 0.6$.

6.5.2.1.4 MTD Problem The two adaptation mechanisms described above are the solutions to MTD problem in this concrete example. When adaptation occurs, the new Planner will have the same configuration as before except the IP address is randomly picked

from an IP pool and the web application's memory variable *delta_mmap* is randomly picked from its available memory address space.

6.5.2.1.5 Adaptation Problem The solution to the adaptation problem in this concrete example is to synthesize a sequence of configuration actions to shutdown the old Planner, start and configure the new Planner, and then notify and update the AssetDB, GeoDB and TargetDB. When PaX ASLR is enabled, each new machine's web application will have its randomly selected *delta_mmap* value.

6.5.2.1.6 Timing Problem The solution to the timing problem is simply to schedule an adaptation in every time interval, T_r, where T_r can take different value.

6.5.2.2 Attack Type Specification

6.5.2.2.1 Attack Goal An attacker, x, has the goal to first exploit the Planner to obtain root privileges and then access TargetDB to steal sensitive data.

6.5.2.2.2 Target System In this example, the mission planning system is the target system, D, with its complete information parameter $\psi_{MissionPlanning}$. The Planner, $d_{Planner}$ (or d_P for short), and the TargetDB, $d_{TargetDB}$ (or d_T for short) are the specific targets of interest, which have complete information parameter, $\psi_{Planner}$ and $\psi_{TargetDB}$. Concrete information parameters that will be used in this example include: $\psi_{d_P \cdot ip}$, $\psi_{d_P \cdot apache\_port}$, $\psi_{d_P \cdot os}$, $\psi_{d_P \cdot mmap\_base}$, $\psi_{d_P \cdot usleep\_offset}$, $\psi_{d_P \cdot delta\_mmap}$, $\psi_{d_T \cdot ip}$, $\psi_{d_T \cdot db\_port}$, $\psi_{d_T \cdot data}$

Notice we use $\psi_{d_T \cdot data}$ to refer any sensitive data the attacker might obtain from the TargetDB.

6.5.2.2.3 Attacker To carry out the intrusion successfully, the attacker, x, must have correct knowledge of the Planner, $\psi^x_{d_P}$, including the Planner's IP address, Apache web

server port number, operating system, the vulnerability of the web server, and tables of database. Formally, these are captured via attacker knowledge in the form of information parameters such as $\psi^x_{d_P \cdot ip}$. In addition, the attacker should have general knowledge in ψ^x that captures special skills, such as how to scan the IP address, port number, how to find out *mmap_base*, *delta_mmap* and how to query data from a database.

6.5.2.2.4 Attack Type In this scenario, the attacker will face an MTD system that changes both IP address as well as the mapped data area location *delta_mmap*. The attack type $\phi_1, \phi_2, \phi_3, \phi_6$ remain unchanged from the previous example, but the ϕ_4 and ϕ_5 are actually decomposed into sub attack types. Table 6.5 and Table 6.6 show the decomposition. Note, we omit part of the preconditions in the table, $\psi^x_{d_P \cdot ip} = \psi_{d_P \cdot ip} \wedge \psi^x_{d_P \cdot apache\_port} = \psi_{d_P \cdot apache\_port} \wedge \psi^x_{d_P \cdot os} = \psi_{d_P \cdot os}$, from ϕ_4 and ϕ_5 to save space.

Table 6.5: Attack Type ϕ_4 Decomposition

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
$\phi_{4.1}$	$exists(\psi_{d_P \cdot mmap\_base})$	$\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle$
$\phi_{4.2}$	$\psi^x_{d_P \cdot mmap\_base} = \psi_{d_P \cdot mmap\_base} \wedge exists(\psi_{d_P \cdot usleep\_offset})$	$\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle$
$\phi_{4.3}$	$\psi^x_{d_P \cdot mmap\_base} = \psi_{d_P \cdot mmap\_base} \wedge \psi^x_{d_P \cdot usleep\_offset} = \psi_{d_P \cdot usleep\_offset} \wedge exists(\psi_{d_P \cdot delta\_mmap})$	$\langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle$

Table 6.6: Attack Type ϕ_5 Decomposition

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
$\phi_{5.1}$	$\psi^x_{d_P \cdot mmap\_base} = \psi_{d_P \cdot mmap\_base} \wedge exists(\psi_{d_P \cdot system\_offset})$	$\langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle$
$\phi_{5.2}$	$\psi^x_{d_P \cdot mmap\_base} = \psi_{d_P \cdot mmap\_base} \wedge \psi^x_{d_P \cdot system\_offset} = \psi_{d_P \cdot system\_offset} \wedge \psi^x_{d_P \cdot delta\_mmap} = \psi_{d_P \cdot delta\_mmap}$	$\langle \psi_{d_P \cdot exa}, \psi^x \cdot_{exa} \rangle$, $\langle \psi^x_{d_P \cdot exa}, \psi^x \cdot_{exa} \rangle$

In addition, we add another attack type ϕ_7, which obtains data from database, Table 6.7 shows the decomposition of ϕ_7. Note, we omit part of the preconditions in the table, $\psi^x_{d_P \cdot ip} = \psi_{d_P \cdot ip} \wedge \psi^x_{d_P \cdot apache\_port} = \psi_{d_P \cdot apache\_port}$ for $\phi_{7.1}$, $\phi_{7.2}$ and $\phi_{7.3}$ to save space. In addition, we assume the TargetDB database can be connected using the Planner machine's root privilege,

otherwise the attacker needs to gain correct credentials before connecting TargetDB and retrieving the data.

Table 6.7: Attack Type ϕ_7 Decomposition

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
$\phi_{7.1}$	$\psi_{d_P \cdot exa} = \psi^x_{d_P \cdot exa} \wedge \psi_{d_P \cdot root} = \psi^x_{d_P \cdot root} \wedge exists(\psi_{d_T \cdot ip})$	$\langle \psi^x_{d_T \cdot ip}, \psi_{d_T \cdot ip} \rangle$
$\phi_{7.2}$	$\psi_{d_P \cdot exa} = \psi^x_{d_P \cdot exa} \wedge \psi^x_{d_P \cdot root} = \psi_{d_P \cdot root} \wedge \psi^x_{d_T \cdot ip} = \psi_{d_T \cdot ip} \wedge exists(\psi_{d_T \cdot db\_port})$	$\langle \psi^x_{d_T \cdot db\_port}, \psi_{d_T \cdot db\_port} \rangle$
$\phi_{7.3}$	$\psi_{d_P \cdot exa} = \psi^x_{d_P \cdot exa} \wedge \psi^x_{d_P \cdot root} = \psi_{d_P \cdot root} \wedge \psi^x_{d_T \cdot ip} = \psi_{d_T \cdot ip} \wedge \psi^x_{d_T \cdot db\_port} = \psi_{d_T \cdot db\_port} \wedge exists(\psi_{d_T \cdot data})$	$\langle \psi^x_{d_T \cdot data}, \psi_{d_T \cdot data} \rangle$

Table 6.8 shows the compositional attack types which are minmal. Later we will compare the analysis applied to each individual attack types and the overall compositional attack type.

Table 6.8: Compositional Attack Types Specification

Type	$\boldsymbol{\Omega_{pre}}$	$\boldsymbol{\Omega_{post}}$
$\phi''_{4.1} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle$, $\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle$, $\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle$, $\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle$
$\phi''_{4.2} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot usleep\_offset})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle$, $\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle$, $\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle$, $\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle$, $\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle$

$\phi''_{4.3} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}, \phi_{4.3}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot usleep\_offset}) \wedge exists(\psi_{d_P \cdot delta\_mmap})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle, \langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle, \langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle, \langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle, \langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle, \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle$
$\phi''_{5.1} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{5.1}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot system\_offset})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle, \langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle, \langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle, \langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle, \langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle$
$\phi''_{5.2} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}, \phi_{4.3}, \phi_{5.1}, \phi_{5.2}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot usleep\_offset}) \wedge exists(\psi_{d_P \cdot delta\_mmap}) \wedge exists(\psi_{d_P \cdot system\_offset})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle, \langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle, \langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle, \langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle, \langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle, \langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle, \langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle, \langle \psi_{d_P \cdot exa}, \psi^x \cdot exa \rangle, \langle \psi^x_{d_P \cdot exa}, \psi^x \cdot exa \rangle$

$\phi''_6 =$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle,$
$[\phi_1, \phi_2, \phi_3, \phi_{4.1},$	$exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge$	$\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle,$
$\phi_{4.2}, \phi_{4.3}, \phi_{5.1},$	$exists(\psi_{d_P \cdot usleep\_offset}) \wedge$	$\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle,$
$\phi_{5.2}, \phi_6]$	$exists(\psi_{d_P \cdot delta\_mmap}) \wedge$	$\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle,$
	$exists(\psi_{d_P \cdot system\_offset}) \wedge exists(\psi_{d_P \cdot root})$	$\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle,$
		$\langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle,$
		$\langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle,$
		$\langle \psi_{d_P \cdot exa}, \psi^x \cdot exa \rangle,$
		$\langle \psi^x_{d_P \cdot exa}, \psi^x \cdot exa \rangle,$
		$\langle \psi^x_{d_P \cdot root}, \psi_{d_P \cdot root} \rangle$
$\phi''_{7.1} =$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle,$
$[\phi_1, \phi_2, \phi_3,$	$exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge$	$\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle,$
$\phi_{4.1}, \phi_{4.2}, \phi_{4.3},$	$exists(\psi_{d_P \cdot usleep\_offset}) \wedge$	$\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle,$
$\phi_{5.1}, \phi_{5.2}, \phi_6,$	$exists(\psi_{d_P \cdot delta\_mmap}) \wedge$	$\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle,$
$\phi_{7.1}]$	$exists(\psi_{d_P \cdot system\_offset}) \wedge exists(\psi_{d_P \cdot root}) \wedge$	$\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle,$
	$exists(\psi_{d_T \cdot ip})$	$\langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle,$
		$\langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle,$
		$\langle \psi_{d_P \cdot exa}, \psi^x \cdot exa \rangle,$
		$\langle \psi^x_{d_P \cdot exa}, \psi^x \cdot exa \rangle,$
		$\langle \psi^x_{d_P \cdot root}, \psi_{d_P \cdot root} \rangle,$
		$\langle \psi^x_{d_T \cdot ip}, \psi_{d_T \cdot ip} \rangle$

$\phi''_{7.2} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}, \phi_{4.3}, \phi_{5.1}, \phi_{5.2}, \phi_6, \phi_{7.1}, \phi_{7.2}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot usleep\_offset}) \wedge exists(\psi_{d_P \cdot delta\_mmap}) \wedge exists(\psi_{d_P \cdot system\_offset}) \wedge exists(\psi_{d_P \cdot root}) \wedge exists(\psi_{d_T \cdot ip}) \wedge exists(\psi_{d_T \cdot db\_port})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle$, $\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle$, $\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle$, $\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle$, $\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle$, $\langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle$, $\langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle$, $\langle \psi_{d_P \cdot exa}, \psi^x \cdot exa \rangle$, $\langle \psi^x_{d_P \cdot exa}, \psi^x \cdot exa \rangle$, $\langle \psi^x_{d_P \cdot root}, \psi_{d_P \cdot root} \rangle$, $\langle \psi^x_{d_T \cdot ip}, \psi_{d_T \cdot ip} \rangle$, $\langle \psi^x_{d_T \cdot db\_port}, \psi_{d_T \cdot db\_port} \rangle$
$\phi''_{7.3} = [\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}, \phi_{4.3}, \phi_{5.1}, \phi_{5.2}, \phi_6, \phi_{7.1}, \phi_{7.2}, \phi_{7.3}]$	$exists(\psi_{d_P \cdot ip}) \wedge exists(\psi_{d_P \cdot apache\_port}) \wedge exists(\psi_{d_P \cdot os}) \wedge exists(\psi_{d_P \cdot mmap\_base}) \wedge exists(\psi_{d_P \cdot usleep\_offset}) \wedge exists(\psi_{d_P \cdot delta\_mmap}) \wedge exists(\psi_{d_P \cdot system\_offset}) \wedge exists(\psi_{d_P \cdot root}) \wedge exists(\psi_{d_T \cdot ip}) \wedge exists(\psi_{d_T \cdot db\_port}) \wedge exists(\psi_{d_T \cdot data})$	$\langle \psi^x_{d_P \cdot ip}, \psi_{d_P \cdot ip} \rangle$, $\langle \psi^x_{d_P \cdot apache\_port}, \psi_{d_P \cdot apache\_port} \rangle$, $\langle \psi^x_{d_P \cdot os}, \psi_{d_P \cdot os} \rangle$, $\langle \psi^x_{d_P \cdot mmap\_base}, \psi_{d_P \cdot mmap\_base} \rangle$, $\langle \psi^x_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot usleep\_offset} \rangle$, $\langle \psi^x_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot delta\_mmap} \rangle$, $\langle \psi^x_{d_P \cdot system\_offset}, \psi_{d_P \cdot system\_offset} \rangle$, $\langle \psi_{d_P \cdot exa}, \psi^x \cdot exa \rangle$, $\langle \psi^x_{d_P \cdot exa}, \psi^x \cdot exa \rangle$, $\langle \psi^x_{d_P \cdot root}, \psi_{d_P \cdot root} \rangle$, $\langle \psi^x_{d_T \cdot ip}, \psi_{d_T \cdot ip} \rangle$, $\langle \psi^x_{d_T \cdot db\_port}, \psi_{d_T \cdot db\_port} \rangle$, $\langle \psi^x_{d_T \cdot data}, \psi_{d_T \cdot data} \rangle$

6.5.2.2.5 Attack Instances The return-to-libc attack can be viewed as a concrete implementation of attack type ϕ_1 to ϕ_6. Once the attacker has the root privilege of Planner, the attacker can learn the TargetDB address and db port through network connections, and then connect to the database with a db client to query data. We will analyze attack instances that implement ϕ_1 to ϕ_7 one by one.

6.5.2.3 Interaction Analysis

6.5.2.3.1 Attack Surface Each attack type has their attack surface.

- $S_{attack}(\phi_1) = \{\psi_{d_P \cdot ip}\}$
- $S_{attack}(\phi_2) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}\}$
- $S_{attack}(\phi_3) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \}$
- $S_{attack}(\phi_{4.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}\}$
- $S_{attack}(\phi_{4.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}\}$
- $S_{attack}(\phi_{4.3}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot delta\_mmap}\}$
- $S_{attack}(\phi_{5.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}\}$
- $S_{attack}(\phi_{5.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{attack}(\phi_6) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}\}$
- $S_{attack}(\phi_{7.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}\}$
- $S_{attack}(\phi_{7.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}, \psi_{d_T \cdot db\_port}\}$
- $S_{attack}(\phi_{7.3}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}, \psi_{d_T \cdot db\_port}, \psi_{d_T \cdot data}\}$

Each compositional attack type also has their own attack surface,

- $S_{attack}(\phi''_{4.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}\}$
- $S_{attack}(\phi''_{4.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}\}$
- $S_{attack}(\phi''_{4.3}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot delta\_mmap}\}$
- $S_{attack}(\phi''_{5.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}\}$
- $S_{attack}(\phi''_{5.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{attack}(\phi''_{6}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}\}$
- $S_{attack}(\phi''_{7.1}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}\}$
- $S_{attack}(\phi''_{7.2}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}, \psi_{d_T \cdot db\_port}\}$
- $S_{attack}(\phi''_{7.3}) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot system\_offset}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}, \psi_{d_P \cdot root}, \psi_{d_T \cdot ip}, \psi_{d_T \cdot db\_port}, \psi_{d_T \cdot data}\}$

6.5.2.3.2 Adaptation Surface $S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}$. Although refreshing the Planner means we remove the complete configuration parameter, π_{d_P}, and recreate it, our analysis does not remain at this level but only focus on concrete configuration parameters that will be changed to a different state. However, it does impact the coverage analysis to attack types $\phi_{5.2}, \phi_6, \phi_{7.1}, \phi_{7.2}, \phi_{7.3}$.

6.5.2.3.3 Engagement Surface The engagement surface between the PaX ASLR enabled mission planning system and each attack type are,

- $S_{engage}(\phi_1, \Sigma) = S_{attack}(\phi_1) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_2, \Sigma) = S_{attack}(\phi_2) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_3, \Sigma) = S_{attack}(\phi_3) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_{4.1}, \Sigma) = S_{attack}(\phi_{4.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_{4.2}, \Sigma) = S_{attack}(\phi_{4.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_{4.3}, \Sigma) = S_{attack}(\phi_{4.3}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}$
- $S_{engage}(\phi_{5.1}, \Sigma) = S_{attack}(\phi_{5.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi_{5.2}, \Sigma) = S_{attack}(\phi_{5.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi_6, \Sigma) = S_{attack}(\phi_6) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi_{7.1}, \Sigma) = S_{attack}(\phi_{7.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi_{7.2}, \Sigma) = S_{attack}(\phi_{7.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi_{7.3}, \Sigma) = S_{attack}(\phi_{7.3}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot exa}\}$

The engagement surface between the PaX ASLR enabled mission planning system and each compositional attack type are:

- $S_{engage}(\phi''_{4.1}, \Sigma) = S_{attack}(\phi''_{4.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi''_{4.2}, \Sigma) = S_{attack}(\phi''_{4.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi''_{4.3}, \Sigma) = S_{attack}(\phi''_{4.3}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}$
- $S_{engage}(\phi''_{5.1}, \Sigma) = S_{attack}(\phi''_{5.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}\}$
- $S_{engage}(\phi''_{5.2}, \Sigma) = S_{attack}(\phi''_{5.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$

- $S_{engage}(\phi''_6, \Sigma) = S_{attack}(\phi''_6) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi''_{7.1}, \Sigma) = S_{attack}(\phi''_{7.1}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi''_{7.2}, \Sigma) = S_{attack}(\phi''_{7.2}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$
- $S_{engage}(\phi''_{7.3}, \Sigma) = S_{attack}(\phi''_{7.3}) \cap S_{adapt}(\Sigma) = \{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}, \psi_{d_P \cdot exa}\}$

Notice the difference between $S_{engage}(\phi_6, \Sigma)$ and $S_{engage}(\phi''_6, \Sigma)$, $S_{engage}(\phi_{7.1}, \Sigma)$ and $S_{engage}(\phi''_{7.1}, \Sigma)$, $S_{engage}(\phi_{7.2}, \Sigma)$ and $S_{engage}(\phi''_{7.2}, \Sigma)$, and $S_{engage}(\phi_{7.3}, \Sigma)$ and $S_{engage}(\phi''_{7.3}, \Sigma)$. The reason for these differences is that compositional attack types combine all information parameters contained in each sub attack type, while each sub attack type only contains the information parameters required by itself. For example, $\psi_{d_P \cdot delta\_mmap}$ belongs to $S_{engage}(\phi''_6, \Sigma)$ because ϕ''_6 include the sub attack type $\phi_{4.3}$ that gains $\psi_{d_P \cdot delta\_mmap}$. However, $\psi_{d_P \cdot delta\_mmap}$ doesn't belong to $S_{engage}(\phi_6, \Sigma)$ because ϕ_6 as an individual attack type does not require $\psi_{d_P \cdot delta\_mmap}$.

6.5.2.3.4 Coverage Based on the definition, coverage between the mission planning system and each attack type are,

- $Coverage(\phi_1, \Sigma) = \frac{|S_{engage}(\phi_1, \Sigma)|}{|S_{attack}(\phi_1)} = 1$
- $Coverage(\phi_2, \Sigma) = \frac{|S_{engage}(\phi_2, \Sigma)|}{|S_{attack}(\phi_2)} = 1/2$
- $Coverage(\phi_3, \Sigma) = \frac{|S_{engage}(\phi_3, \Sigma)|}{|S_{attack}(\phi_3)} = 1/3$
- $Coverage(\phi_{4.1}, \Sigma) = \frac{|S_{engage}(\phi_{4.1}, \Sigma)|}{|S_{attack}(\phi_{4.1})} = 1/4$
- $Coverage(\phi_{4.2}, \Sigma) = \frac{|S_{engage}(\phi_{4.2}, \Sigma)|}{|S_{attack}(\phi_{4.2})} = 1/5$
- $Coverage(\phi_{4.3}, \Sigma) = \frac{|S_{engage}(\phi_{4.3}, \Sigma)|}{|S_{attack}(\phi_{4.3})} = 2/6$
- $Coverage(\phi_{5.1}, \Sigma) = \frac{|S_{engage}(\phi_{5.1}, \Sigma)|}{|S_{attack}(\phi_{5.1})} = 1/5$

- $Coverage(\phi_{5.2}, \Sigma) = \frac{|S_{engage}(\phi_{5.2}, \Sigma)|}{|S_{attack}(\phi_{5.2})} = 3/7$
- $Coverage(\phi_{6}, \Sigma) = \frac{|S_{engage}(\phi_{6}, \Sigma)|}{|S_{attack}(\phi_{6})} = 2/4$
- $Coverage(\phi_{7.1}, \Sigma) = \frac{|S_{engage}(\phi_{7.1}, \Sigma)|}{|S_{attack}(\phi_{7.1})} = 2/5$
- $Coverage(\phi_{7.2}, \Sigma) = \frac{|S_{engage}(\phi_{7.2}, \Sigma)|}{|S_{attack}(\phi_{7.2})} = 2/6$
- $Coverage(\phi_{7.3}, \Sigma) = \frac{|S_{engage}(\phi_{7.3}, \Sigma)|}{|S_{attack}(\phi_{7.3})} = 2/7$

Note, the coverage for attack types $\phi_{5.2}, \phi_6, \phi_{7.1}, \phi_{7.2}$ and $\phi_{7.3}$ take $\psi_{d_P \cdot exa}$ (which is an execution agent uploaded by attacker) into account because it is added to the system by the attacker and it will be removed by refreshing the Planner. Clearly, if the MTD system does not adopt a refreshing approach, but only changes the IP and *delta_mmap* inside the original Planner machine, then $\psi_{d_P \cdot exa}$ should not be taken into consideration.

We can also derive the coverage between the mission planning system and each compositional attack type.

- $Coverage(\phi''_{4.1}, \Sigma) = \frac{|S_{engage}(\phi''_{4.1}, \Sigma)|}{|S_{attack}(\phi''_{4.1})} = 1/4$
- $Coverage(\phi''_{4.2}, \Sigma) = \frac{|S_{engage}(\phi''_{4.2}, \Sigma)|}{|S_{attack}(\phi''_{4.2})} = 1/5$
- $Coverage(\phi''_{4.3}, \Sigma) = \frac{|S_{engage}(\phi''_{4.3}, \Sigma)|}{|S_{attack}(\phi''_{4.3})} = 2/6$
- $Coverage(\phi''_{5.1}, \Sigma) = \frac{|S_{engage}(\phi''_{5.1}, \Sigma)|}{|S_{attack}(\phi''_{5.1})} = 1/5$
- $Coverage(\phi''_{5.2}, \Sigma) = \frac{|S_{engage}(\phi''_{5.2}, \Sigma)|}{|S_{attack}(\phi''_{5.2})} = 3/7$
- $Coverage(\phi''_{6}, \Sigma) = \frac{|S_{engage}(\phi''_{6}, \Sigma)|}{|S_{attack}(\phi''_{6})} = 3/8$
- $Coverage(\phi''_{7.1}, \Sigma) = \frac{|S_{engage}(\phi''_{7.1}, \Sigma)|}{|S_{attack}(\phi''_{7.1})} = 3/9$
- $Coverage(\phi''_{7.2}, \Sigma) = \frac{|S_{engage}(\phi''_{7.2}, \Sigma)|}{|S_{attack}(\phi''_{7.2})} = 3/10$
- $Coverage(\phi''_{7.3}, \Sigma) = \frac{|S_{engage}(\phi''_{7.3}, \Sigma)|}{|S_{attack}(\phi''_{7.3})} = 3/11$

6.5.2.3.5 Potential Effectiveness As the coverage values are all greater than zero, the ASLR-enabled mission planning system is *potentially* effective against all the specified attack types from ϕ_1 to $\phi_{7.3}$. Thus, we can proceed to analyze the effectiveness further in terms of success likelihood of intrusion.

6.5.2.3.6 Success Likelihood of Intrusion In the previous example, we analyzed the $P_{success}(\oint)$ of the attack instances that implement ϕ_1, ϕ_2, ϕ_3 and ϕ_4 under a changing IP environment. In this section, we analyze the MTD impact to $P_{success}$ when multiple configuration parameters are adapted simultaneously. We will show that the results of $P_{success}$ are different given the two adaptation options presented in the randomization section.

Let's start by assuming attack instances that implement ϕ_1 to ϕ_7 have the time interval, T_a, and P_{static} as specified in Table 6.9. To simplify the writing, we use symbols $\oint_1$ - $\oint_{7.2}$ to represent the event that each attack instance is successful.

Table 6.9: Attack Instances Specification

Symbol	**Attack Instances**	$\boldsymbol{T_a}$	$\boldsymbol{P_{static}}$
$\oint_1$	$\oint_{t_0}^{t_1}(x, d_P, \phi_1)$	T_{a_1}	P_{static_1}
$\oint_2$	$\oint_{t_1}^{t_2}(x, d_P, \phi_2)$	T_{a_2}	P_{static_2}
$\oint_3$	$\oint_{t_2}^{t_3}(x, d_P, \phi_3)$	T_{a_3}	P_{static_3}
$\oint_{4.1}$	$\oint_{t_3}^{t_{4.1}}(x, d_P, \phi_{4.1})$	$T_{a_{4.1}}$	$P_{static_{4.1}}$
$\oint_{4.2}$	$\oint_{t_{4.1}}^{t_{4.2}}(x, d_P, \phi_{4.2})$	$T_{a_{4.2}}$	$P_{static_{4.2}}$
$\oint_{4.3}$	$\oint_{t_{4.2}}^{t_{4.3}}(x, d_P, \phi_{4.3})$	$T_{a_{4.3}}$	$P_{static_{4.3}}$
$\oint_{5.1}$	$\oint_{t_{4.3}}^{t_{5.1}}(x, d_P, \phi_{5.1})$	$T_{a_{5.1}}$	$P_{static_{5.1}}$
$\oint_{5.2}$	$\oint_{t_{5.1}}^{t_{5.2}}(x, d_P, \phi_{5.2})$	$T_{a_{5.2}}$	$P_{static_{5.2}}$
$\oint_6$	$\oint_{t_{5.2}}^{t_6}(x, d_P, \phi_6)$	T_{a_6}	P_{static_6}
$\oint_{7.1}$	$\oint_{t_6}^{t_{7.1}}(x, d_T, \phi_{7.1})$	$T_{a_{7.1}}$	$P_{static_{7.1}}$
$\oint_{7.2}$	$\oint_{t_{7.1}}^{t_{7.2}}(x, d_T, \phi_{7.2})$	$T_{a_{7.2}}$	$P_{static_{7.2}}$
$\oint_{7.3}$	$\oint_{t_{7.2}}^{t_{7.3}}(x, d_T, \phi_{7.3})$	$T_{a_{7.3}}$	$P_{static_{7.3}}$

Based on this, we first analyze the success likelihood of intrusion based on the first adaptation option, then extend it to a more general case, the second adaptation option.

First adapation option: For every T_r, there is a probability of p_r that the Planner

actually gets refreshed. Thus, the current IP and *delta_mmap* has the probability of $(1-p_r)$ of remaining unchanged. If the Planner gets refreshed, the IP address and *delta_mmap* will be changed to a different value. This option is the same as the approach used in the previous example, except that it changes both the IP address and *delta_mmap* simultaneously. The $P_{success}$ for attack instances that implements ϕ_1-$\phi_{4.2}$ can be derived similarly as the previous example.

$$P_{success}(\oint_1) = (1-p_r)^{\frac{T_{a_1}}{T_r}} \times P_{static_1}$$

$$P_{success}(\oint_1, \oint_2) = (1-p_r)^{\frac{T_{a_1}+T_{a_2}}{T_r}} \times P_{static_1} \times P_{static_2}$$

$$P_{success}(\oint_1, \oint_2, \oint_3) = (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3}$$

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}) = (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}+T_{a_{4.1}}}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3} \times P_{static_{4.1}}$$

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) = (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}+T_{a_{4.1}}+T_{a_{4.2}}}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}}$$

Because the attack surface of $\phi_1 - \phi_{4.2}$ doesn't include *delta_mmap*, the success likelihood derivation is the same as the previous mission planning system example.

However, from attack type $\phi_{4.3}$, the change of *delta_mmap* should be taken into consideration. We can rewrite $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3})$ according to the bayesian rule,

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) = P_{success}(\oint_{4.3} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2})$$

As $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2})$ is already known, thus, we only need to derive $P_{success}(\oint_{4.3} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2})$.

Again, according to Definition 6.60

$$P_{success}(\oint) = P(holds(\Omega_{pre}, [t_s, t_f])) \times P_{static}$$

where :

$$P(holds(\Omega_{pre}, [t_s, t_f])) = P(unchanged(\delta(\phi.\Omega_{pre}, [t_s, t_f])) \times P(holds(\Omega_{pre}, t_s))$$

Then,

$$\begin{aligned}
P_{success}(\oint_{4.3} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) =& P(unchanged(\delta(\phi_{4.3}.\Omega_{pre}), [t_{4.2}, t_{4.3}])) \\
& \times P(holds(\phi_{4.3}.\Omega_{pre}, t_{4.2})) \times P_{static_{4.3}} \\
=& P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap_base}, \\
& \psi_{d_P \cdot usleep_offset}, \psi_{d_P \cdot delta_mmap}\}, [t_{4.2}, t_{4.3}])) \times 1 \times P_{static_{4.3}} \\
=& P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta_mmap}\}, [t_{4.2}, t_{4.3}])) \times P_{static_{4.3}} \\
=& (1 - p_r)^{\frac{Ta_{4.3}}{T_r}} \times P_{static_{4.3}}
\end{aligned}$$

Notice, $P(holds(\phi_{4.3}.\Omega_{pre}, t_{4.2}))$ is set to one because we consider the success likelihood of $\oint_{4.3}$ given that $\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}$ are successful. In addition, $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}]))$ has been simplified to $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}), [t_{4.2}, t_{4.3}])$ because $apache\_port, os, mmap\_base$, and *usleep_offset* all remains static and are independent of *ip* and *delta_mmap*. However, $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}]))$ equals $(1 - p_r)^{\frac{Ta_{4.3}}{T_r}}$, which is the same as when we only change the IP address. Notice that this fomula still applies when *delta_mmap* is also adapted. The reason is quite straight forward due to the adaption mechanism adopted. The change of IP and *delta_mmap* is controlled by a single switch, p_r. Thus,

these two factors either change simultaneously or both remain unchanged. In the second adaptation option, we will see that the changes of IP and *delta_mmap* each have their own switch, then $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}])) =$ $P(unchanged(\psi_{d_P \cdot ip}, [t_{4.2}, t_{4.3}])) \times P(unchanged(\psi_{d_P \cdot delta\_mmap}, [t_{4.2}, t_{4.3}]))$, which will lead to a different result.

However, this does not mean that using the first adaptation option that changing multiple factors adds no benefit. The reason is the same as disscussed in the previous example. Imagine an attacker purchased the same system and studied it offline. If *delta_mmap* remain unchanged, then this knowledge can be used directly to the target system, which will greatly reduce the attack time $T_{a_{4.3}}$ and lead to a higher success likelihood. Thus, if PaX ASLR is enabled, the attack time used to gain *delta_mmap* will also be forced to remain relatively stable instead of decreasing. From this perspective, the adaptation pushes the time cost on each intrusion to be relatively stable and makes each attack look like a first time intrusion. Thus, in the long term, the more configuratoin parameters an MTD system adapts, a relatively longer attack time can be forced on the attacker, which leads to a relatively less success likelihood of intrusion.

Based on this derivation, we have

$$\begin{aligned} P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) =& P_{success}(\oint_{4.3} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) \\ =& (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}+T_{a_{4.1}}+T_{a_{4.2}}+T_{a_{4.3}}}{T_r}} \times P_{static_1} \times P_{static_2} \\ & \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}} \end{aligned}$$

By changing IP address and *delta_mmap*, an MTD could maintain the mean time of compromise T_{a_1} and $T_{a_4.3}$ relatively stable and avoid the situation discussed above. Clearly, if the MTD system could also add difficulty to attackers each time they want to gain *port*, *os*, *mmap_base* and *usleep_offset*, then the overall intrusion time $T_{a_1} + T_{a_2} + T_{a_3} + T_{a_{4.1}} +$

$T_{a_{4.2}} + T_{a_{4.3}} T_r$ could be remarkably increased, which leads to less success likelihood.

Next, we have:

$$\begin{aligned} P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{5.1}) &= P_{success}(\oint_{5.1} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}) \\ &= (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}+T_{a_{4.1}}+T_{a_{5.1}}}{T_r}} \times P_{static_1} \times P_{static_2} \\ &\quad \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{5.1}} \end{aligned}$$

Notice, the composite attack type $[\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{5.1}]$ is *minimal* in terms that $\phi_{5.1}$ try to gain *system_offset*, which doesn't depends on $\phi_{4.2}$ that gains *usleep_offset* and $\phi_{4.3}$ that gains *delta_mmap*. Again, as indicated, only *minimal composite attack type* is considered.

Next, we consider the composite attack type $[\phi_1, \phi_2, \phi_3, \phi_{4.1}, \phi_{4.2}, \phi_{4.3}, \phi_{5.1}, \phi_{5.2}]$. This time, all previous attack types needs to be considered because uploading the execution agent needs all the information parameters gained in previous attack types.

Before derive $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2})$, we need to first derive $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1})$.

$$\begin{aligned} P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}) &= P_{success}(\oint_{5.1} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) \\ &\quad \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) \\ &= P_{success}(\oint_{5.1} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) \\ &= (1-p_r)^{\frac{T_{a_1}+T_{a_2}+T_{a_3}+T_{a_{4.1}}+T_{a_{4.2}}+T_{a_{4.3}}+T_{a_{5.1}}}{T_r}} \times P_{static_1} \times P_{static_2} \\ &\quad \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}} \times P_{static_{5.1}} \end{aligned}$$

Then,

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}) = P_{success}(\oint_{5.2} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1})$$
$$= (1-p_r)^{\frac{Ta_1+Ta_2+Ta_3+Ta_{4.1}+Ta_{4.2}+Ta_{4.3}+Ta_{5.1}+Ta_{5.2}}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}} \times P_{static_{5.1}} \times P_{static_{5.2}}$$

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6) =$$
$$P_{success}(\oint_6 \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2})$$
$$= (1-p_r)^{\frac{Ta_1+Ta_2+Ta_3+Ta_{4.1}+Ta_{4.2}+Ta_{4.3}+Ta_{5.1}+Ta_{5.2}+Ta_6}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}} \times P_{static_{5.1}} \times P_{static_{5.2}} \times P_{static_6}$$

Notice, the attack instance that implements ϕ_6 is an interesting case. By the precondition of ϕ_6, we see it doesn't depend on the *apache_port*, *os*, *mmap_base*, etc. However, because of the refreshing, once the Planner is adapted, the uploaded execution agent will be deleted and the attacker will lose the root privilege. Thus, the attacker will be forced to perform all the intrusion steps again. However, if in the first adaptation option, we switch to another mechanism that only changes the IP address and *delta_mmap* within the original Planner machine, then once $\oint_{5.2}$ is true, the uploaded execution agent can remain inside the Planner after adaptation. Then, obtaining the root privilege is just a matter of whether it can successfully onnect to the uploaded agent or not. If the uploaded agent can provide a reverse shell automatically, then $P_{success}(\oint_6) = (1-P_r)^{\frac{Ta_6}{T_r}} \times P_{static_6}$. This means there is no need to perform the whole sequence of attack any more, and at this stage, the attacker has

successfully reduced the overall intrusion time from $T_{a_1} + T_{a_2} + T_{a_3} + T_{a_{4.1}} + T_{a_{4.2}} + T_{a_{4.3}} + T_{a_{5.1}} + T_{a_{5.2}} + T_{a_6}$ to T_{a_6}, which greatly increases the chance of success. This also shows the benefit of including refreshing to the MTD system.

In addition,

$$
\begin{aligned}
P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6, \oint_{7.1}) = \\
P_{success}(\oint_7 \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6) \\
\times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6) \\
= (1 - p_r)^{\frac{T_{a_1} + T_{a_2} + T_{a_3} + T_{a_{4.1}} + T_{a_{4.2}} + T_{a_{4.3}} + T_{a_{5.1}} + T_{a_{5.2}} + T_{a_6} + T_{a_{7.1}}}{T_r}} \times P_{static_1} \times P_{static_2} \times P_{static_3} \\
\times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}} \times P_{static_{5.1}} \times P_{static_{5.2}} \times P_{static_6} \times P_{static_{7.1}}
\end{aligned}
$$

$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6, \oint_{7.1}, \oint_{7.2})$ and $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}, \oint_{5.1}, \oint_{5.2}, \oint_6, \oint_{7.1}, \oint_{7.2}, \oint_{7.3})$ can be derived similarly and we omit here.

Second adaptation option: In this option, for each T_r, the Planner gets replaced. However, the difference is that there is a probability, p_{r_1}, that IP address gets changed and, p_{r_2}, that *delta_mmap* get changed. Thus, the IP address and *delta_mmap* each have their own switch to decide change or not.

In this case, the probability from $P_{success}(\oint_1)$ to $P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2})$ remains the

same as the first adaptation option. The difference happens at $\oint_{4.3}$,

$$P_{success}(\oint_{4.3} \mid \oint_{1}, \oint_{2}, \oint_{3}, \oint_{4.1}, \oint_{4.2}) = P(unchanged(\delta(\phi_{4.3}.\Omega_{pre}), [t_{4.2}, t_{4.3}])) \times P(holds(\phi_{4.3}.\Omega_{pre}, t_{4.2})) \times P_{static_{4.3}}$$

$$= P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot apache\_port}, \psi_{d_P \cdot os}, \psi_{d_P \cdot mmap\_base}, \psi_{d_P \cdot usleep\_offset}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}])) \times 1 \times P_{static_{4.3}}$$

$$= P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}])) \times P_{static_{4.3}}$$

$$= P(unchanged(\psi_{d_P \cdot ip}, [t_{4.2}, t_{4.3}])) \times P(unchanged(\psi_{d_P \cdot delta\_mmap}, [t_{4.2}, t_{4.3}])) \times P_{static_{4.3}}$$

$$= (1 - p_{r_1})^{\frac{T_{a_{4.3}}}{T_r}} \times (1 - p_{r_2})^{\frac{T_{a_{4.3}}}{T_r}} \times P_{static_{4.3}}$$

As we see, this adaptation option makes the change of IP address and *delta_mmap* independent, thus we can decompose $P(unchanged(\{\psi_{d_P \cdot ip}, \psi_{d_P \cdot delta\_mmap}\}, [t_{4.2}, t_{4.3}]))$ into $P(unchanged(\psi_{d_P \cdot ip}, [t_{4.2}, t_{4.3}])) \times P(unchanged(\psi_{d_P \cdot delta\_mmap}, [t_{4.2}, t_{4.3}]))$. In addition, if we set $p_{r_1} = p_{r_2} = p_r$, then $P_{success}(\oint_{4.3} \mid \oint_{1}, \oint_{2}, \oint_{3}, \oint_{4.1}, \oint_{4.2}) = (1 - p_r)^{\frac{2 \times T_{a_{4.3}}}{T_r}} \times P_{static_{4.3}}$ instead of $P_{success}(\oint_{4.3} \mid \oint_{1}, \oint_{2}, \oint_{3}, \oint_{4.1}, \oint_{4.2}) = (1-p_r)^{\frac{T_{a_{4.3}}}{T_r}} \times P_{static_{4.3}}$. Thus, changing two factors is like doubling the attack time $T_{a_{4.3}}$, which in turn decreased the $P_{success}$. On the other hand, if set $p_{r_2} = 0$, which makes the *delta_mmap* remain unchanged, then $(1 - p_{r_2})^{\frac{T_{a_{4.3}}}{T_r}}$ immediately becomes one and the overall success likelihood is increased. From this aspect, changing multiple factors will indeed help reduce the success likelihood.

Accordingly,

$$P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}, \oint_{4.3}) =$$
$$P_{success}(\oint_{4.3} \mid \oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2}) \times P_{success}(\oint_1, \oint_2, \oint_3, \oint_{4.1}, \oint_{4.2})$$
$$= (1-p_{r_1})^{\frac{Ta_1+Ta_2+Ta_3+Ta_{4.1}+Ta_{4.2}+Ta_{4.3}}{T_r}} \times (1-p_{r_2})^{\frac{Ta_{4.3}}{T_r}} \times P_{static_1} \times P_{static_2}$$
$$\times P_{static_3} \times P_{static_{4.1}} \times P_{static_{4.2}} \times P_{static_{4.3}}$$

The other success likelihood can also be updated correspondently and omitted here.

6.5.3 Discussion

In this validation section, we make a start towards validation of our theory by introducing two concrete example scenarios and show how various definitions in the theory can be instantiated. As we see, the theory can be successfully applied to these example scenarios. These examples provide concrete understanding about how it can be used to analyze the potential effectiveness of an MTD system as well as $P_{success}$. Moreover, they give the MTD designer a more complete understanding of how various parameter settings will impact its effectiveness compared to a static system, as well as what can an MTD system do to thwart specific attack types.

Note that there are two parts that constitute $P_{success}$, $P(holds(\Omega_{pre}, [t_s, t_f]))$ and P_{static}. However, we only have shown how to quantify $P(holds(\Omega_{pre}, [t_s, t_f]))$. Because the focus here is to analyze and quantify the security benefit gained as compared to a static system, which $P(holds(\Omega_{pre}, [t_s, t_f]))$ illustrates nicely. More discussion about this can be found in Section 7.3.

6.6 Conclusion

This chapter presents a theory of moving target defense. It starts by first introducing several real world examples and extracts the work flow of MTD in general.

Secondly, it defines the basic concepts and problems associated with an MTD system. The MTD system theory starts by defining a configurable system based on configuration management theory and labeled transition systems. Then, it defines system goals and policies to help capture and determine what are valid and complete configurations. Adaptation and MTD system definitions are given next. Finally, the configuration space is defined along with a new concept called the exploration space. Two key tactics as discussed in Chapter 3, diversification and randomization, are also formally defined.

Next, Cyber Attack Theory is presented, which include definitions to precisely capture targets, target systems, attacker's knowledge, attack types, attack instances and exploration space. Building on top of MTD System Theory and Cyber Attack Theory, MTD Theory captures the dynamic and changing nature of MTD by defining several new concepts – attack surface, adaptation surface, engagement surface. These three theories together provide a better understanding of the interactions between attackers and MTD systems, which enable the development of a powerful and fine-grained model to analyze the success likelihood of a multi-step intrusion.

To validate the theory, examples showing how this theory maps to a memory-based remote exploit attack demonstrate its use in supporting intrusion analysis under typical movement mechanisms. How this theory can be used to guide the design of an MTD system is also shown.

Chapter 7

Conclusion

This chapter concludes this dissertation by first summarizing the current state of MTD research in Section 7.1 and the contributions of this work in Section 7.2, then describing the limitations of this work in Section 7.3 and highlighting some areas of future work in Section 7.4.

7.1 Current State

Powered by its capability to eliminate an attacker's asymmetric time advantage and increase the attacker's uncertainty, MTD is envisioned as a potential game changer in cyber warfare. However, in its current state, there is little science behind it. In fact, the approach is so new that there is no standard definition of what an MTD is, what is meant by attack surface or metrics to define the effectiveness of such systems.

Although there are many ongoing research efforts, moving target defense is still in its infancy. Most of the previous work focuses on some specific aspect of system configuration, such as IP address, memory layout, instruction set, HTML keyword, etc. While as discussed in Chapter 3 there are several comprehensive frameworks that have been proposed, most of

these frameworks are still at the conceptual level and significant effort is required to bring them to fruition, from both theoretical and practical standpoints.

As a recall from Chapter 3, there are two major challenges in MTD research based on the literature review. The first major challenge is the need for new metrics. Existing metrics for attack surface areas are not suitable for evaluating a moving attack surface because two basic assumptions of the existing metrics have been broken. One assumption is that the attack surface remains unchanged, while the other is that the target attack surface is always reachable by attackers. Thus, new metrics are required to take into account MTD's changing and unpredictable nature. Although the changing of attack surface can be done by introducing diversity, quantitative models for guiding the design of good diversification techniques and assessing their effectiveness remain largely unexplored. Actually, a fundamental challenge in understanding the impact of diversification is to introduce a precise, computationally-meaningful way to measure the increase in difficulty for the attacker. As will be discussed in Section 7.2, the main contribution of this work is to tackle this challenge.

Operation and security are both important concerns today. Unfortunately, these two aspects could be conflicting in MTD. Increased security usually brings increased operational costs. The work in this thesis will not only help measure and analyze the different mechanism's impact on security, but will also help MTD designers to make better decisions about system parameter settings such that reasonable trade offs between security and operation can be achieved.

The second major challenge is also due to the tension between operation and security in MTD. It is engineering based and lies in how to sanitize potentially compromised machines while not interrupting normal operations significantly. MT6D[20] does a good job at the network-level although IP collisions are still possible. But at the machine-level, ensuring a potentially compromised VM is cleaned is still a major challenge. Although ideas like checkpointing have been proposed, no concrete system that evaluates the performance or

validates such an approach's applicability is presented. Although this challenge is out of the scope, this work could still be beneficial in terms that it can help MTD designers make decisions about adopting different techniques to balance operation and security.

7.2 Contributions

This thesis focuses on tackling the first major challenge for understanding and quantifying moving target defense.

First, a taxonomy based on the review of related MTD work is presented. This taxonomy shows that many different MTD mechanisms have been tried to defeat different attack types. These mechanisms include different combinations of strategies and tactics that applied to various adaptable aspects. It not only helps one understand the state-of-the-art research in MTD, but also underscores the need for a comprehensive theoretical framework for MTD. Because the taxonomy clearly indicates that the effectiveness of MTD only makes sense in the context of a specific attack type, and to anlyze and quantify the effectiveness of MTD, we must be able to reason the interactions between attacker and MTD system, which further requires that an MTD System Theory that clarifies what constitutes an MTD system as well as a Cyber Attack Theory that captures the characteristics of cyber attacks to be defined.

Second, a high-level MTD system design is presented. Based on this design, an MTD simulator is developed with core algorithms given. The simulator is used to investigate the degree to which proactive and random adaptations can decrease an adversary's chance of success. A set of experiments are conducted to examine both a purely random MTD system, as well as an intelligent MTD system, which uses attack indicators to augment adaptation selection. The results show that the attackers success likelihood can be reduced under such MTD system. Although simulations help us understand the effect of MTDs with intuitive results, this brings up the question of whether an underlying model exists that can

be used to predict the intrusion success likelihood. Models like this will be invaluable for MTD systems because they can explicitly reveal the key relationships between the important parameters involved and are the key to help understanding why and how adaptation can improve security. Driven by this question, a scalable analytical model suitable for non-cyclic network topology structure is proposed. The model is scalable because it can be used to analyze deep multi-hop remote attacks from high-level. The original simulator is extended to validate the accuracy of this model. The results show that the success likelihood of intrusion estimated by this model match the simulation results.

The taxonomy, simulated experiments and analytical model provide important insight for MTD designers. However, a systematic way to understand and quantify the effectiveness of MTD is still missing. To address this, the third contribution is a theoretic framework.

This framework includes an MTD System Theory, a Cyber Attack Theory and the MTD Theory. MTD System Theory defines the basic concepts and problems associated with an MTD system and is founded on configuration management theory in order to define a configurable system, upon which the definition of an MTD system is built. Goals and policies are also captured as essential elements to determine what should be considered complete and valid configurations of an MTD system. Based on that, configuration space is formally defined. MTD system theory also formally defines the concepts of diversification and randomization and showed how they relate to MTD systems and how their use can increase the effectiveness of those systems. In addition, key problems of MTD system is also formally defined. Specifically, the essential problem of an MTD system is how to select the next valid configuration state of the system. This problem drives the solution of the other problems, such as the Adaptation Selection Problem and the Timing Problem. Cyber Attack Theory introduced a new concept called information parameter, which has been used to support the information-based definitions of target systems and attacker's knowledge. In addition, information parameters have also been used to define the concepts of attack type,

attack goal, attack instance and exploration space. The Cyber Attack Theory encourages MTD designers and researchers to formally specify and compose attack types to discover the exact system information parameters of interest and how they interact with specific attacks, the attacker's knowledge, and the target system. Specification and analysis of attacks will enable MTD designers to potentially ignore unrelated information parameters while focusing on those most critical, dramatically limiting the scope and cost of MTD systems without sacrificing effectiveness. Finally, MTD Theory is introduced based on MTD System Theory and Cyber Attack Theory. It defines how elements of the MTD system and cyber attacks interact. Specifically, concepts, such as attack surface, adaptation surface, and engagement surface, are defined to capture the interaction between an MTD system and a specific attack type. Metrics, such as coverage, success likelihood of intrusion, are defined to analyze the effectiveness of an MTD system against an attack type. Built on top of the theory definitions, several useful theorems are derived, which can be used as general guidelines for MTD system implementation. To validate the theory, two concrete scenarios are provided and show how the theory can be used to guide the MTD design as well as analyze what an MTD system could do and to thwart a concrete *return-to-libc* attack.

7.3 Limitations

There are two limitations of this work. First, it lacks experimental validation from real world. All the experiments conducted are based on a simulated environment. Although simulation captures the most important factors or parameters, it also abstracts away some details. For example, the adaptation happens every T_r time interval in the simulation, which means the adaptation start and finish instantaneously every T_r. However, in reality, adaptation itself takes time and could break the normal functionality of the system. Although simulation does not precisely capture the reality, I should emphasize that the adaptation case that the

simulation actually gives the advantage to the attacker, because a zero time of adaptation essentially allows the attacker to make full usage of every adaptation interval T_r.

Second, when analyzing the success likelihood of an intrusion under MTD, there are two parts that constitute $P_{success}$, $P(holds(\Omega_{pre}, [t_s, t_f]))$ and P_{static}. However, only how to quantify $P(holds(\Omega_{pre}, [t_s, t_f]))$ is shown. It may be unfair to claim this as a limitation of this thesis because, in a cyber security context, to quantify P_{static} in general is hard. However, for completeness, without quantifying P_{static}, we can never get the absolute value of $P_{success}$, even though a relative analysis for MTD could be good enough. In addition, when we analyze composite attack types, the assumption is that for each sub attack type, P_{static} part is always satisfied, which means we only focus on the analysis of one attempt. However, it is possible that for each sub attack, more than one intrusion needs to be made. For example, under a static system, a sub attack with success probability P_{static}, may fail in the first and second attack, and yet be successful on the third try, leading to a probability of $(1 - P_{static}) \times (1 - P_{static}) \times P_{static}$. However, as this thesis focuses on analyzing and quantifying the security benefit gained as compared to static system, the effort is spent on understanding how an MTD system impacts $P(holds(\Omega_{pre}, [t_s, t_f]))$, and we leave P_{static} as future work.

7.4 Future Work

One area of future work would be to develop a real MTD testbed where machines that provide the same functionality could be adapted to different configurations in specified time intervals. As a comparison, a static system can be implemented as a control and an intrusion testbed can be developed based on penetration tools, such as Metasploit, which launches attacks on the MTD testbed. The results can be used to compare with the results obtained from the theory. Conversely, the theory can be used to guide the settings of MTD testbed

to increase or decrease the success likelihood of intrusion, and in turn validate the theory from newly results gained.

Another future work could be relaxing the assumption for $P_{success}$ analysis that P_{static} is always satisfied, as discussed in the limitation section, and extend it to a more general case where each sub attack type could take n arbitrary times of intrusion until it is successful. In this situation, P_{static} in the definition of $P_{success}$ will need to be updated to $(1 - P_{static})^{(n-1)} \times P_{static}$. But quantifying $P(holds(\Omega_{pre}, [t_s, t_f]))$ could be more challenging, because for each of the first $n - 1$ attacks, $holds(\Omega_{pre}, [t_s, t_f])$ associated with it could be either true or false, and only the $holds(\Omega_{pre}, [t_s, t_f])$ associated with the last attack needs to be true. In addition, during these n attacks, there could be multiple adaptations. Although this direction could provides more general results, from a research perspective, this doesn't add too much for understanding the essence of MTD, because it mixes the analysis of $P(holds(\Omega_{pre}, [t_s, t_f]))$ and P_{static}. However, the way MTD invalidates the intrusion is essentially through invalidating the attacker's knowledge about the system, which only requires or is captured by $P(holds(\Omega_{pre}, [t_s, t_f]))$.

Bibliography

[1] National cyber leap year summit cochair's report. https://www.qinetiq-na.com/wp-content/uploads/2011/12/National_Cyber_Leap_Year_Summit_2009_CoChairs_Report.pdf, 2009. Online, accessed Oct 09, 2013.

[2] National cyber leap year summit participants ideas report. https://www.qinetiq-na.com/wp-content/uploads/2011/12/National_Cyber_Leap_Year_Summit_2009_Participants_Ideas_Report.pdf, 2009. Online, accessed Oct 09, 2013.

[3] Pratyusa K Manadhata and Jeannette M Wing. An attack surface metric. *Software Engineering, IEEE Transactions on*, 37(3):371–386, 2011.

[4] Pratyusa K Manadhata and Jeannette M Wing. A formal model for a systems attack surface. In *Moving Target Defense*, pages 1–28. Springer, 2011.

[5] Security awareness - hardening your computer. http://www.colorado.edu/oit/it-security/security-awareness/hardening-your-computer, 2012. Online, accessed Oct 09, 2013.

[6] Recommended resources for system hardening. https://security.berkeley.edu/node/143, 2013. Online, accessed Oct 09, 2013.

[7] T. Ryutov, C. Neuman, K. Dongho, and Z. Li. Integrated access control and intrusion detection for web servers. In *23rd International Conference on Distributed Computing Systems(ICDCS)*, pages 394–401, 2003.

[8] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for intrusion detection and response. In *DARPA Information Survaivability Conference and Exposition II*, page 1003, 2000.

[9] A. Somayaji and S. Forrest. Automated response using system-call delay. In *Proceedings of the 9th USENLX Security Symposium*, 2000.

[10] S. Musman and P. Flesher. System or security managers adaptive response tool. In *DARPA Information Survivability Conference and Exposition II*, 2000.

[11] P. Porras and P. Neumann. Emerald: event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the National Information Systems Security Conference*, 1997.

[12] C. A. Carver. *Adaptive agent-based intrusion response*. PhD thesis, Texas A and M University, 2001.

[13] D. Schnackenberg, H. Holiday, R. Smith, and et al. Cooperative intrusion traceback and response architecture citra. In *DARPA Information Survivability Conference and Exposition I*, 2001.

[14] W. Lee, W. Fan, M. Miller, S. Stolfo, and E. Zadok. Towards cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1-2):5–22, 2002.

[15] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1/2):169–184, 2007.

[16] B. Foo, Y. Wu, Y. Mao, S. Bagchi, and E. Spafford. Adepts: Adaptive intrusion response using attack graphs in an e-commence environment. In *Proceedings of the*

International Conference on Dependable Systems and Networks, pages 508–517, Piscataway, NJ, 2005.

[17] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *The International Conference on Advanced Information Networking and Applications*, 2007.

[18] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. Using dynamic addressing for a moving target defense. In *Proceedings of the 6th International Conference on Information Warfare and Security. Academic Conferences Limited*, page 84, 2011.

[19] Spyros Antonatos, Periklis Akritidis, Evangelos P Markatos, and Kostas G Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, 2007.

[20] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. Mt6d: a moving target ipv6 defense. In *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*, pages 1321–1326. IEEE, 2011.

[21] Justin Yackoski, Harry Bullen, Xiang Yu, and Jason Li. Applying self-shielding dynamics to the network architecture. In *Moving Target Defense II*, pages 97–115. Springer, 2013.

[22] PaX Team. Pax address space layout randomization (aslr), 2003.

[23] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning. Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 339–348. IEEE, 2006.

[24] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307. ACM, 2004.

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

[26] Stephen W Boyd, Gaurav S Kc, Michael E Locasto, Angelos D Keromytis, and Vassilis Prevelakis. On the general applicability of instruction-set randomization. *Dependable and Secure Computing, IEEE Transactions on*, 7(3):255–270, 2010.

[27] Ehab Al-Shaer. Toward network configuration randomization for moving target defense. In *Moving Target Defense*, pages 153–159. Springer, 2011.

[28] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi 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.

[29] Mihai Christodorescu, Matthew Fredrikson, Somesh Jha, and Jonathon Giffin. End-to-end software diversification of internet services. In *Moving Target Defense*, pages 117–130. Springer, 2011.

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

[31] Yih Huang and Anup K Ghosh. Introducing diversity and uncertainty to create moving attack surfaces for web services. In *Moving Target Defense*, pages 131–151. Springer, 2011.

[32] Rui Zhuang, Su Zhang, Scott A DeLoach, Xinming Ou, and Anoop Singhal. Simulation-based approaches to studying effectiveness of moving-target network defense. In *National Symposium on Moving Target Research*, 2012.

[33] Scott DeLoach, Xinming Ou, Rui Zhuang, and Su Zhang. Model-driven, moving-target defense for enterprise network security. In *Uwe Amann, Nelly Bencomo, Gordon Blair, Betty H. C. Cheng, Robert France (eds) State-of-the-Art Survey Volume on Models @run.time.* Springer LNCS, in press.

[34] Rui Zhuang, Su Zhang, Alex Bardas, Scott A DeLoach, Xinming Ou, and Anoop Singhal. Investigating the application of moving target defenses to network security. In *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, pages 162–169. IEEE, 2013.

[35] Shardul Vikram, Chao Yang, and Guofei Gu. Nomad: Towards non-intrusive moving-target defense against web bots. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 55–63. IEEE, 2013.

[36] Stephen W Boyd and Angelos D Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[37] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. *N-variant systems: a secretless framework for security through diversity.* Defense Technical Information Center, 2006.

[38] Georgios Portokalidis and Angelos D Keromytis. Global isr: Toward a comprehensive defense against unauthorized code execution. In *Moving Target Defense*, pages 49–76. Springer, 2011.

[39] Angelos D Keromytis, Roxana Geambasu, Simha Sethumadhavan, Salvatore J Stolfo, Junfeng Yang, Azzedine Benameur, Marc Dacier, Matthew Elder, Darrell Kienzle, and Angelos Stavrou. The meerkats cloud security architecture. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 446–450. IEEE, 2012.

[40] Hamed Okhravi, Eric I Robinson, Stephen Yannalfo, Peter W Michaleas, Joshua Haines, and Adam Comella. Talent: Dynamic platform heterogeneity for cyber survivability of mission critical applications. In *Secure and Resilient Cyber Architecture Conference (SRCA'10)*, 2010.

[41] Ltd. Coronado Group. Proactive cyber defense – a new moving target defense strategy. http://www.coronadogroup.com/images/Moving-Target-Defense-Coronado.pdf, 2012. Online, accessed Oct 18, 2013.

[42] Yih Huang, David Arsenault, and Arun Sood. Incorruptible system self-cleansing for intrusion tolerance. In *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International*, pages 4–pp. IEEE, 2006.

[43] Mohamed Azab, Riham Hassan, and Mohamed Eltoweissy. Chameleonsoft: a moving target defense system. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 241–250. IEEE, 2011.

[44] Pratyusa K Manadhata. Game theoretic approaches to attack surface shifting. In *Moving Target Defense II*, pages 1–13. Springer, 2013.

[45] Krishna Kant. Configuration management security in data center environments. In *Moving Target Defense*, pages 161–181. Springer, 2011.

[46] Steven M Bellovin. On the brittleness of software and the infeasibility of security metrics. *Security & Privacy, IEEE*, 4(4):96–96, 2006.

[47] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[48] Eric W. Weisstein. Geometric series. From Mathworld. http://mathworld.wolfram.com/GeometricSeries.html. Online, accessed 18, 2014.

[49] Enterprise network definition. http://www.techopedia.com/definition/7044/enterprise-network, 2013. Online, accessed Dec 20, 2013.

[50] Virtualbox. https://www.virtualbox.org/, 2013. Online, accessed Dec 20, 2013.

[51] Vmware workstation. http://www.vmware.com/products/workstation/, 2013. Online, accessed Dec 20, 2013.

[52] Kernel based virtual machine. http://www.linux-kvm.org/, 2013. Online, accessed Dec 20, 2013.

[53] Xen project. http://www.xenproject.org/, 2013. Online, accessed Dec 20, 2013.

[54] Linux containers. https://linuxcontainers.org/, 2013. Online, accessed Dec 20, 2013.

[55] An open virtual switch. http://openvswitch.org/, 2013. Online, accessed Dec 20, 2013.

[56] Puppet Labs. Puppet. http://puppetlabs.com/. Online, accessed Jan 07, 2014.

[57] Chef. Chef. http://www.getchef.com/chef/. Online, accessed Jan 07, 2014.

[58] AnsibleWorks. Ansible. http://www.ansibleworks.com/. Online, accessed Jan 07, 2014.

[59] Saltstack. Salt. http://www.saltstack.com/community/. Online, accessed Jan 07, 2014.

[60] Amazon web services. http://aws.amazon.com/, 2013. Online, accessed Dec 20, 2013.

[61] Windows azure. http://www.azure.microsoft.com/en-us/, 2013. Online, accessed Dec 20, 2013.

[62] Google cloud platform. https://cloud.google.com/, 2013. Online, accessed Dec 20, 2013.

[63] Openstack. http://www.openstack.org/, 2013. Online, accessed Dec 20, 2013.

[64] Netflix - watch tv shows and movies anytime, anywhere. https://www.netflix.com/?locale=en-US, 2013. Online, accessed Dec 20, 2013.

[65] Adobe. http://www.adobe.com/, 2013. Online, accessed Dec 20, 2013.

[66] Cobbler. http://www.cobblerd.org/, 2013. Online, accessed Dec 20, 2013.

[67] Understanding pxe booting and kickstart technology. http://docs.oracle.com/cd/E24628_01/em.121/e27046/appdx_pxeboot.htm, 2013. Online, accessed Dec 20, 2013.

[68] Understanding pxe booting and kickstart technology. http://docs.oracle.com/cd/E24628_01/em.121/e27046/appdx_pxeboot.htm, 2013. Online, accessed Dec 20, 2013.

[69] Metal as a service (maas). https://maas.ubuntu.com/, 2013. Online, accessed Dec 20, 2013.

[70] Juju - automate your cloud infrastructure. https://juju.ubuntu.com/, 2013. Online, accessed Dec 20, 2013.

[71] Software defined networking. http://en.wikipedia.org/wiki/Software-defined_networking, 2013. Online, accessed Dec 20, 2013.

[72] Openflow. http://www.openflow.org, 2013. Online, accessed Dec 20, 2013.

[73] Tech Design Forum. Side channel attacks. http://www.techdesignforums.com/practice/guides/side-channel-analysis-attacks/. Online, accessed Oct 10, 2013.

[74] Ruby Lee. Moving target defense for secure hardware design. http://www.cyber.st.dhs.gov/oct2012pi-presentations/, 2012. Online, accessed Sep 28, 2013.

[75] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[76] Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, et al. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*, volume 81, pages 346–355, 1998.

[77] Hiroaki Etoh and Kunikazu Yoda. Propolice: Improved stacksmashing attack detection. *IPSJ SIG Notes*, 75:181–188, 2001.

[78] Matthew Van Gundy and Hao Chen. Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. In *NDSS*, 2009.

[79] Yih Huang and Arun Sood. Self-cleansing systems for intrusion containment. In *Procceedings of Workshop on Self-Healing, Adaptive, and Self-Managed Systems(SHAMAN)*, New York City, June 2002.

[80] Yih Huang, Arun Sood, and Ravi K. Bhaskar. Countering web defacing attacks with system self cleansing. In *Proceedings of 7th Word Multiconference on Systemics, Cybernetics and Informatics*, pages 12–16, Orlando, Florida, July 2003.

[81] Terracotta. http://terracotta.org/, 2013. Online, accessed Dec 20, 2013.

[82] Quan Jia, Kun Sun, and Angelos Stavrou. Motag: Moving target defense against internet denial of service attacks. In *Proceedings of the International Conference on Computer Communications and Networks ICCCN*, august 2013.

[83] Stuart Staniford, Vern Paxson, Nicholas Weaver, et al. How to own the internet in your spare time. In *USENIX Security Symposium*, pages 149–167, 2002.

[84] Stuart Staniford, David Moore, Vern Paxson, and Nicholas Weaver. The top speed of flash worms. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 33–42. ACM, 2004.

[85] Richard Colbaugh and Kristin Glass. Predictive moving target defense. In *National Symposium on Moving Target Research*, 2012.

[86] Richard Colbaugh and Kristin Glass. Moving target defense for adaptive adversaries. In *Intelligence and Security Informatics (ISI), 2013 IEEE International Conference on*, pages 50–55. IEEE, 2013.

[87] Yih Huang, David Arsenault, and Arun Sood. Scit-dns: Critical infrastructure protection through secure dns server dynamic updates. *Journal of High Speed Networks*, 15(1):5–19, 2006.

[88] Jeff Rowe, Karl N. Levitt, Tufan Demir, and Robert Erbacher. Artificial diversity as maneuvers in a control theoretical moving target defense. In *National Symposium on Moving Target Research*, 2012.

[89] Sandeep Bhatkar, Ron Sekar, and Daniel C DuVarney. Efficient techniques for comprehensive protection from memory error exploits. In *Proceedings of the 14th USENIX Security Symposium*, pages 271–286, 2005.

[90] Martin Rinard. Manipulating program functionality to eliminate security vulnerabilities. In *Moving Target Defense*, pages 109–115. Springer, 2011.

[91] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.

[92] The UCI KDD Archive. Kdd cup data. http://http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. Online, accessed Oct 18, 2013.

[93] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM transactions on Information and system Security*, 3(4):262–294, 2000.

[94] Carson Brown, Alex Cowperthwaite, Abdulrahman Hijazi, and Anil Somayaji. Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–7. IEEE, 2009.

[95] Ang Cui and Salvatore J Stolfo. Symbiotes and defensive mutualism: Moving target defense. In *Moving Target Defense*, pages 99–108. Springer, 2011.

[96] Martin J Osborne. *A course in game theory*. Cambridge, Mass.: MIT Press, 1994.

[97] Michael Crouse and Errin W Fulp. A moving target environment for computer configurations using genetic algorithms. In *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on*, pages 1–7. IEEE, 2011.

[98] Justin Yackoski, Peng Xie, Harry Bullen, Jason Li, and Kun Sun. A self-shielding dynamic network architecture. In *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*, pages 1381–1386. IEEE, 2011.

[99] Marco M Carvalho, Thomas C Eskridge, Larry Bunch, Jeffrey M Bradshaw, Adam Dalton, Paul Feltovich, James Lott, and Daniel Kidwell. A human-agent teamwork command and control framework for moving target defense (mtc2). In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, page 38. ACM, 2013.

[100] Christopher Zhong and Scott A. DeLoach. An investigation of reorganization algorithms. In *the International Conference on Artificial Intelligence (IC-AI'2006).*, June 2006.

[101] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, nov 2002.

[102] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. *Managing Cyber Threats: Issues, Approaches and Challanges*, 2003.

[103] R.P. Lippmann, K.W. Ingols, C. Scott, K. Piwowarski, K.J. Kratkiewicz, M. Artz, and R.K.Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical report, MIT Lincoln Laboratory, 2005.

[104] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to at-

tack graph generation. In *13th ACM Conference on Computer and Communications Security*, Oct 2006.

[105] Cynthia Phillips and Laura Painton Swiler. Graph-based system for network-vulnerability analysis. In *NSPW 98: Proceedings of the 1998 Workshop on New Security Paradigms*, 1998.

[106] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *the IEEE Symposium on Security and Privacy*, may 2002.

[107] S. Schmidt, R. Bye, J. Chinnow, K. Bsufka, A. Camtepe, and S. Albayrak. Application-level simulation for network security. *SIMULATION*, 86:311–330, 2010.

[108] John Homer and Xinming Ou. Sat-solving approaches to context-aware enterprise network security management. *Selected Areas in Communications, IEEE Journal on*, 27(3):315–322, 2009.

[109] Manish Jain, Bo An, and Milind Tambe. Security games applied to real-world: Research contributions and challenges. In *Moving Target Defense II*, pages 15–39. Springer, 2013.

[110] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[111] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe, and Fernando Ordóñez. Software assistants for randomized patrol

planning for the lax airport police and the federal air marshal service. *Interfaces*, 40 (4):267–290, 2010.

[112] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[113] Munindar P. Singh. Towards a science of security. http://www.computer.org/portal/web/computingnow/archive/january2013, 2013. Online, accessed June 30, 2014.

[114] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[115] Thomas Hobson, Hamed Okhravi, David Bigelow, Robert Rudd, and William Streilein. On the challenges of effective movement. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 41–50. ACM, 2014.

[116] Mark Burgess and Alva L Couch. Modeling next generation configuration management tools. In *LISA*, pages 131–147, 2006.

[117] Nelly Bencomo, Jon Whittle, Pete Sawyer, Anthony Finkelstein, and Emmanuel Letier. Requirements reflection: requirements as runtime entities. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 199–202. ACM, 2010.

[118] Peter Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems.

In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 95–103. IEEE, 2010.

[119] Rui Zhuang, Scott A. DeLoach, and Xinming Ou. A model for analyzing the effect of moving target defenses on enterprise networks. In *Proceedings of the 9th Cyber and Information Security Research Conference*, Oak Ridge, Tennessee, April 2013. ACM.

[120] David John Leversage and ERIC James. Estimating a system's mean time-to-compromise. *Security & Privacy, IEEE*, 6(1):52–60, 2008.

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

[122] Rui Zhuang, Alexandru G Bardas, Scott A DeLoach, and Xinming Ou. A theory of cyber attacks: A step towards analyzing mtd systems. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 11–20. ACM, 2015.

[123] Ross Sheldon et al. *A first course in probability*. Pearson Education India, 2002.

[124] Alexander Bogomolny. Number of trials to first success from interactive mathematics miscellany and puzzles. http://www.cut-the-knot.org/Probability/LengthToFirstSuccess.shtml, 2015. Online, accessed Aug 15, 2015.

[125] Jafar Haadi H Jafarian, Ehab Al-Shaer, and Qi Duan. Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 69–78. ACM, 2014.