FOUNDATIONS OF ADAPTIVE CYBER DEFENSE
AGAINST ADVANCED PERSISTENT THREATS

by

Luan Huy Pham
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Dr. Massimiliano Albanese, Dissertation Director
_____	Dr. Sushil Jajodia, Committee Member
_____	Dr. Emanuela Marasco, Committee Member
_____	Dr. Kai Zeng, Committee Member
_____	Dr. Deborah Goodings, Associate Dean
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Spring Semester 2020
George Mason University
Fairfax, VA

Foundations of Adaptive Cyber Defense Against Advanced Persistent Threats

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Luan Huy Pham
Master of Science
George Mason University, 2010
Bachelor of Science
George Mason University, 2007

Director: Dr. Massimiliano Albanese, Associate Professor
Department of Information Science and Technology

Spring Semester 2020
George Mason University
Fairfax, VA

Dedication

For my dad;
always been
always will be
my hero

Acknowledgments

I would like to thank my dissertation director, Dr. Albanese, for his time and immense patience over the years. I would also like to thank my co-authors, friends and other loved ones throughout the years who have supported me. I would not have made it without you.

The work presented in this thesis, was supported in part by the National Science Foundation under award CNS-1822094 and by the Army Research Office under award W911NF-13-1-0421.

Table of Contents

List of Tables

List of Figures

Abstract

FOUNDATIONS OF ADAPTIVE CYBER DEFENSE AGAINST ADVANCED PERSISTENT THREATS

Luan Huy Pham, PhD

George Mason University, 2020

Dissertation Director: Dr. Massimiliano Albanese

The term Advanced Persistent Threats (APTs) refers both to highly-sophisticated, often nation-state attackers with tremendous resources and to the malware they employ to compromise their target to which no organization has proven immune. Dynamic and deception-based defense techniques offer a possible solution. Such techniques, including Moving Target Defense (MTD) and Adaptive Cyber Defense(ACD) techniques, prevent or delay attacks against computer networks by dynamically altering characteristics of the systems or network in a manner to present attackers with a variable, possibly deceptive attack surface and disrupt the planning or execution of cyber-attacks.

To better leverage these techniques, this work proposes a novel model to capture how advanced, stealthy adversaries, including APT actors, acquire knowledge about the target network and establish and expand their foothold within the system. This model quantifies the cost and reward, from the adversary's perspective, of compromising and maintaining control over targets within the network. With this foundational understanding of attacker incentives and deterrents, as well as their predicted position in the network, existing defenses can be refined and innovative defenses can be built specifically to counteract the threat posed by APTs.

Chapter 1: Introduction

"APTs are the most sophisticated form of cyber weapon that exist" [1]

The term Advanced Persistent Threat is used to designate both a highly-motivated, sophisticated adversary and the cutting-edge cyber malware which they often employ. APTs (the adversary) are often either strongly affiliated with - or a directly-funded cyber branch of - nation states. While known for their cyber expertise, these adversaries leverage multiple avenues of attack to accomplish their objectives, including psychological operations involving social media as well as traditional intelligence operations [1][2]. Their overall objectives often include exfiltrating information or otherwise undermining or impeding critical aspects of a mission, program, or organization. The United States National Institute of Standards and Technology (NIST) [3] recognizes APTs by 1) the repeated pursuit of objectives over an extended period of time, 2) adaptation to defender efforts, and 3) determination to maintain interaction to execute their objectives.

Perhaps the most reknown example of an APT is captured in the high-profile "APT1" report [4] by Mandiant (now a part of FireEye). This report described a cyber-focused branch of the Chinese People's Liberation Army (PLA), which became collectively known as APT1. APT1's objective was apparently to conduct espionage. To this end, the researchers who authored the report observed APT1 compromising 141 companies spanning 20 major industries employing advanced malware platforms. In one instance, APT1 exfiltrated 6.5 terabytes of compressed data from a compromised target, over a ten-month time period. In another case APT1 maintained access in a compromised networks for a period of four years and ten months - demonstrating both patience and persistence which are defining characteristics of APTs. In the report, Mandiant acknowledged it was very likely that its

observations represented only a small portion of APT1's total activities, indicating that the overall scope was likely much larger.

As a whole, China has been reported to possess the world's second-most powerful cyber arsenal overall and the largest number of APT groups directly attributed to the state [2]. Furthermore, these groups have also demonstrated a willingness to deploy their capabilities broadly, monitoring opposing political groups in Tibet, Taiwan and Hong Kong; conducting worldwide industrial espionage (as in the case of APT1), and even infiltrating the infrastructure of other nation-states. In relation to the United States, Chinese state-sponsored groups have been linked to cyber operations to gather highly detailed intelligence, including high-profile breaches of the US Office of Personnel and Management (OPM) and the US Department of Defense (DOD) NIPRNET. As a result, these groups have obtained information as varied as biometric data for 5.6 million US Federal employees and the technical data for the advanced F-35 fighter aircraft. China has likely implanted various forms of malicious code into critical American systems, presumably to be leveraged in the event of conflict [1].

China is not the only nation which has leveraged cyber capabilities. The threat of APTs is widespread. Iranian-affiliated groups have conducted regional operations against oil-producing rivals in Saudi Arabia. India and Pakistan routinely engage in cyber conflict. Similarly, North Korean and South Korea spar in the cyber domain; with North Korea also implicated in the highly-publicized Sony Pictures hack. Russian-affliated groups have conducted their own wiper attacks to destabilize Ukraine [2]. More notably, the breach of the US Democratic National Committee by these Russian-affliated groups and the subsequent release of politically-damaging documents arguably influenced the outcome of the 2016 United States Presidential Election [1][2]. It is conceivable that APTs have influenced the course of history and will do so again in the future.

The repeated compromises of nation-states make it evident that no entity is invulnerable to the threat of APTs. In addition, the techniques pioneered by APTs have become increasingly deployed by criminal organizations. The MalwareBytes Labs 2019 State of

Malware Report [5], noted the use of zero-day exploits which were publicly-leaked by the ShadowBrokers hacking group and were originally developed by the the Equation Group APT. Instead of geopolitical gain, these criminal organizations aim to benefit financially, most often through ransom demands or through sale of information obtained during a data breach. Early detection of these breaches is critical. The Ponemon Institute has conducted yearly studies on the cost of organizational data breaches for 14 years. The studies have concluded that earlier detection of malicious activity has tangible economic benefits for an organization. In 2019, organizations which required over 200 days to identify that a breach had occurred faced 36% higher costs compared to organizations which identified breaches in under 200 days [6]. The delay in detecting malicious activity can result in adversaries having the opportunity to gain control of a larger portion of the network and exfiltrate larger amounts of sensitive information – which may include critical trade secrets or closely-guarded strategic data.

Given these considerations, mitigating the threat of APTs has indisputable value and the work presented in this thesis represent an important step in that direction.

1.1 Research Approach and Thesis

APTs are able to penetrate networks inaccessible to other threats through commitment of both extraordinary resources and effort. Once a foothold is established, they maneuver and spread through the target's network, pilfering information or committing acts of sabotage as they see fit. Contending against such threats is a daunting challenge, one that has victimized organizations large and small. However, though formidable, these attackers are also not infallible.

While certainly a growing threat, APTs are not yet ubiquitous. Attacker resources, while intimidating, are not inexhaustible. Attackers must make calculated decisions involving tradeoffs based upon considerable, but not unlimited knowledge. Therefore, one may conclude that there exist avenues for defenders to deter attackers. To develop such countermeasures, a strong understanding of how APTs operate, as well as their strengths and

weaknesses is required.

This work establishes a framework to model attacker behavior as well as incentives and deterrents to serve as a foundation upon which to build future safeguards against APTs, particularly dynamic defenses. Dynamic - as opposed to static - defenses seek to alter components of the target system over time to deny APT advantages and mitigate their impact.

In developing this work, a comprehensive analysis of the unique characteristics of the APT malware was required, including their objectives, means of operation, and the factors which make them far more effective than traditional cyber attacks. This required an exhaustive review of industry technical reports, summary analysis and the current state-of-the-art. A major focus of this analysis was to determine common limitations which constrain APTs, which also provides an explanation for their relative rarity.

In examining the available reports and research regarding APTs, it became clear that conventional graph-based attack modeling does not accurately portray APT behavior. These prior efforts tended to model networks with relatively few target devices. These target devices were often critical databases or other high-value objective. APTs challenge this traditional paradigm. While prior models assume relatively few targets of value, in practice APTs have shown that they can seize value from nearly every device throughout the network. Thus, I sought to develop a more accurate model which accounts for APT reward incentives as well as the cost deterrents. This initial model also considered how the APT may shift and slowly grow its presence within the network over time.

Once an initial framework was developed, it was quickly leveraged in a testbed lab environment across networks of varying sizes. Furthermore, the framework accounted for the effect of notional defenses, both static and dynamic - quantitatively measuring the difference in effectiveness.

Afterwards, I evolved the framework, imparting additional granularity to capture how reward values may change due to the effect of the attacker as well as common activity within the network. In a significant improvement, the framework differentiated the views

of the attacker and defender - opening the possibility for suboptimal decisions due to incomplete knowledge. Such a step is necessary to model the effects of cyber deception upon an adversary - which would not be possible against an omniscient attacker.

I then tested the impact of multiple MTDs techniques, implemented concurrently, against a simulated persistent attacker. For these simulations, the attacker employed real-world exploits to compromise vulnerable software platforms and operations in a testbed environment. This testing demonstrated that MTDs used in concert, when properly configured, disrupted attackers with greater success than each MTD implemented individually. However, doing so incurred additional service disruption; the extent of which was dependent upon how each MTDs was configured both independently, and in relation to other implemented MTDs. This served as evidence that dynamic defenses require careful consideration and tuning to provide the the greatest benefit while minimizing tradeoffs. Without proper understanding of the adversary and their capabilities, informed decisions regarding the configuration of dynamic defenses and the selection of techniques to deploy are impossible.

As the final step in developing this dissertation, I grounded the abstract theory of the framework in metrics collected from a high-fidelity testbed which is used to conduct public and commercial cybersecurity research.

1.2 Organization

This dissertation is organized as follows. Chapter 2 covers background information regarding APT malware and adaptive cyber defense. Chapter 3 establishes preliminaries, underlying assumptions, the attacker reward/cost model of incentives and deterrents, as well as the problem statement. Chapter 4 describes the initial solution, a greedy algorithm to construct an attacker's malware footprint within a network. Chapter 5 extends the model and the solution, relaxing assumptions regarding defender vs attacker knowledge and capturing additional granularity for how node values can change over time. Chapter 6 examines the use of multiple MTDs used in concert at differing settings to demonstrate the need to customize their use against threats. Chapter 7 further grounds the model by associating

rewards and costs to measurable values extracted from simulated networks in a high-fidelity testbed environment against an attacker which minimizes activity to maximize stealth.

The dissertation concludes with a summary of findings and discussion of future work.

Chapter 2: Background and Related Work

The framework proposed in this dissertation is built upon a prior body of research regarding the unique characteristics of APT malware, an overview of their primary objectives, and the level of resources which an APT actor is willing to expend to develop and maintain the APT malware. A taxonomy of APT characteristics is presented and several notable APTs are selected and examined leveraging the taxonomy. Mechanisms by which they bypass even state-of-the-art cyber defenses while also evading detection. Finally, categories of ACD/MTD are discussed, accentuating properties which uniquely counter the strengths of APTs.

2.1 APT Malware Characteristics

APT malware exhibits unique characteristics compared to other malware. The most evident is their relative size and complexity compared to other contemporary malware. This section discusses other features which are critical considerations.

2.1.1 Data Exfiltration

Of the 66 APTs identified by the Kaspersky Global Research and Analysis Team, 61 are classified as having the primary function of exfiltrating data, including credential theft, and cyber espionage [7]. The remaining APTs generally fall under the category of cyber sabotage, as it is the case for the renown Stuxnet APT. While perhaps not a primary function, it has been shown that many of these threats (e.g., Shamoon) also include some form of reporting function to transmit data to attacker-operated command and control (C&C) servers. Even Stuxnet, known primarily for sabotaging Iranian nuclear centrifuges, incorporated extensive features to transmit system information to the C&C infrastructure

2.1.2 Cyber Evasion

A defining characteristic of APT malware is their ability to evade detection versus traditional intrusion detection systems (IDS). APT actors spend considerable resources to develop stealth features. Arguably the most well-known example of an APT campaign, Operation Olympic Games purportedly involved the development and deployment of the Stuxnet malware to degrade the progression of the Iranian nuclear program.

Furthermore, the APT actors deployed Stuxnet in a manner which would further promote stealth. While the malware possessed the capability to catastrophically damage all affected centrifuges at any given time, instead, the centrifuges were subtly degraded in a manner which would not be immediately evident. In effect, this allowed the malware to delay the overall progress of the Iranian nuclear program to an extent greater than a conventional military strike.

2.1.3 Cost

Development of APT malware is a considerable endeavor. Nation-states commit massive resources to the development of APT malware and supporting related operations. In the aforementioned "APT1" report [4], the it was estimated that thousands of personnel were employed, including malware authors, industry experts, translators, IT support staff, and other associated logistical/overhead personnel. Despite these costs, an analysis of the Stuxnet malware, determined that as much as 50% of the development cost was allocated to stealth-related features [8].

One of the most well-known characteristics of APT malware is the usage of zero-day exploits, which are exploits which are not generally known to the public or to the security community at large. By definition, no IDS signature exists for the exploit, nor has the a software vendor provided a specific patch for the exploit. These properties make zero-days extremely valuable for APT threat actors who strive for stealthy operations and can devote the resources to either discover or otherwise procure the exploits. Even so, a recent study [9] conducted by the RAND corporation over 14 years from 2002-2016 concluded that the costs

to acquire zero-day exploits are considerable.

Merely discovering a vulnerability often requires months of expert time as well as potential capital expenditures, if specialized hardware is required. Even after the vulnerability is discovered, additional time and expertise is required to develop and test a working exploit - which the authors estimate to cost $29.914.95. Purchasing an exploit can involve expenditures of hundreds of thousands to millions of U.S. dollars. While such a cost may seem achievable for many large organizations, also consider that many APTs leverage multiple zero-day exploits. Furthermore, these costs do not include other expenditures involved in developing other aspects of the malware, including the payload. Furthermore, there are other costs to consider as well. Insertion of the malware into the target environment may require traditional intelligence or covert operation, which requires a high degree of sophistication and a class of expertise outside of exploit development or operation of the malware itself, which must also be considered.

2.2 APT Taxonomy

Table 2.1 below identifies attributes of APT malware and provides definitions of each. Furthermore, we provide taxonomies and accompanying definitions in the following subsections.

Table 2.1: Malware Attributes

Attribute	**Description**
Alternative Name(s)	APT malware often can be referenced via several names due to independent discovery and/or other factors. While this survey references the name of the APT malware by its most well-recognized name, we include alternative names for completeness.

Discovery Date	the initial date of discovery as identified by the security community at-large. Discoveries made privately and not disclosed are excluded.
Suspected Target(s)	The primary target entities of the threat agent as determined by the security community at-large. APT malware often is detected as a result of it spreading beyond its primary target.
Suspected Objectives(s)	The primary goal of the suspected threat agent based on the analysis of the the security community at-large.
Suspected APT Actor(s)	The actor which the the security community at-large suspects is the most likely entity to deploy the APT malware.
Component(s) Exploited	Components which are specifically exploited by APT malware.
Payload Feature(s)	Features utilized to complete the primary objective of the APT malware.
Propagation Method(s)	Methods which APT malware has been demonstrated to spread.
Stealth Mechanism(s)	The primary goal of the suspected threat agent based on the analysis of the the security community at-large.
Zero-Day Exploit(s)	APT malware is often associated by the use of zero-day exploits which were not known publicly prior to their use in the APT malware. As such, no antivirus signature for the exploit existed prior to the discovery of the APT malware. We list the known zero-day exploits of the APT malware by their MITRE CVE number and description.

2.2.1 Component(s) Exploited

Table 2.2 below describes a taxonomy of components which are specifically exploited by APT malware. APT malware tends to exploit similar components as general malware. Furthermore, software targeted for exploitation typically have a significant install base. If the components exploited are very limited, this typically indicates a more targeted attack with a specific objective in mind. Analysis of this attribute is often a factor in how the security community-at-large determines both Suspected Target Entities, the Suspected Objective(s), as well as the Suspected APT Actor(s). For example, if the malware targets only a specific type of industrial hardware, this significantly narrows the range of potential targets. Furthermore, if the specific hardware requires specialized expertise, then the range of potential APT Actor(s) is limited to those which has access to such expertise.

Table 2.2: Component(s) Exploited

Component	**Description**
Industrial Control Systems (ICS) / Supervisory Control and Data Acquisition (SCADA)	Various systems which control industrial infrastructure are often highly specialized to reflect the highly specialized equipment they control. As a result, flaws which are present in these system often remain undiscovered for extended lengths of time.
Office Applications / Documents	While the Microsoft Office Suite may represent the most well-known office suite, different countries and industries often utilize unique office applications. APT actors have many incentives to compromise such applications. As the products generally have a wide user and install base, any developed exploits would affect a larger . Users devote significant portions of their time with these products and may overlook suspicious signs or behavior.

Multimedia Software Platforms	These software platforms add rich media capabilities, including animations and video across many operating systems and are often embedded into major web browsers. The most common example, Adobe Flash, was installed on 96.3 percent of all Windows PCs [10]. This wide user base, rich functionality, and relatively low level of concern for security during its initial development has made such platforms an enticing target for exploitation. From November 2015 to November 2016, an examination of 144 exploit kits by Recorded Futures showed that Flash vulnerabilities comprised 6 of the top 10 kits [11].
Operating System Kernel	Compromise of the core, built-in functionality of the operating system. While many operating systems have incorporated security features and practices as a major portion of the development lifecycle, malware authors are incentivized to develop exploits to compromise the operating system directly as such exploits would not require other platforms to be installed. Furthermore, kernel-level exploits operate with permissions at the highest allowed level, making them particularly enticing.
Software Frameworks	This category includes the Java and .Net Frameworks. This category is A Bit9 (now Carbon Black) report identifies Java as the most targeted endpoint technology, with Java-related vulnerabilities in the top ten of all NIST CVE vulnerabilities and over 90% of enterprises running versions at least 5 years old [12].

Internet of Things (IOT)	This is a broad category encompassing devices embedded with electronics and networked. This category includes sensors, appliances, implants, small home office devices, network storage devices and other devices of similar nature.

2.2.2 Payload Feature(s)

Table 2.3 below describes a taxonomy of features utilized by the APT malware to accomplish their primary objective once resident within the target network.

Table 2.3: Payload Feature(s)

Feature	**Description**
Backdoor	Allows the threat agent access to the target entity systems, typically for the purposes of data gathering and espionage.
Beaconing	Activity in which the APT malware maps out the network of the target entity, potentially compiling statistics and report back to the actor.
Infostealer	A generic term for specialized APT malware software/modules to exfiltrate data. These modules are often tailored for the individual target and the types of exfiltrated information/data.

Keylogger	The APT malware includes software that records user keystrokes. Typically, not all traffic is captured. Instead, the keylogger may only be activated, for instance, upon the user visiting logins to websites, enabling credential theft.
Packet Sniffer	The APT Malware will capture traffic which traverses the network and report back to the APT actor. Typically, not all traffic is captured. Instead, traffic for certain applications may be monitored and reported back to the APT actor.
Sabotage/Destructive Code	The APT malware contains code intended to inflict destructive harm to the target entity. This is
Remote Access Trojan (RAT)	Also known as a Remote Administration Tool, this is a software tool installed on compromised systems to enable the threat agent to remotely control the victims systems in real time without the victims knowledge.
Wiper	A software tool used to erase bulk amounts of data. This feature is employed in cases where the APT malware's primary goal is sabotage.

2.2.3 Propagation Method(s)

Table 2.4 below identifies methods which the APT malware employs to spread.

Table 2.4: Propagation Method(s)

Feature	**Definition**

Compromised Files	The APT malware propagates via execution of compromised files when loaded
Local Network Compromise	Encompasses any method of propagation from a compromised endpoint directly communicating via the local network to other, uninfected endpoints.
Shared Network Resources	Encompasses any propagation method from a compromised endpoint compromising a shared network resource which would then consequently infect other endpoint which would also utilize that resource.
Phishing	A form of social engineering attack where internal users are targeted and encouraged to either open mail or mail attachments which execute malware. This technique is not exclusive to APT malware and has been shown to both be effective and low-cost compared other vectors. This propagation method includes spear phishing, where emails are customized for specific organizations and whaling, where specific high-value persons (such as corporate executives) are specifically targeted. Attackers often carefully select themes and topics used in attacks, using actual news stories which copy the text directly from the news websites. Other attacks can appeal to personal hobbies.
Removable File Storage	This propagation method specifically relates to malware which spreads through some form of auto-run feature typically employed by operating systems handling removable media such as USB drives, as opposed to manual file transfer.

2.2.4 Stealth Mechanism(s)

Table 2.5 below identifies mechanisms which APT malware utilize to maintain stealth before, during and after compromise of a target system. Arguably, APT malware has demonstrated advanced techniques to evade detection compared to general malware [13][14]. Several of these techniques reduce the traffic volume of the APT malware. Per Jafarian, detectability [15] is affected by the rate of committing illegitimate actions during an attack. Intuitively, APT malware which reduces its traffic volume also reduces its detectability.

Table 2.5: Stealth Mechanism(s)

Feature	**Definition**
Automated Self-Destruct	The APT malware includes the ability to wipe itself from the compromised system in order to prevent both detection and forensic analysis. This feature can be triggered via system clock - once a date is passed, the malware will automatically remove itself.
Compromised Certificates	The APT malware uses certificates which are fraudulently signed by a trust authority to allow elevated execution permissions for binaries.
Compromised Credentials	The APT malware uses previously-compromised credentials to enable use of privileged functions.

Compromised Third-Party	A third party is compromised to facilitate attacks on the primary target by exploiting an existing trust relationship between the target and the third party. • Compromised Third-Party Credential: The APT malware employs the use of credentials which are nominally considered private to a third party. • Compromised Third-Party Network: The APT malware routes traffic through the third party website or network to defeat reputation-based scoring defenses [16]. • Compromised Third-Party Security Product/Service: The Target Entity employs a security product or service from a third party. Thus, the APT compromises the Product/Service, enabling the malware.
Antivirus Tampering	The APT malware searches for antivirus products on the compromised system and either disables them or otherwise renders them ineffective.
Crypter	As antivirus and intrusion detection systems often rely on signature-based methods of detection, encryption of either the binary and/or C&C traffic encryption introduces an impediment to both initial discovery, forensic analysis and ongoing detection efforts as well.

Fileless	The APT malware avoids leaving files on the hard disk of the system, running exclusively in system memory. As memory is volatile, a shutdown or reboot of the system will also erase the evidence of the malware activity.
Packer	Compresses the malware file, reducing the effectiveness of signature-based detection.
Polymorphic Code	The APT malware mutates its code, but maintains identical functionality.
Rootkit	The APT malware directly manipulates the operating system of of the target system to mask activity. For example, CPU utilization may be reported as less than its actual usage to hide the computational processing employed by the APT malware.
Selective Targeting	While the APT malware does infect a variety of systems, it is relatively dormant on systems which do not match its target profile. Malware with this attribute often spread through networks to target SCADA systems.
Service Injection	The APT malware .
Suspended Operation	After certain criteria is met, the malware ceases operation, though remains on the compromised system. Often, the malware retains a backdoor for the APT actor to issue commands at a later date.
Threshold Propagation	The APT malware limits the number of times it may spread it order to restrict its overall propagation.
Wiper	Similar to the payload feature mentioned above, a wiper employed as a stealth feature is used to remove evidence of the APT malware's presence and evade detection.

2.2.5 Other Notable Feature(s)

Table 2.6 below describes other features of APTs which are otherwise notable, but do not fall within the prior aforementioned categories.

Table 2.6: Other Notable Features

Attribute	**Description**
Bootkit	The APT malware modifies the master boot record of the compromised system, allowing the malware to survive operating system reboots.
Command and Control (C&C) Server Communication	Typically used in conjunction with a Backdoor or RAT, C&C communication often serves as a "dead-drop" for APT malware performing data extraction. Furthermore, the a connection with a C&C infrastructure allows APT malware to be periodically updated to add new features, or, in the case of discovery, manually initiated self-destruct to prevent forensic analysis.
Peer-to-Peer (P2P) Communication	The APT malware communicates within the local LAN. This feature allows the APT malware to reduce the outbound communication volume when performing data exfiltration or inbound traffic receiving upgrades or commands.

Modular/Plugin Architecture	The APT malware is designed with an extensible plugin architecture. This often allows the malware to download relevant plugins as necessary. This reduces the overall footprint of the malware as unnecessary plugins are not downloaded, and allows for modular updates as individual plugins can be updated independently, without having to re-download an entire binary.

2.3 Selected Notable APT Malware

Present discussion of APT malware must acknowledge a considerable amount of uncertainty. Attribution of cyber attacks is a notoriously difficult problem [17][18][19][20]. However, speculative analysis performed by intelligence agencies, cybersecurity-based firms and other similar organizations will often implicate a certain entity or multiple entities as the primary threat agent. The threat agent(s) deploy the APT malware against the target(s) and may also be involved with the development effort as well.

Organizations based their analysis on various Tactics, Techniques and Procedures (TTPs) required to develop, test, operate the APT malware. This may eliminate entities which clearly lack the available resources to ultimately deploy the malware. Direct examination of the APT malware code or behavior also can suggest particular authors, such as through examination of metadata keyboard languages or with compile times that correspond to typical working hours in particular time zones. Furthermore, the analysis also examines the potential motivating factor which would drive the use of the malware. Actors without a direct motivating factor are regarded as less likely than more actors with a direct motivating factor.

This approach does lead to some obvious flaws. Nation-states typically disguise their full capabilities as national security concern. Malware authors will attempt to obfuscate, as

part of general best practices [21][22]. Furthermore, techniques used to obfuscate the threat agent could be used to deliberately masquerade as another entity entirely in what are known as "false flag" attacks. This could spark retaliatory action against the falsely implicated entity, which may have been the primary objective of the threat agent. Analysis of motives is similarly fraught with potential errors. This analysis is based largely on public and known sources of information, whereas most organizations retain vast troves of non-public information.

Analysis of malware in general, and the more sophisticated APT malware in particular, is an uncertain endeavor. However, this intends to convey the consensus of the security community at-large.

2.3.1 Stuxnet

The Stuxnet [23][20][24][25][26][27] worm, recognized as the first cyber weapon, was employed to sabotage the Iranian nuclear program. According to David Sanger[28], a NY Times journalist, Stuxnet was a joint United States and Israeli effort which began as reconnaissance malware under the second Bush administration. At that time, the United States had mistakenly accused Iraq of seeking nuclear capability and committed its military strength to topple the Iraqi government. When no such weapons were found, it would become politically damaging for the United States to accuse yet another nation of developing nuclear weapons. As a result, Iran seized the opportunity to accelerate their own nuclear ambitions. The Israeli government, seeking to prevent the development of an Iranian nuclear bomb, considered a conventional strike which could potentially embroil the entire region into war. Stuxnet posed an alternative.

Noted Stuxnet expert Ralph Langer contends that the malware was relatively "low-yield" by design, intending on slowing the development of the Iran nuclear program rather than destroy all of of the centrifuges [8] at once. While Stuxnet retained the capability, Langer's analysis shows that even catastrophic destruction of all the centrifuges at any given point would not have significantly slowed the Iranian nuclear development compared

to the employed approach, which hampered the Iranian efforts over a longer period of time. Langer postulates that Stuxnet delayed the Iranian nuclear development by roughly two years.

Table 2.7: Stuxnet Malware Attributes

Attribute	**Description**
Alternative Names	Operation Olympic Games
Discovery Date	June 2010
Suspected Objectives	Sabotage of the Iranian uranium enrichment facility at Natanz; and the overall Iranian nuclear development program.
Suspected Threat Agent	United States National Security Agency (NSA) and Israeli Unit 8200 (Israeli equivalent to the US CIA).
Components Exploited	- Industrial Control Systems (ICS)/Supervisory Control and Data Acquisition (SCADA) - Cascade Protection System - Centrifuge Drive System

Table 2.8 below describes payload features of Stuxnet:

Table 2.8: Stuxnet Payload Features

Feature	**Description**
Backdoor	If possible, upon newly infecting a system, Stuxnet attempted to reach out to C&C servers. The C&C servers would presumably register the system and maintain a RPC backdoor for external commands [24].
Beaconing	Stuxnet recorded the information regarding the system's name, IP address, domain, OS version and whether it had PLC-related files installed in a "Configuration Data Block" which was preserved upon each infection. This preserved a history of the infection, allowing the operator to map the highly restricted environments where it was inserted. Furthermore, this feature would allow the discovery of additional facilities and previously-unknown contractor relationships. Later, this feature also allowed the security community to trace the path of the malware and determine the original point of infection [24][26].

Sabotage Module	Stuxnet featured two separate means of sabotage among its multiple variants. Both targeted the Siemens PLCs which controlled the centrifuges enriching uranium for the the Iranian nuclear program. The lesser-known first method, was through an overpressure attack against gas centrifuges, which are sensitive to increased process pressure and would cause a variety of issues. The Natanz facility design featured a unique system which addressed the excess gas pressure. An earlier variant of Stuxnet interfered with this system [8]. The second, more renown method was to cause the centrifuge rotor fans to rapidly accelerate and decelerate causing physical stress on relatively delicate components [29]. Both variants operated once a month to evade discovery.

Table 2.9 below describes propagation methods of Stuxnet:

Table 2.9: Stuxnet Propagation Methods

Method	**Description**
Compromised Files	Stuxnet would install itself into Step7 project directory used by Siemens software. The entire directory would often be copied from system to system to facilitate configuration. Whenever the files would be loaded, Stuxnet would execute [24].
Local Network Compromise	Stuxnet spread through Windows Print Spooler and SMB vulnerabilities [24][25][26]
Shared Network Resources	Stuxnet infected both windows shares and Siemens WinCC servers. Vulnerable systems connecting to these shared resources would also be infected.
Removable File Storage	Stuxnet infects USB drives and will infect vulnerable systems through the use of the Windows AutoRun feature.

Table 2.10 below describes stealth mechanisms of Stuxnet:

Table 2.10: Stuxnet Stealth Mechanisms

Method	**Description**
Compromised Certificates	Stuxnet employed drivers digitally signed by Realtek Semiconductor and JMicron Technology. With the signed driver, compromised systems allowed to allowed the drivers to execute silently at an elevated permission level. As Realtek and JMicron have facilities which are physically located in proximity to one another, it was speculated that the certificates where physically stolen [24][30].
Crypter	Stuxnet would perform XOR-based encryption with a static 31-byte string on many of its files, including the main payload, communications, along with data, configuration and log files [24].
Rootkit	One of Stuxnet's most significant features was the use of rootkits, both to mask its activities on compromised Windows systems and more uniquely, the Siemens PLCs. For both attacks which directly damaged the centrifuges, Stuxnet masked the operation of the attack from the operators. During the overpressure attack, process input signals are record for 21 seconds and replayed for the duration of the attack. During the rotor fan attack, the user-configured rotor speed is displayed to the user, despite the actual rotor speed being manipulated by the attack code [8]. The attack would still be detected via direct observation of the audible centrifuge noise, which would change drastically based on the rotor speed.

Selective Targeting	On systems without files related to PLC software, Stuxnet is relatively dormant, only seeking to further spread.
Suspended Operation	Stuxnet had hard-coded dates in which the variant was to cease either all or various aspects of its operation [31].[25]. This included several of its propagation methods, which would be dependent on particular exploits. This may have been an effort to prevent detection of Stuxnet should a leveraged vulnerability be discovered.
Threshold Propagation	For each USB drive infected, Stuxnet's code would attempt to spread for only 21 days, allowing only 3 infections, restricting its dispersion and reducing the chance of detection outside of its intended targets [30].

Table 2.11 below describes other notable features of Stuxnet:

Table 2.11: Other Stuxnet Features

Feature	**Description**
Command and Control (C&C) Server Communication	Whenever Stuxnet compromised a new system, it would attempt to report back to C&C servers at http://www.mypremierfutbol.com. The server was used for data exfiltration, upgrades, and the execution of arbitrary commands. In August 2010, the Iranian primary telecom provider blocked communications to the domain, severing the C&C communication [24].

Peer-to-Peer (P2P) Communication	Stuxnet communicated among peers via a RPC client/server architecture. The Symantec technical analysis on Stuxnet concludes that the primary objective of the P2P communication was to allow systems which did not have external connectivity to receive updates and commands from other peers which would have external connectivity [30][24].

Table 2.12 below describes Stuxnet Zero-Day Exploits:

Table 2.12: Stuxnet Zero-Day Exploits (5)

CVE Number	**Description**
CVE-2010-2568	Windows Shortcut 'LNK/PIF' Files Automatic File Execution
CVE-2010-2729	Windows Print Spooler Service Remote Code Execution
CVE-2010-2743	Windows Kernel Win32K.sys Keyboard Layout Privilege Escalation
CVE-2010-2772	Siemens Simatic WinCC Default Password Security Bypass
CVE-2010-3888	Windows Task Scheduler Privilege Escalation

2.3.2 Duqu

Analysis of Duqu revealed that Stuxnet and Duqu shared not only binaries, but source code. Thus, the the security community at-large has deduced that both malware originated from either the same source, or affiliated sources. As Duqu appears to have compromised similar targets as Stuxnet, it is suspected that both malware are both part of greater effort. While in abstract, Duqu is a simple information-stealing malware; it is a notable early example of several features: operating in system memory to avoid leaving remnants on the compromised

system, employing a peer-to-peer system to both spread and exfiltrate traffic through the network, and using a self-destruct mechanism to remove itself from compromised systems. Many of these techniques have become more broadly utilized [32][33].

Table 2.13: Duqu Malware Attributes

Feature	**Description**
Alternative names	Stars virus
Discovery date	October 14, 2011 by the Laboratory of Cryptography and System Security (CrySys) at Budapest University of Technology and Economics. [27]
Suspected Threat Agent	United States-government entities or affiliated organizations
Components Exploited	Operating System Kernel: Duqu uses a specially-crafted Microsoft Word document which executes a 0-day Windows kernel exploit.

Table 2.14 below describes the payload feature of Duqu:

Table 2.14: Duqu Payload Features

Feature	**Description**
Keylogger	Duqu recorded keystrokes (primarily related to account access) and relayed them to the C&C server.
Infostealer	Duqu employed several variants of infostealers. Most commonly, exfiltrated information included network information (such as interfaces, routing tables, list of network shares, etc), system information, account information, and screenshots.
Remote Access Trojan (RAT)	The Duqu malware installs a trojan which allowed the malware operator to invoke root-level commands on compromised systems.

Table 2.15 below describes the propagation methods of Duqu:

Table 2.15: Duqu Propagation Methods

Method	**Description**
Office Documents	Duqu uses a specially-crafted Microsoft Word document which executes shellcode containing a 0-day Windows kernel exploit.
Shared Network Resources	Duqu would replicate through network shares to additional computers on the network.

Table 2.16 below describes stealth mechanisms of Duqu:

Table 2.16: Duqu Stealth Mechanisms

Feature	**Description**
Antivirus Tampering	Duqu specifically checks for Kaspersky, McAfee, AntiVir, Bitdefender, Etrust, Symantex, ESET, Trend and Rising antivirus products and injects itself into their memory processes to interfere with them
Automated Self-Destruct	Duqu will delete itself by default after 30 to 36 days, depending on the variant. However, additional communication form the C&C servers can extend the lifetime of Duqu on target machines.
Compromised Certificates	In a manner very similar to Stuxnet, Duqu employed the use of compromised signed certificates from JMicron Technology and Realtek Semiconductor to bypass default restrictions on unknown drivers. Duqu also employed drivers digitally signed by Verisign [34].
Compromised Third Party	Credentials stolen from JMicron Technology and Realtek Semiconductor were likely employed to generate certificates employed by Duqu.
Crypter	The configuration file uses AES-CBC encryption.
Fileless	Potentially Duqu's most notable feature, the APT malware operates almost solely in the system memory, minimizing its footprint upon the compromised system. At the time of discovery, relatively few malware employed this technique [32][33].

Table 2.17 below describes other notable features of Duqu:

Table 2.17: Other Duqu Features

Feature	**Description**
Command and Control (C&C) Server Communication	Duqu exfiltrated information attached as to dummy JPEG files to C&C servers located at specific IP addresses over HTTP or HTTPS. The C&C servers at those addresses were distributed across nations, including India, Vietnam and Belgium. The C&C servers redirected traffic to other servers likely to prevent identification of the threat actor. JPEG and HTTP/S are common files and protocols which were presumably employed to obfuscate the communications.
Peer-to-Peer (P2P) Communication	After establishing a foothold within the target network, Duqu spread primarily through a peer-to-peer mechanism. Subsequent infections would update a configuration file which contained a path from the initial point of compromise. The malware would employ this path to proxy traffic back through the network to the C&C Server.
Modular/Plugin Architecture	Duqu employed several variants of infostealer and other modules customized for the individual compromised system.

Table 2.18 below describes the Duqu Zero-Day Exploit:

Table 2.18: Duqu Zero-Day Exploit (1)

CVE Number	**Description**
CVE-2011-3402	TrueType Font Parsing Vulnerability

2.3.3 Sednit

Sednit [35][36] is primarily a data exfiltration APT. It is suspected that the APT has ties to Russian intelligence as the malware has been utilized against organizations with Russian interest including the Ukraine during the 2016 border disputes involving Crimea. This group has also been linked to compromise of the American Democratic National Committee prior to and during the 2016 American presidential election campaign. American intelligence agencies have concluded that Russia operatives have been sought to influence the American political process though the use of Sednit and other means.

Table 2.19: Sednit Attributes

Attribute	**Description**
Alternative Names	Operation Pawn Storm, Sofacy, STRONTIUM, Tsar Team
Discovery Date	October 2014
Suspected Objective	Military/Defense and Political Espionage. Notable targets have been the American Democratic National Committee, the German parliament and the French television network TV5Monde
Suspected Threat Agent	Russian-based intelligence organizations

Component(s) Exploited	Sednit primarily spreads through phishing emails with compromised attachments of Microsoft Word and Excel. Some of Sednit's early versions included compromised versions of Adobe Reader which would deploy the malware upon opening the compromised file.

Table 2.20 below describes the payload feature of Sednit:

Table 2.20: Sednit Payload Feature

Feature	**Description**
Backdoor	The Sednit toolkit includes two backdoor malware tools, Sedreco and Xagent. XAgent is often an initial data-gathering tool, and Sedreco is used for more continuous, long-term monitoring.

Table 2.21 below describes propagation methods of Sednit:

Table 2.21: Sednit Propagation Methods

Method	**Description**
Phishing	By far the most common method used to insert Sednit into a network is to use spear phishing/whaling techniques. This was true in the case of the compromise of the American Democratic National Committee.

Watering Hole	Several emails directed users to click on links which lead to carefully-crafted websites

Table 2.22 below describes stealth mechanisms of Sednit:

Table 2.22: Sednit Stealth Mechanisms

Mechanism	**Description**
Crypter	Various Sednit tools employ encryption. For example, 3DES encryption is used in the Sedreco tool in its outbound file format.
Peer-to-Peer Communication	Sednit incorporates a dedicated tool, known as Xtunnel to which converts compromised systems into network proxies which can allow the malware operator to access portions of the network which are otherwise inaccessible.
Rootkit	The Downdelph tool is rootkit which intercepts operating system commands and disguises malware activity.

Table 2.23 below describes other notable features of Sednit:

Table 2.23: Other Notable Sednit Features

Feature	**Description**
Bootkit	As a part of the Sednit collection of tools, the bootkit/rootkit known as Downdelph is included. As a bootkit, the system master boot record is compromised and survives system reboots.
Command & Control Communication	As a primarily espionage-focused APT, communication with external servers is critical to the success of Sednit.

Table 2.24 below describes Sednit zero-day exploits:

Table 2.24: Sednit Zero-Day Exploits (5)

CVE Number	**Description**
CVE-2016-7855	Use-after-free vulnerability in Adobe Flash Player
CVE-2015-3043	Flash
CVE-2015-1701	Windows LPE
CVE-2015-2590	Java
CVE-2015-4902	Java click-to-play bypass
CVE-2015-2424	Office RCE
CVE-2015-7645	Flash

2.4 Adaptive Cyber Defense and Moving Target Defense

Moving Target Defense was first introduced in a series of papers that modeled a system's security as a function of its exposed attack surface and showed how MTDs increased diversity based on software and network transformations [37].

Since its introduction, a myriad of Adaptive Cyber Defense and MTD techniques have been developed in the literature, each targeting different aspects of a system. They have been generally organized by type according to a taxonomy published by Lincoln Labs [38][39] into the following categories:

- **Dynamic Runtime Environments**
- **Dynamic Platforms**
- **Dynamic Software**
- **Dynamic Data**
- **Dynamic Networks**

However, it has been clear that in recent years MTD techniques have flourished as evidenced by the addition of new surveys by Lei [40], Zheng[41], Sengupta [42] and Cho [43].

Moving Target Defense was first introduced in a series of papers that modeled a system's security as a function of its exposed attack surface and showed how MTDs increased diversity based on software and network transformations [37]. Later papers expanded on this concept, incorporating aspects of game theory, where an attacker or defender may adopt different strategies based on the actions of the other [44] or introduce machine learning into MTD behavior [45].

Although the MTD taxonomy described covers most MTDs as they apply to conventional computer systems, MTD techniques have also been applied on several other platforms that don't fall neatly into those categories. For example, MTDs have been studied in resource-constrained environments such as tactical network devices or FPGAs [46], cyber-physical systems [47], and wireless sensor networks [48][49].

2.4.1 Dynamic Runtime Environments

Dynamic Runtime Environments operate by dynamically changing the environment presented to an application. Generally implemented at a low level fairly close to hardware,

there are two major categories: Address Space Layout Randomization (ASLR) and Instruction Set Randomization (ISR). ASLR is one of most mature and widely-adopted forms of MTD in conventional usage and protects against buffer overflow attacks by randomizing key locations of memory [50]. Since first being introduced, many improvements have been proposed, such as changing the focus of the MTD from preventing invalid memory accesses to offering unpredictable results [51] or by randomizing instructions while in operation to improve entropy [52]. Another technique that incorporates aspects of address randomization in its protection is DieHard [53] [54], which also protects against heap buffer overflows by increasing space between elements and maintaining multiple replicas of the heap and using voting to prevent subversion of control.

ISR works to mitigate Return-Oriented Programming (ROP) and code injection attacks. These attacks are generally not protected by the use of ASLR [55]. ISR operates by ensuring injected code is not immediately compatible with the target, often by performing simple encryption or adding some additional required label to each opcode. This can be done at compile time [56], or performed at run-time in an emulator [57][58]. Is it noted that ISR techniques can often be used in conjunction with ASLR techniques to supplement each other [38].

2.4.2 Dynamic Platforms

MTDs which fall under the classification of Dynamic Platform operate at a slightly higher level of abstraction than Dynamic Runtime Environments by dynamically changing platforms such as OS version, OS instance, or CPU architecture. To implement these techniques, various forms of virtualization is commonly employed. One method would be to operate by rotating between multiple variants of the Linux OS [59], or by designating roles for each VM and shuffling them between hosts [60].

Dynamic platforms could also be implemented using multivariant systems, where multiple variations of an OS are run at the same time and monitored for divergence divergence [61]. While potentially introducing significant operational overhead, a malicious attacker attempting to divert control would only do so one one of the variants, which would then be detected. Subsequently, the afflicted variant would be reverted to a known good state.

Making OS changes on a regular interval can be disruptive to running applications but an MTD can accomplish this by first taking a snapshot of the current state, execution state, open files, and network sockets [62]. Other MTDs use similar methods of snapshotting system images and replacing them with known good copies if tampering is detected or to disrupt attacker's persistence on a system [63][64][65].

2.4.3 Dynamic Software

MTDs classified as Dynamic Software often operate in a similar manner as Dynamic Platforms; but operating instead at the application level as opposed to the OS level. The grouping, order, format, or the actual instructions within an application's code can be changed dynamically. This includes Multivariant approaches that run several different versions of software to prevent all machines being compromised by the same exploit [66]. A simpler implementation of this approach uses a single replica compiled with the stack working in the opposite direction so that an exploit cannot work on both [67]. Another Dynamic Software Method would to be implement some sort of shuffling or rotation between software that is currently being executed [68], which can potentially cause operational concerns and often requires coordination for practical implementation. One software approach would be to generalize DieHard for individual applications as opposed to OS use [69].

2.4.4 Dynamic Data

Dynamic Data MTDs are implemented primarily via some form of continuous transformation to the format, syntax, encoding, or representation of an application's data. For

example, alterations to common protocols, such as manipulating HTML tags from a web server to thwart bots (but allowing legitimate users to render them correctly) [70] or adding additional keywords to SQL commands and table names which are required to run the commands may prevent SQL Injection attacks [71].

2.4.5 Dynamic Networks

Dynamic Networks involves changing network addresses or other network properties dynamically. Dynamic Networks are one of the most widely studied areas of Moving Target Defense, as many cyber attackers leverage computer networks as an attack vector and network MTDs. Furthermore this form of MTD can be implemented at a level of abstraction above individual systems or applications, in theory making the implementation of the defense transparent to the application itself. This form of protection is desirable to prevent an attacker from accessing the system altogether.

Perhaps the earliest and most oft-cited example of a Dynamic Network MTD would be the various IP hopping schemes [72][73]. For example, a scheme could include decoy nodes and shuffle them regularly along with actual nodes to further delay attackers [74]. Instead of changing the target system IP addresses directly, an MTD can implemented instead by a series of rotating proxies which act an intermediaries to the defended system [75]. These intermediary proxies will transparently pass traffic, but can shift IP addresses with more ease than the defended application.

An improvement on the IP-hopping scheme is Random Host Mutation [76][77] which is implemented at a centralized DNS server and maps ephemeral IP addresses (eIP) to real IP addresses rIP). This technique randomizes host-to IP bindings based on source identity and time [78] and is able to maintain connection states. The technique also has the ability to adapt to an attacker probes by moving hosts to addresses with a lower probability of being scanned or moving nodes to addresses that have already been scanned [79].

Instead of centralizing operation of the MTD, it is possible to implement it across an entire network by using a hypervisor to rewrite packets at each node to make each network hop

dynamic. The Self-Shielding Dynamic Network (SDNA) protocol also allows for encryption, authentication, and redirection to a honeypot for unauthenticated users [80][81][82].

Besides actually changing IP addresses, a network MTD can also take other actions to virtually affect the network and disrupt attackers. For example, an MTD might only manipulate an attacker's view of the network, using some sort of protocol scrubber [83] or the dynamic defense could come in the form of lightweight sensors that are able to move around the network and swarm around any areas where there are potential discrepancies [84].

It is worth noting that network MTDs also take advantage of evolving technology. IPv6 offers a vastly larger address space as opposed to IPv4 and therefore greater entropy to techniques that use it. MT6D uses the IPv6 address space to create an encrypted tunnel that uses a range of addresses and ensures protection as well as privacy [85][86]. This technique is also applicable to embedded systems on the smart grid using IPv6 [87] or as part of a hybrid approach with a mix of static and dynamic IP addresses [88] to protect mobile-enabled systems.

Chapter 3: Preliminary Definitions and Problem Statement

This chapter defines incentives and deterrents that APT actors may weigh when evaluating which nodes to target and how to move within a network. This system of incentives and deterrents is at the basis of the proposed quantitative framework.

3.1 Assumptions

The value an APT actor gains from a target network during a given time interval depends on the number and nature of the compromised nodes. The malware footprint within the network is modeled as an overlay tree, rooted at the attacker's entry point into the network. The choice of using a tree overlay is justified by the following reasoning. As mentioned previously, APT actors are highly determined to ensure that their actions are stealthy in order to maintain persistence and accrue more value over time. To achieve this goal, they seek to avoid detection by minimizing the number of communication channels, as observed in several instances of existing malware [34, 89]. This behavior can be captured by modeling communication links as the edges of an overlay tree. However, there are other means an attacker can use to minimize detectability. Jafarian *et al.* introduce the notion of *quantifying detectability* by measuring malware activity during a given time window [15]. Leveraging this concept, it is assumed that the need to maintain stealth constrains attackers to minimize detectability. The constraint is modeled as a *detectability budget* B, representing the attacker's level of risk tolerance within any time interval Δt_i.

The network is represented as a graph $G = (V, E)$, where V is a set of network elements – such as routers and end hosts – and E captures the connectivity between them. As mentioned earlier, APT actors continue to accrue value as they persist within the target network. To capture the temporal dynamics of APTs, and without loss of generality, I

discretize time as a a finite sequence of integers $\mathcal{T} = \langle t_0, t_1, \ldots, t_m \rangle \subseteq \mathbb{N}^m$, with $m \in \mathbb{N}$ and $t_i < t_{i+1}$ for each $i \in [1, m]$. I then define $reward(t_i)$, with $i \in [1, m]$, as the value accrued by an APT actor at time t_i, or, more precisely, during the time interval $\Delta t_i = [t_{i-1}, t_i]$. Therefore, the value accrued by the attacker over the entire time horizon $\mathcal{T}$ can be simply defined as $Reward = \sum_{k=1}^{m} reward(t_i)$.

Due to the sophisticated nature of APTs, the amount of resources that threat actors invest in researching their targets, and the potential exploitation of zero-day vulnerabilities, it is reasonable to assume that APTs are always successful in compromising a target node. Additionally, a strong adversarial model is assumed in which attackers have full knowledge of the network topology, the location of various assets – including data items – and their respective value. APTs are known for being able to bring to bear considerable resources, including the use of traditional intelligence and covert operations. In the case of Stuxnet, as mentioned in Chapter 2, the APT actor possessed specific intelligence regarding the Iranian industrial control devices. This knowledge allowed the actor to specifically target surreptitiously target and sabotage the specific device.

3.2 Cost

Attacker interactions with network nodes incur a *detectability* cost, which quantifies the risk of those interactions being detected. As mentioned earlier, the cost an attacker is willing to incur during any time interval Δt_i is bounded by B. Detectability costs for individual nodes can be determined based on several characteristics of those nodes, as described below.

- **Role:** Intuitively, nodes with more mission-critical roles, such as a database server, would undergo more scrutiny and would be more heavily monitored than typical user workstations. For example, staff may routinely check the status of a critical database, whereas a user workstation may only be examined when there is an apparent issue.

- **Operating System:** Newer operating systems inherently incorporate additional protection mechanisms which render attacks designed for older systems ineffective. Furthermore, security monitoring options are typically more robust and diversified on these operating systems, resulting in more effective detection of malicious activity. As a result, newer operating systems generally force attackers to resort to *noisier* attacks, which defenders can more readily detect.

- **Deployed Services and Applications:** Services and applications inherently expose an attack surface which attackers may employ to exploit the overall system. This is particularly the case for services and applications which require network connectivity to operate. Common examples may include web, mail and database servers.

- **Deployed Defense Mechanisms:** Defenders often install additional defense mechanisms to provide another layer of defense on top of baseline operating system features. These may include host-based antivirus software, host-bases intrusion detection systems, file integrity monitoring systems, and other similar defenses.

The detectability cost of a node is also affected by the level of attacker's effort necessary to compromise the node and maintain it in the malware footprint. The attacker's effort includes activities that can be broadly classified in the following two categories:

- **Compromise and acquisition**: activities performed to establish the attacker's presence and control on the target node, including activities such as scanning and reconnaissance, initial compromise, and privilege escalation.

- **Operation-on-target and maintenance**: activities performed to carry out the attacker's objectives on a compromised node, including activities such as exfiltrating information, downloading malware updates, and routing attack traffic.

Another important consideration is that the malware footprint within a network may not be composed solely of compromised nodes. An attacker may leverage standard network mechanisms to forward malicious traffic through several nodes – such as routers – without

necessarily compromise them. If such nodes are on the path between two compromised nodes, they can be considered part of the malware footprint. Compromising every node within the malware footprint is impractical for a variety of reasons. Assuming that the attacker has the capability to do so, leveraging such capability may not be cost-effective. For instance, compromising a router may give the attacker additional rewards, but the risk of detection would also be much higher. In this case, it may be more cost-effective for the attacker to compromise nodes hosting sensitive data and use the router to forward exfiltration traffic originating at the compromised nodes. Intuitively, including non-compromised nodes in the malware footprint enables APT actors to more efficiently allocate their detectability budget and gain more rewards from the target network by using the budget to compromise more desirable and cost-effective nodes.

To account for this type of scenario, two types of nodes are consider in the overlay model, namely *compromised nodes* and *traffic-forwarding nodes*. The latter are nodes which pass malicious traffic, but are not compromised by the attacker. Since these nodes are used only passively to forward traffic, the attacker incurs a lower cost to maintain them in the malware footprint. However, there is still a small risk associated with these nodes, as malicious traffic passing through them can be potentially detected, thus revealing the presence of the attacker. Thus, the notion of *APT Malware Footprint* is defined as follows.

Definition 1 (APT Malware Footprint)**.** *The footprint of an APT malware in a network* $G = (V, E)$ *is a tree* $T = (C \cap F, root)$*, where*

- $C \subseteq V$ *is a set of compromised nodes*
- $F \subseteq V$ *is a set of non-compromised non-leaf traffic-forwarding nodes*
- $root \in C$ *is the root of the tree.*

Detectability is modeled as two separate cost functions, namely $cost_c : V \times \mathcal{T} \to \mathbb{R}$ and $cost_f : V \times \mathcal{T} \to \mathbb{R}$. Specifically, $cost_c(v, t_i)$ is the detectability cost incurred by the APT malware for compromising node v and maintaining the compromise at time t_i. Similarly, $cost_f(v, t_i)$ is the detectability cost incurred by the APT malware for forwarding traffic

through node v at time t_i. As discussed previously, the detectability cost of compromising a node is higher than the cost of forwarding traffic through the same node, therefore $\forall v \in V, \forall t_i \in \mathcal{T}, cost_c(v,t_i) \geq cost_f(v,t_i)$. Additionally, $\forall v \in V$, $cost_c(v,t_0) = 0$ and $cost_f(v,t_0) = 0$ by definition. The total cost at time t_i can be computed as

$$cost(t_i) = \sum_{v \in C} cost_c(v,t_i) + \sum_{v \in F} cost_f(v,t_i), \forall i \in [1,m]$$

It must be noted that, once a node v has been compromised at time t_j, the cost for the attacker to maintain v in its footprint during subsequent time intervals is lower than the cost sustained for the initial compromise. Formally, the cost function $cost_c$ for a node v compromised at time t_j can be defined as

$$cost_c(v,t_i) \begin{cases} = 0 & \text{if } t_i < t_j \\ = c_v & \text{if } t_i = t_j \\ < c_v & \text{if } t_i > t_j \end{cases} \tag{3.1}$$

where c_v is a constant representing the one-time cost sustained by the attacker to compromise node v.

3.3 Reward

APT malware gains value from information extracted from target systems. Many prior approaches use the concept of one or more *target nodes* [90, 91], which an attacker seeks to compromise. In these approaches, target nodes may represent *crown jewels*, such as critical databases or email servers, which would severely impact an organization, if compromised. While these targets are undoubtedly critical, these approaches do not account for other valuable information which may reside on other nodes within the network, nor for the intrinsic value those nodes may have for an attacker who seeks to maintain persistence

within the network.

An attacker may choose to forgo these targets entirely in favor of more lightly-defended options. This approach may be especially enticing if the network defenses are densely concentrated around critical resources and relatively sparse in other regions of the network. In such a situation, an attacker can compromise a larger proportion of the network without exceeding its detectability budget. Accruing value in a more incremental fashion may in fact be a more effective strategy for an attacker aiming to evade detection. For example, an attacker may forgo an organization's heavily-guarded email server and instead compromise the organization's lightly-secured user workstations. Depending on the organizational security policy, the attacker may be able to extract not only email messages that could have been otherwise retrieved from the server at a much higher risk, but also additional locally-stored information.

Two separate *reward* functions are used to model the value of nodes throughout the network, namely $reward_c : V \times \mathcal{T} \rightarrow \mathbb{R}$ and $reward_f : V \times \mathcal{T} \rightarrow \mathbb{R}$. Specifically, $reward_c(v, t_i)$ is the reward gained by the APT malware for compromising node v or controlling the compromised node at time t_i. Similarly, $reward_f(v, t_i)$ is the reward gained by the APT malware for forwarding traffic through node v at time t_i. As discussed previously, the reward gained from a compromised node is higher than the reward from a non-compromised node used solely for forwarding traffic, therefore $\forall v \in V, \forall t_i \in \mathcal{T}, reward_c(v, t_i) \geq reward_f(v, t_i)$. Additionally, $\forall v \in V$, $reward_c(v, t_0) = 0$ and $reward_f(v, t_0) = 0$ by definition. Finally, the total reward at time t_i, with $i \in [1, m]$, which was introduced earlier in Section 3, can be computed as

$$reward(t_i) = \sum_{v \in C} reward_c(v, t_i) + \sum_{v \in F} reward_f(v, t_i)$$

3.4 Tree Formation Problem

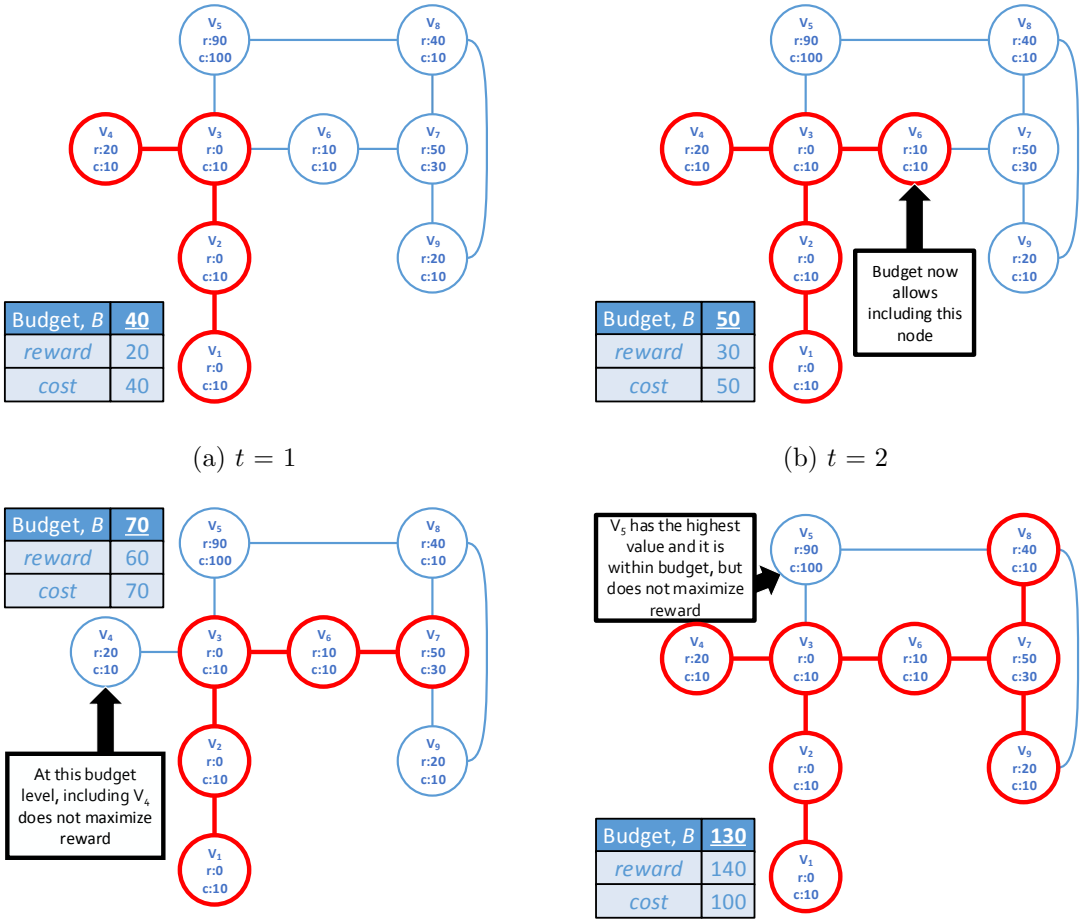
The long-term attacker's objective is to construct and overlay tree $T = (C \cap F)$ that maximizes total rewards over the time horizon $\mathcal{T} = \langle t_0, t_1, \ldots, t_m \rangle$, subject to cost constraints.

$$\begin{aligned} \underset{T}{\text{maximize}} \quad & \sum_{i=1}^{m} reward(t_i) \\ \text{subject to} \quad & cost(t_i) \leq B, \forall i \in [1, m] \end{aligned}$$

However, as the above optimization problem would be unpractical for the attacker to solve, it is reasonable to assume that a more realistic objective is to construct, during each time interval Δt_i, a partial tree $T_i = (C_i \cap F_i)$ that reuses – either partially or totally – the tree T_{i-1} generated during the previous time interval and maximizes the rewards, subject to the detectability budget B.

$$\begin{aligned} \underset{T_i}{\text{maximize}} \quad & \sum_{v \in C_i} reward_c(v, t_i) + \sum_{v \in F_i} reward_f(v, t_i) \\ \text{subject to} \quad & \sum_{v \in C_i} cost_c(v, t_i) + \sum_{v \in F_i} cost_f(v, t_i) \leq B \end{aligned}$$

As formulated, this problem is closely related to the class of Steiner Tree Problems [92, 93]. In particular, this problem closely models the Prize-Collecting Steiner Tree Problem (PCST) and, more specifically, the Node-Weighted Prize-Collecting Steiner-Tree Problem (NW-PCST), for which a number of approximation algorithms exist [94,95]. These approximation algorithms include rooted variations, where a specific root node is required to exist in the tree. These variations would accurately model an entry or extraction point for the malware. However, all variants of this problem are generally considered NP-Hard, and the budgeted variants are shown to be at least as hard to approximate as the maximum coverage problem [96].



(a) $t = 1$ (b) $t = 2$

(c) $t = 3$ (d) $t = 4$

Figure 3.1: Basic example of APT malware footprint for different values of the budget

Example 1 (APT Malware Footprint)**.** *Figure 3.1 illustrates the concept of APT Malware Footprint using a simple network and different values, ranging from 40 to 130, for the attacker's detectability budget. In this example, it is assumed that all nodes in the malware footprint, denoted with a bold red outline, are compromised. In these figures, which capture a snapshot of the network at time t_1, each node v is labeled with its respective reward $r = reward_c(v, t_1)$ and cost $c = cost_c(v, t_1)$. In this example, V_1 is the root node, that is the entry/exit point for the attacker. Note that this model intuitively reflects that an increased detectability budget allows the attacker to access a larger proportion of the target network*

and accrue a larger reward. Also note that, in Figures 3.1a and 3.1b, the attacker selects V_4 as the budget does not allow compromising other more rewarding nodes. However, as shown in Figure 3.1c, with a larger budget, the attacker chooses to forgo $V4$ in favor of V_7, which allows more efficient use of the available budget and leads to a larger overall reward. In Figure 3.1d, the attacker budget is so large that nearly the entire network is compromised. However, it should be noted that the one node which is not compromised, V_5, has the single highest value in the entire graph. However, due to also having the highest associated cost – indicating the presence of more sophisticated defense mechanisms – including this node would not lead to the highest possible aggregate reward for this budget level. Other threat modeling approaches focusing on target nodes would likely designate V_5 as a primary target for the attacker, thus steering even more defensive resources towards it. This example demonstrates that an attacker does not necessarily need to compromise the node with the highest value within the network in order to maximize the reward.

3.5 Summary

In this chapter, a novel model for organizational networks is detailed as well as its real-world underpinnings. In contrast to prior work, this model accounts for the reward APTs gain from compromising any arbitrary node within the organization's network - though at the cost of potentially revealing their presence. Given this model, this chapter also presents the attacker problem of building a footprint to maximize the reward from the network; and implicitly, the defender's problem of minimizing the attacker's reward. The next chapter addresses both problems.

Chapter 4: Problem Solution

In the previous chapter, a model was introduced which allows attackers to quantify reward gained from organizational networks. Attackers seek to maximize this reward gained over time, whereas defenders seek to minimize it. This chapter describes practical approaches to addressing both problems and evaluations of both.

4.1 Greedy Algorithm

In this section, an algorithm is described to model how an APT actor may build its footprint in the target network. To ensure practical runtimes and to scale to large networks, an iterative, greedy approach is proposed. At a conceptual level, the algorithm starts from a root node, which represents the attacker's entry point into the network, and, during each time interval Δt_i, the attacker enumerates routes to potential target nodes originating from any of the current leaf nodes. Each route includes a target node, which the attacker seeks to compromise, and possibly a number of intermediate $forwarding$ nodes. Note that, for each such routes to be added to the malware footprint, the route's terminal node must be compromised for the attacker to be able to control that route. The length of the routes is controlled by the parameter hop_{max}. Each route has an associate reward and cost, which correspond to the aggregated reward and cost of all nodes that comprise the route.

When the attacker selects a route, the route is added to the malware footprint and the attacker may enumerate new routes originating from any of the currently compromised nodes. The attacker selects routes in this iterative manner until the the budget B is exhausted or there are no more viable routes to consider. Different strategies may be employed by the attacker to select a route amongst the available ones. Three possible *route selection strategies* are proposed: (i) Greatest_Reward, which selects the route that maximizes

reward; (ii) Lowest_Cost, which chooses the route that minimizes cost; (iii) Cost_Effective, which chooses the route that optimizes the cost/reward ratio.

Two different tree evolution approaches are considered. In the *conservative* approach, the attacker maintains any node that was compromised during previous time intervals. In this case, a fraction of the budget B for the current time interval goes toward maintaining the compromised nodes. As discussed earlier, the maintenance cost is lower than the cost of compromising, so the remaining portion of the budget can be used to expand the tree. In the *dynamic* strategy, the attacker may forgo maintaining some of the previously compromised nodes and have a larger portion of the budget B available to compromise new nodes if this choice leads to better values of the chosen objective function.

The pseudo-code of the proposed *treeConstruction* algorithm is shown in Algorithm 1, and the pseudo-code of the $findRoutes$ algorithm is shown in Algorithm 2.

Algorithm 1 $treeConstruction(G, T, B, hop_{max})$

Require: Graph $G = (V, E)$, current tree $T = (C \cap F, root)$, budget B, and parameter hop_{max}.
Ensure: An updated tree T.
▷ Initialize
$R \leftarrow \emptyset$
$runningBudget \leftarrow 0$
$r \leftarrow \emptyset$
▷ Bootstrap R with root
findRoutes(G, T, r, root, R, runningBudget, 0, B, hop_{max})
while $R \neq \emptyset$ **do**
▷ selectRoute selects route depending on route selection strategy, e.g. Greatest_Reward
    $r \leftarrow selectRoute(R)$
▷ final destination node is compromised, other nodes in the route are traffic-forwarding nodes
    $\mathcal{T} \leftarrow r$
    for all $routes \in R$ **do**
▷ manage R:
▷ remove invalid routes (routes with the final destination not compromised)
▷ update costs and rewards for remaining routes which share nodes with r
    end for
▷ Find routes using the newly compromised node
    findRoutes($G, T, finalDest, r, R, B, runningBudget, rCost, rReward, hop_{max}$)
end whilereturn T

Example 2 (Malware Footprint Evolution)**.** *To illustrate how the malware footprint evolves over time, consider again the simple example discussed earlier. In this example, the budget remains constant through time ($B = 55$ for each time interval). However, as shown in Figure 4.1, during each time interval, the attacker compromises a larger proportion of the network due to cost of maintaining compromised nodes being lower than the cost of*

Algorithm 2 $findRoutes(G, T, v, r, R, B, runningBudget, rCost, rReward, hop_{max})$

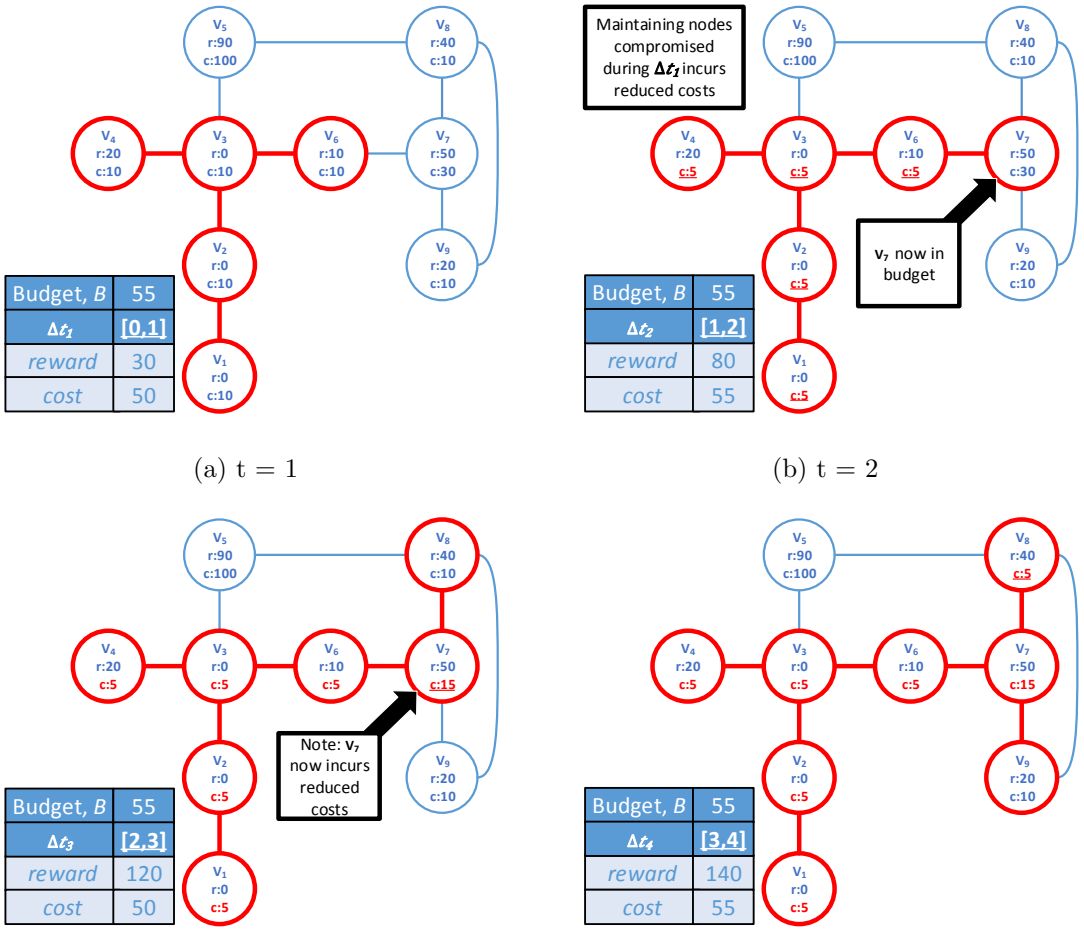
Require: Graph $G = (V, E)$, current tree $T = (C \cap F, root)$, node v, node list r, current set of viable routes R, budget B, $runningBudget$, $rCost$, $rReward$, and parameter hop_{max}
Ensure: The set of viable routes, R, is updated.

▷ Determines if v is suitable for serving as final route destination (compromised node)
$nodeInfectCost \leftarrow$ detectability cost for malware to compromise v
if $v \notin C$ **then**
  if $runningBudget + rCost + nodeInfectCost < B$ **then**
    ▷ Update the r with node data and add to R
    $newRCost \leftarrow rCost + nodeInfectCost$
    $nodeInfectReward \leftarrow$ malware reward from the compromise of v
    $newRReward \leftarrow rReward + nodeInfectReward$
    $newRoute \leftarrow$ deep copy of r, v
    $newRouteOption \xleftarrow{+} newRoutw, newRCost, newRReward$
    $R \xleftarrow{+} newRouteOption$
  end if
end if
//Determines if v could be a forwarding node
$rLen \leftarrow$ length of r
if $rLen < hop_{max}$ **then**
  $neighbors \leftarrow$ neighbors of n
  for all $neighbor \in neighbors$ **do**
    ▷ check to ensure neighbor not already in r
    if $neighbor \notin routeOptions$ **then**
      $nodeForwardCost \leftarrow$ detectability cost for malware to forward traffic though v
      if $runningBudget + routeCost + nodeForwardCost < B$ **then**
        $newRouteCost \leftarrow routeCost + nodeForwardCost$
        $nodeForwardReward \leftarrow$ any malware reward when forwarding traffic through v
        $newRouteReward \leftarrow routeReward + nodeForwardReward$
        $newRouteList \leftarrow$ deep copy of $routeList, v$
        findRoutes $(G, T, neighbor, newRouteList, routeOptions, B, runningBudget,$
        $newRouteCost, newRouteReward, hop_{max})$
      end if
    end if
  end for
end if

the original compromise. In this example, node detectability costs are reduced by one-half in the time interval after initial compromise. By incorporating the temporal dynamics of detectability cost, the threat model reflects the progressive nature of APT malware behavior, slowly expanding through the network to avoid detection.

4.2 Simulations

In this section, performance of the algorithm for large networks is presented. For each network size, 30 different network topologies are generated and the results were averaged over different network settings. There is a lack of existing models that capture the connectivity



(a) t = 1 (b) t = 2

(c) t = 3 (d) t = 4

Figure 4.1: Malware footprint evolution

of an enterprise network at both layer 2 and layer 3. Therefore, in order to generate different network topologies, scale-free networks were used and synthesized enterprise network topologies of different sizes. Such networks are known to accurately capture the connectivity in ISP networks at router level [97]. The NetworkX 2.0 library was used to generate these networks in an incremental fashion, using the Holme and Kim algorithm, a variant of Barabási-Albert (BA) model. In the BA model, new nodes are added to the network one at a time and each new node is connected to one of the existing nodes with a probability that is proportional to the current degree of that node. Therefore, a node with a higher

degree has a higher probability of becoming the new node's neighbor. The Holme and Kim algorithm tends to generate networks with more clusters than typical BA networks, which is intended to more closely model enterprise networks.

Rewards. Figure 4.2 reports the total rewards accrued by the attacker over a time horizon $\mathcal{T} = \{t_1, \ldots, t_m\}$, with $m = 10$, and for all three route selection strategies, when using the dynamic approach. As expected, the malware gains more rewards from larger networks, and the cost-effective strategy yields the best results among all the strategies considered.

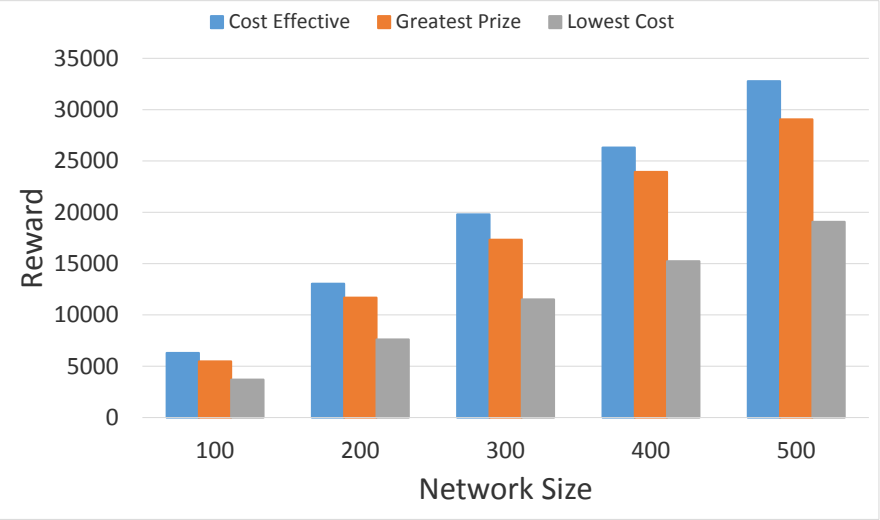


Figure 4.2: Rewards vs. Network Size

Malware Footprint. Figure 4.3 reports the percentage of the total number of nodes that are compromised by the malware for different network sizes at time t_m. The Lowest_Cost strategy achieves the better results because it tends to compromise a large number of low-cost but low-reward nodes, whereas the Greatest_Reward tends to include high-reward, but also high cost, nodes, thus exhausting its budget sooner.

Runtime. Fig. 4.4 illustrates the average runtime for different network sizes. It can be seen that the total runtime increases linearly with the network size. Instead, Figure 4.5 compares the runtime of the dynamic and conservative approaches at different network sizes. The

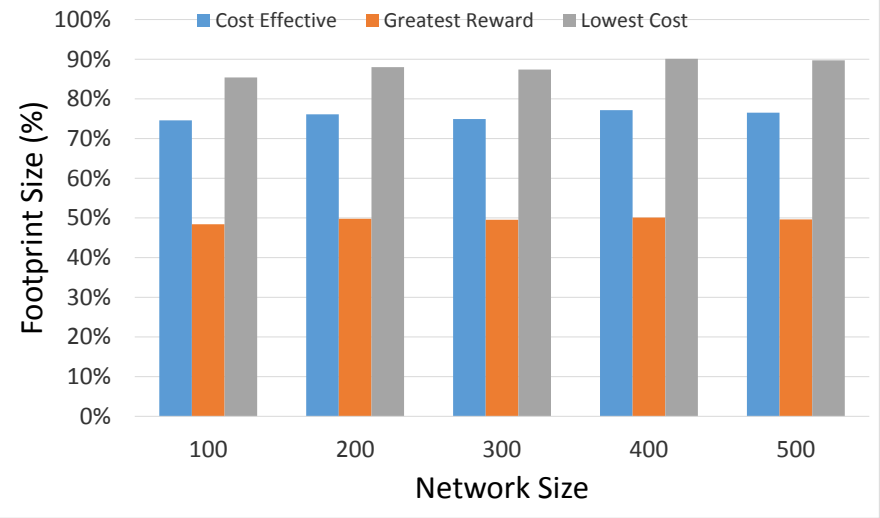


Figure 4.3: Footprint vs. Network Size

dynamic approach requires roughly twice as much time to compute.

Conservative vs. dynamic approach. Figure 4.6 shows the reward accrued by the malware over time as a percentage of the total value of all network assets. The chart compares the dynamic and conservative approaches, and reports values averaged over of all network sizes considered (100,200,300,400,500). During the first time interval, the two approaches yield the same reward, but then they start to diverge. As the dynamic approach may reconsider the choice of target nodes made in previous time intervals, it is likely to achieve results that are closer to optimal.

Approximation Ratio. To evaluate the approximation ratio of the proposed algorithm, the rewards gained over time using the three route selection strategies were compared with the rewards corresponding to the optimal solution, which was computed by exhaustively exploring the search space with a brute-force approach. Figures 4.7a and 4.7b show how the approximation ratio – for the conservative and the dynamic approach respectively – converges to 100% as the route length increases. As expected, the dynamic approach converges more rapidly than the conservative. Results presented here are generated using networks of 25 nodes, as computation of the optimal solution for larger graphs is prohibitive.

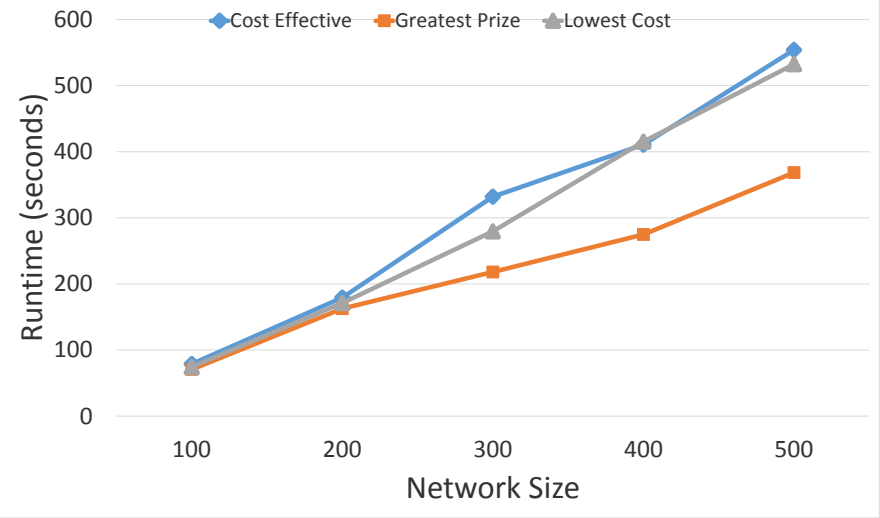


Figure 4.4: Runtime vs. Network Size

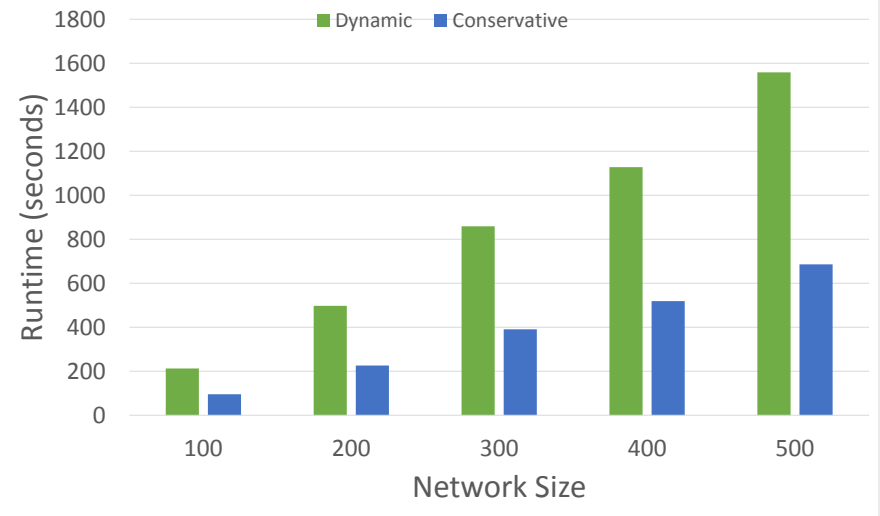


Figure 4.5: Conservative vs. dynamic approach

4.3 Defender Model

The goal of the defender is to minimize the attacker's reward. As the attacker is invested in solving the rooted tree problem discussed above, and success during earlier time intervals

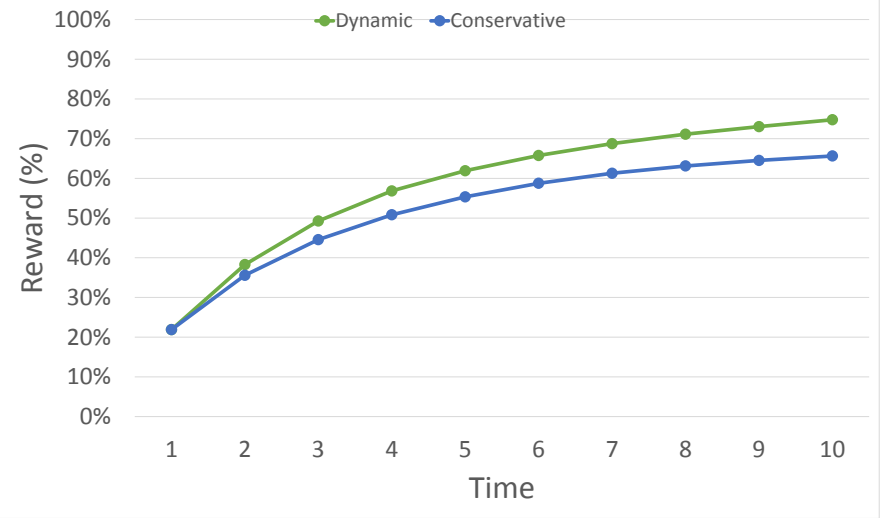
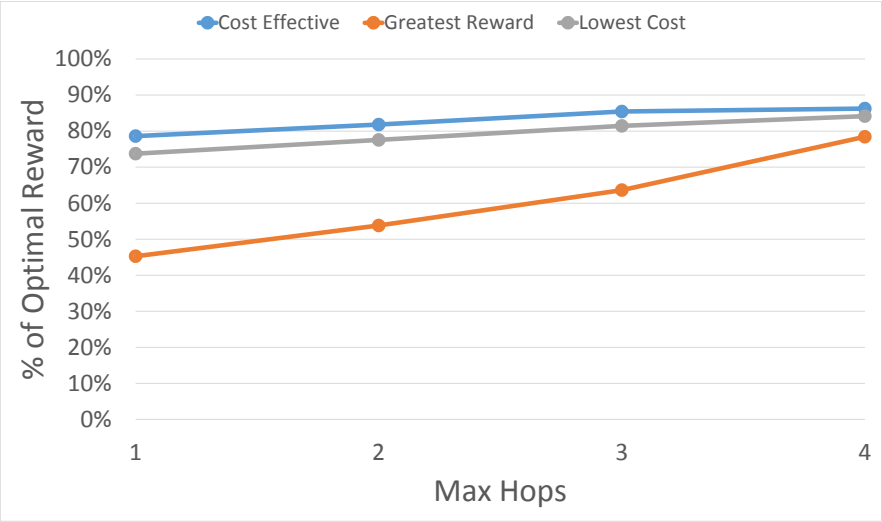


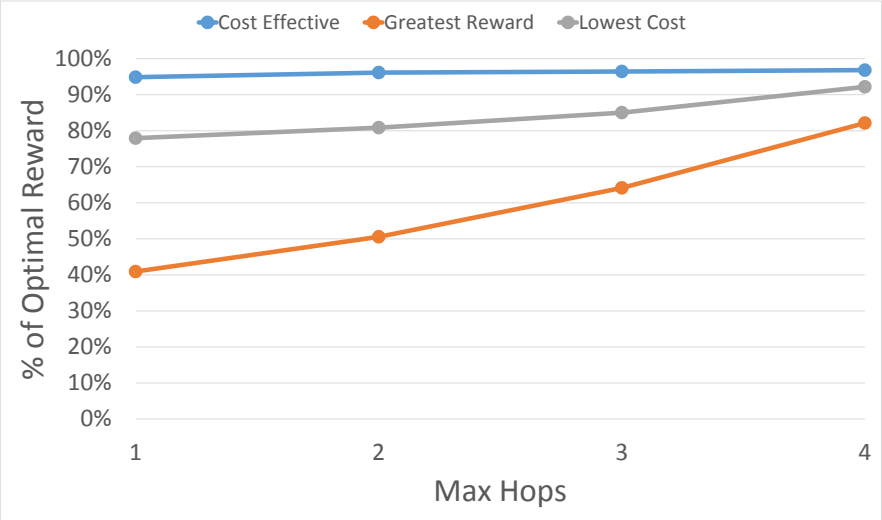
Figure 4.6: Conservative vs. dynamic approach

implies an increased effective budget for the current interval, a maximally effective defender will be one that disrupts the attacker's effort to maintain a tree, thus forcing the attacker to continually compromise new nodes, which results in inefficient use of the detectability budget.

It is assumed that the defender has sufficient resources to actively defend $k < n$ nodes, where $n = |V|$ is the number of network nodes. In real world scenarios, such nodes might receive more frequent security upgrades or be more heavily monitored. The parameter k reflects the organization's capacity for processing alerts and allocating analyst time. Such actively defended nodes are noted as *watched*, and the process by which the defender selects a k-element subset $S \subseteq V$ is designated as the defender's *strategy*. A watched node's cost increases by a multiplicative factor $\alpha > 1$. This increase in cost reflects the additional work and risk an attacker assumes by trying to compromise a node that is subject to patching and increased monitoring. In each interval Δt_i, the defender selects a set $S_i \subseteq V$ according to one of a family of strategies. Let nodes S_{i-1} be the set of watched nodes from the previous interval. For each $v \in S_i \setminus S_{i-1}$ – the newly watched nodes – $cost_c(v, t_i)$ is increased by a factor of α. For each node $v \in S_{i-1} \setminus S_i$, $cost_c(v, t_i)$ is decreased by a factor of α, bringing



(a) Conservative approach



(b) Dynamic approach

Figure 4.7: Approximation ratio

the cost back to its normal value. These nodes are no longer watched. The costs of nodes $v \in S_i \cup S_{i-1}$ is unchanged, as they continue to be watched.

Several defender strategies were considered. Two simple strategies are the uniform and the cost-effective strategy. A defender using the uniform strategy selects a set of k nodes

uniformly at random from V during each interval. This is a very simple, low-overhead strategy that does not consider any structural properties of the graph, nor any of the node weights.

The cost-effectiveness of a node $v \in V$ is defined as $e(v, t_i) = reward_c(v, t_i)/cost_c(v, t_i)$. Very cost-effective nodes are those who offer great rewards at a relatively low cost. A defender adopting the cost-effective strategy selects a set $S_i \subseteq V$ of k nodes, where each node $v \in S_i$ is sampled with probability proportional to $e(v, t_i)/\sum_{u \in V} e(u, t_i)$. The cost-effective strategy is a generalization of the uniform strategy that maintains its simplicity. However, the cost-effective strategy takes more information into account when deciding the allocation of resources. A more sophisticated strategy is the betweenness strategy, which is described in the following subsection.

4.3.1 Betweenness strategy

Betweenness centrality is a fundamental measure of node importance that is well-studied in network analysis. The betweenness centrality of a node is defined in terms of the proportion of shortest paths that pass through it. Thus, a node with high betweenness is one that connects many other nodes to one another – such as a boundary node connecting tightly-clustered subgraphs. This property – the connection between clusters – is a natural measure of importance in many types of networks, including power, communication, and disease transition networks [98]. For $u, v, w \in V$, suppose that $\lambda_{v,w}$ is the number of shortest paths from v to w, and $\lambda_{v,w}(u)$ is the number of such paths that include u. Then the betweenness centrality of u is calculated as

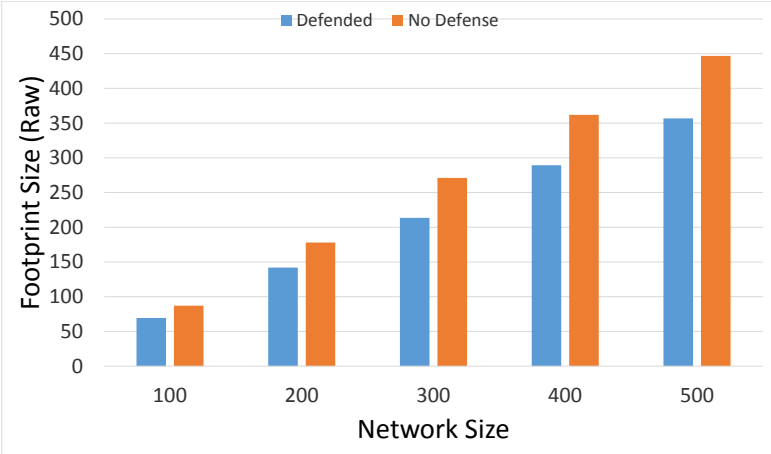
$$C(u) = \sum_{\substack{u \notin \{v,w\} \\ \lambda_{v,w} \neq 0}} \frac{\lambda_{v,w}(u)}{\lambda_{v,w}}.$$

Selecting the highest betweenness nodes can sometimes backfire when there is a great overlap in the shortest paths that contribute to a pair or more selected nodes. In that case,

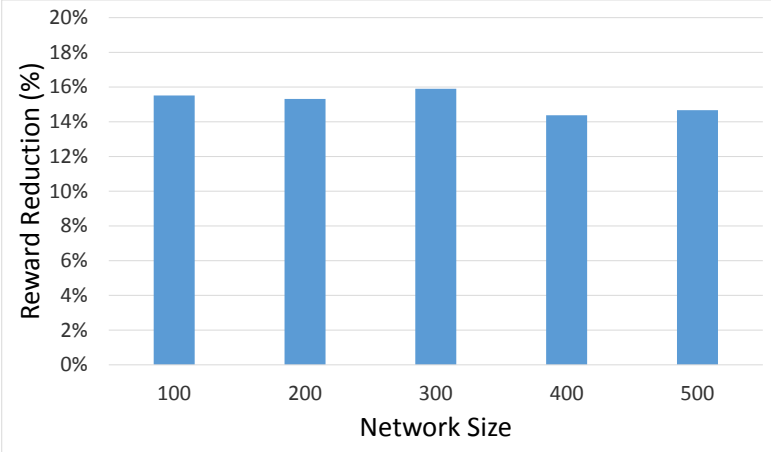
the coverage with respect to shortest paths provided by these nodes is largely redundant. Consequently, in practice one computes the list of top k nodes with respect to betweenness adaptively, removing all edges incident to a selected node and recomputing before selecting a subsequent node. This method of finding the top k betweenness nodes is designated as *adaptive-k* betweenness. Fortunately, there exist online polynomial time exact and approximate algorithms for computing the betweenness centrality of a graph that is subject to the addition and removal of nodes [99]. Furthermore, there exist online approximate adaptive betweenness centrality algorithms intended to scale to very large graphs [100].

A defender implementing the betweenness strategy interprets the undirected, unweighted graph $G = (V, E)$ as a directed, weighted graph $G' = (V, E', w)$, where $(u, v), (v, u) \in E'$ for every $(u, v) \in E$ and for every $(u, v) \in E'$, $w(u, v) = cost_c(v)/reward_c(v)$. In this graph, the weight of an edge connecting nodes u to v has weight which is the inverse of the cost-efficiency of v. That is, the more cost-effective (and therefore attractive to the attacker) v is, the more likely shortest paths are to go through it due to the weighting on the nodes. Hence, the more cost-effective a node, the more likely it is to have high betweenness centrality. This disproportionately skews in favor of cost-effective nodes that have high betweenness centrality in the unweighted version of the problem. Hence, the defender computes the adaptive betweenness centrality of the graph G' and selects the top k nodes in each time interval.

Figure 4.8a compares the malware footprint before and after the deployment of a defender's strategy. In this example, the attacker is employing the cost-effective strategy, whereas the defender is selecting k nodes to watch using the adaptive-k betweenness strategy, where k is set to 10% of the number of nodes in the network. These nodes increase their cost for a time interval by a factor $\alpha = 10$, resulting in a 20% reduction in the number of compromised nodes, and a 15% reduction in attacker's reward across all network sizes.



(a) Comparison of number of compromised nodes



(b) Reduction in attacker's reward

Figure 4.8: Impact of defensive action on malware

4.4 Summary

This chapter describes a practical solution for the attacker's problem of building a malware footprint, as posed in the prior chapter, while also addressing the implicit corresponding problem defender's problem of mitigating this threat using a dynamic defense. Furthermore, the model and algorithm quantitatively considers (i) cost incurred by APT actors to

compromise and persist within a target system; (ii) value gained by persisting in the system over time; (iii) how the malware footprint evolves over time when, to maintain stealth, the attackers have constraints on the amount of potentially detectable activity they can perform.

Chapter 5: Framework Evolution

This section discusses several important refinements to the basic framework presented in Chapter 4. In particular, the improvements include (i) differentiating the views of the attacker and defender; (ii) capturing how reward values can change over time due to attacker action and normal network activity; and (iii) accounting for non-deterministic attackers.

5.1 Attacker and Defender Views

In the previous chapter, attacker and defender shared a common view of the target network. This common view included not only knowledge of endpoints and the connectivity between them – represented by the graph $G = (V, S)$ – but also the reward and cost values for each node in the network. However, a more accurate and realistic representation requires to model attacker and defender separately, as their respective views of the network are likely to differ. For example, given that the intent of this work is to address stealthy attackers, it is assumed that the defender has no knowledge of the attacker's position within network. As shown in the Ponemon research [6], the defender is unlikely to be aware of the attacker's presence at all. However, the attacker is fully aware of its own position.

The attacker's view is modeled as a graph G_A and the defender's view as a graph G_D. For the purposes of the work presented here, the two graphs have the same nodes and edges – i.e., the attacker, as stated earlier, has full knowledge of the network topology – but the values of each node's cost and reward may differ. The attacker and defender views were modeled as two separate graphs – rather than simply introducing different cost and reward functions for attacker and defender – to facilitate further research beyond the scope of this article. For instance, an attacker may initially have only partial knowledge of the network topology, resulting in graphs with different sets of nodes and edges.

Modeling asymmetrical attacker and defender views as described above has significant implications for the model. It is reasonable to assume that an APT actor has memory of its own actions during an extended intrusion. For instance, the attacker, hypothetically, would recall compromising and exfiltrating information from a database during the initial stages of a persistent intrusion. In such an instance, in subsequent time intervals, this database would be presumably be less appealing to the attacker, given that information has already been partially exfiltrated. A similar consideration holds for attackers seeking to sabotage or otherwise degrade the integrity of a system. Hypothetically, if the attacker has degraded the capabilities of a given device, then the attacker would likely perceive that device as having less value.

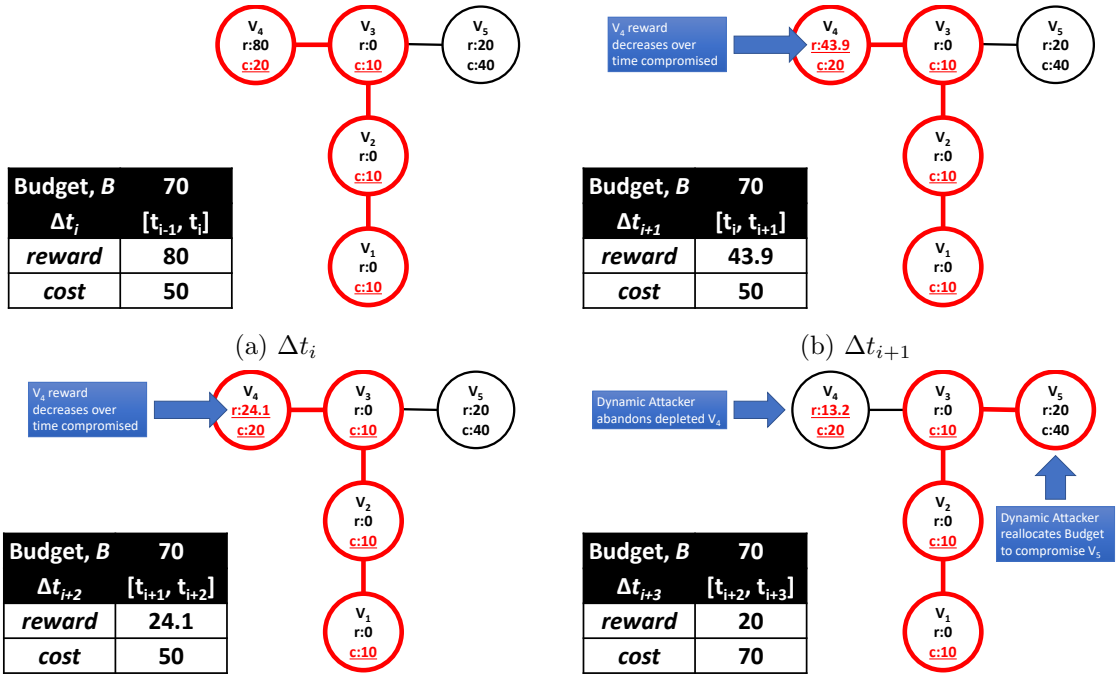
Given the lack of knowledge about the attacker's actions and perceived costs and rewards, defenders may be allocating defensive resources in an non-optimal manner. For instance, consider the aforementioned database residing in a network, particularly if the database were to contain sensitive or otherwise valuable information. Intuitively, the defender recognizes the relative value of the database and allocates significant resources to defend it throughout the time horizon. However, consider the implications should the attacker manage to compromise the database and successfully exfiltrate its data undetected. Presumably, maintaining the foothold on the high-value target would be difficult, considering the defender resources allocated to it. Having exhausted the value on the database, the attacker has now an incentive to abandon the database and allocate resources to compromise other nodes in the network instead. A defender which overtly concentrates defensive resources on a limited segment of the network (even though regarded as high-value) would be unlikely to detect or deter the attacker which may otherwise devote the majority of its resources and time compromising other network assets.

Therefore, when modeling the differences between the two views, the actions of the APT actor only affect G_A and not G_D. If the defender does not employ dynamic defenses, the defender's view is nearly static with the exception of minor network fluctuations, which are discussed later in this paper. Thus, the defender's view will not account for the reduction

in the cost associated with nodes that have been already been compromised, as described in the prior work. Nor will the defender be privy to the *Compromise Reward Decay*, which is described in the next section as part of several refinements of the original reward model.

5.2 Reward Model Refinements

In this section, several important refinements of the reward model that was presented as part of the earlier model of the quantification framework [101] are described.



(a) Δt_i (b) Δt_{i+1}

(c) Δt_{i+2} (d) Δt_{i+3}

Figure 5.1: Compromise Reward Decay Example ($\lambda = 0.4$)

5.2.1 Reward Decay

After compromising a node in the target network, an APT actor can perform various *actions on objectives* according to its original goals, which may include data exfiltration or sabotage. As the attacker accomplishes its objectives, the node may become less valuable.

For instance, if the attacker's objective is to exfiltrate data from a node, the residual value of that node decreases as fewer data items of interest are left to exfiltrate. This can be modeled by introducing a decay function that captures how the value of a node decreases over time. Depending on the specific circumstances the decay could be linear or exponential. Without loss of generality, the model assumes an exponential decay for the purpose of illustration. An exponential decay function can model scenarios in which the attacker gains diminishing returns by maintaining a node over time, whereas a linear decay function can model scenarios like data exfiltration at a constant rate (i.e., the same amount of data is exfiltrated during each time interval).

This refinement of the model can capture the attacker's attempt to dynamically prioritize targets while exfiltrating data or conducting other actions on objectives. For example, if a banking system were to be compromised, the attacker could prioritize exfiltration of the most recent records first – which presumably are more valuable than older records. An actor with other objectives, such as manipulation or destruction of data (as seen with the Shamoon APT) could also prioritize information or services to disrupt. Once these initial, high-priority goals are met, the attacker could proceed to pursue lower-priority (and presumably lower-value) objectives. The rate at which the decay occurs is controlled by the exponential decay constant λ. Ignoring other factors, $\forall v \in C$, the reward associated with node v at time t_i (i.e., during the time interval Δt_i) can be computed as follows:

$$reward(v, t_i) = \begin{cases} reward(v, t_0), \text{ if } t_i \leq t_c \\ reward(v, t_0) \cdot e^{-\lambda \cdot (t_i - t_c)}, \text{ if } t_i > t_c \end{cases} \tag{5.1}$$

where t_c is the time at which v is compromised. In other words, the reward remains constant until the compromise, and then starts decreasing. Equation 5.1 assumes that the decay constant λ is the same for every node in the network, but in real-world scenarios the rate of decay may be different for different assets. Dropping this assumption and generalizing the model would be straightforward, therefore, for the sake of presentation, it is assumed that

all assets have the same decay constant.

Figure 5.1 depicts a simple example from the perspective of the attacker (i.e., the figure shows various snapshots of graph G_A) and illustrates how the reward associated with a compromised node decreases over time, triggering changes in the overall APT footprint when APT actors employ the dynamic approach described earlier. In this example, at a time t_i, the APT malware has established a foothold within the target network, comprising the nodes highlighted with a bold red outline in Figure 5.1. At this time, v_4 is the only node within the footprint with a non-zero reward value. In subsequent time intervals, the reward value decreases exponentially according to Equation 5.1, with $\lambda = 0.6$. Eventually, the decay causes the value of the node to drop to the point that the attacker, consistently with the dynamic approach, abandons the node and compromises v_5 instead in order to maximize the reward within the next time interval.

The value of a node starts to decay in the first time interval following its compromise. Thus, if a node was compromised at time t_i, then the effect of the decay will start manifesting at time t_{i+1}. Also, in cases where the attacker abandons a node, the decay process halts, as the APT actor is no longer directly impacting the node. In the example, during the time interval Δt_{i+4} (not shown in the figure), the value of node v_4 would not further decrease as the attacker is no longer actively operating on that node.

Therefore, based on Equation 5.1, the Compromise Reward Decay for a node $v \in V$ at time $t_i \in \mathcal{T}$ can be defined as

$$decay(v, t_i) = \begin{cases} 0, \text{ if } t_i \leq t_c \\ reward(v, t_0) \cdot \left(1 - e^{-\lambda \cdot (t_i - t_c)}\right), \text{ if } t_i > t_c \end{cases} \tag{5.2}$$

where t_c is the time at which v is compromised. This distinction is significant for proper calculation of reward fluctuations, which were alluded to previously, and are detailed in the next section.

5.2.2 Reward Fluctuations

Operational networks change over time, fluctuating for a variety of reasons, even without the intervention of a third party. In our previous model, the reward an attacker gains from compromising nodes remains static throughout $\mathcal{T}$. However, operational networks are rarely static. Thus, to model networks in a more realistic fashion, the *reward* value of individual nodes should be modeled dynamically as well. A node may *accrue* value over time for any of the following reasons:

- New data is stored on the node
- New data is generated on the node
- External or environmental circumstances increase the value of the node

A node may *lose* value over time for any of the following reasons:

- Data is removed from the node
- Data becomes obsolete
- External or environmental circumstances decrease the value of the node

Outside circumstances can influence reward values, even if the affected nodes do not directly change. For example, if a politician formally announces candidacy for an elected office, the value of records related to that politician would become more valuable for rivals, though the records may actually be static. Correspondingly, data may become less valuable for a variety of reasons. For example, records pertaining to supply or development of a specific pharmaceutical may become less valuable if similar or superior competitors enter the market. However, these specific examples are somewhat extreme and are outside of the more typical day-to-day fluctuations which this work considers.

Intuitively, these fluctuations impact every node in the network – for both the attacker and the defender – as opposed to the previously-discussed Compromise Reward Decay, which only impacts compromised nodes in the attacker's view. Thus, these fluctuations apply

equally to both G_A and G_D. We chose to model this phenomenon using a random variable $fluct$ following a normal distribution with a mean of μ_{fluct} and standard deviation of σ_{fluct}. As a result of these shifting reward values, the APT actor may be further incentivized to dynamically alter the malware footprint over time in order to to maximize the rewards. We assume that, through its extensive reconnaissance efforts, the APT actor is aware of the specific interactions of the target network and thus will be able to predict the value of data residing on the nodes.

We denote the cumulative effects of these factors on a node $v \in V$ at time t_i (with $i \in [0, m]$) as $fluct(v, t_i)$, with $fluct(v, t_0) = 0$. Reward fluctuations at time t_i can be thought of as the difference in reward between two consecutive time points t_{i-1} and t_i, i.e., the variation over time interval Δt_i, when no decay due to attacker's activity is considered. However, we need to define constraints to account for the fact that, intuitively, neither reward fluctuations nor decay can render the reward of a node negative. Thus, in the defender's view G_D, $\forall v \in V$, the reward fluctuation over a time interval Δt_i is bound by the following equation:

$$reward(v, t_i) = reward(v, t_{i-1}) + fluct(v, t_i) \geq 0, \tag{5.3}$$

thus, $fluct(v, t_i) \geq -reward(v, t_{i-1})$ must hold. Similarly, in the attacker's view G_A, the reward fluctuation over a time interval Δt_i is bound by the following equation:

$$reward(v, t_i) = reward(v, t_{i-1}) - decay(v, t_i) + fluct(v, t_i) \geq 0, \tag{5.4}$$

thus, $fluct(v, t_i) \geq -reward(v, t_{i-1}) + decay(v, t_i)$ must hold.

5.3 Formal Reward Model

In summary, the revised reward model can be summarized as follows.

Defender's view G_D. The defender has no knowledge about data that may have already been exfiltrated by the attacker, therefore the reward of a node from the defender's perspective is not affected by the attacker's actions.

$$\forall v \in V, \forall i \in [1, m], reward(v, t_i) = reward(v, t_0) + \sum_{k=1}^{i} fluct(v, t_i) \tag{5.5}$$

Attacker's view G_A. The attacker is aware of which nodes have already been depleted, so it can adjust its perceived value of nodes accordingly.

$$\forall v \in V, \forall i \in [1, m], reward(v, t_i) = reward(v, t_0) - decay(v, t_i) + \sum_{k=1}^{i} fluct(v, t_i) \tag{5.6}$$

Probabilistic Route Selection

Within each time interval, the attacker iteratively enumerates potential reachable targets and determines routes to such targets. These routes are scored in terms of reward and cost, similarly to our previous model, and then optimal routes are chosen based on one of the *route selection strategies* discussed earlier, namely Greatest_Reward, Lowest_Cost, and Cost_Effective. However, this approach results in malware footprints which are deterministic in nature and thus highly predictable. More sophisticated attackers may try to be less predictable by introducing some randomness in the route selection process. To model this behavior, we introduce the notion of Probabilistic Route Selection. As usual, each route is ranked with respect to the chosen objective function. However, instead of selecting the highest-ranking routes, a probability proportional to the metric to be optimized is assigned to each route, and routes are selected based on such probability distribution. Later in the paper we compare this probabilistic approach against its deterministic counterpart.

5.4 Defender's Model

As mentioned earlier, we assume that the defender is not aware of the attacker's actions, which we model using asymmetrical attacker and defender views of the target network. In our previous model, the defender would select k nodes during each time interval and deploy detectors on those nodes, thus increasing the attacker's cost to compromise and maintain those nodes in its footprint during that time interval. A determined and persistent attacker, if allocating sufficient resources (budget), could still compromise such nodes and bypass detectors, using a variety of tools and capabilities, including zero-day exploits. However, allocating a substantial portion of their budget to compromise a few heavily-guarded nodes would force the attacker to make tradeoffs, such as dedicating significantly more time to evade the detectors. As a result, the attacker may ultimately forgo compromising more desirable portions of the network.

This defensive approach present clear advantages. Most notably, dynamically shifting detector placement over time forces the stealthy attacker to constantly reassess the network and shifting its own position over time as well. This abandoning and recapturing nodes detracts from the attacker's objectives, whether it be exfiltration, sabotage or some combination of both. However, this defensive approach also has limitations. Our prior approach probabilistically placed detectors with weights calculated using Adaptive Betweenness Centrality. This approach used only the structure of the graph and the reward/cost values from the defender's perspective. With the refinements to our model, the attacker's view can differ significantly from the defender's view. Furthermore, the previous defensive approach was agnostic to the understanding of the attacker. In this paper, we propose two new defender models in which the defender simulates the behavior of an attacker.

For each time step, the defender takes its view of the network and simulates an attack using the attack model for *sim* future time steps. For example, at time t_1, the defender will simulate an attacker at times t_1 and t_2. At time t_2 the defender will pick another random node and simulate an attack at times t_2 and t_3). For each time step, the output of this process is the Defender Simulated Malware Network (DSMN) and Defender Simulated

Malware Footprint (DSMF).

The defender uses the DSMN and DSMF to calculate a set of weights which are then employed by the defense to probabilistically place detectors in the network. The two new defensive models proposed differ in the means by which they employ the DSMN and DSMF to calculate the weights.

1. Simple Cumulative Weighted Probabilistic - During each time interval, the Defender selects k nodes using a simple frequency tally of infected nodes in the DSMFs as weights. For example, a given node is infected in 3 DSMFs, then that node has a value of 3. Hypothetically if the sum tally were 100, then that node would have a 3% chance of being selected when the defender determines detector placement.

2. Percolation Centrality Cumulative Weighted Probabilistic - Percolation Centrality is a measure employed in a number of contexts, perhaps most prominently in disease research to model the spread of epidemics [102][103]. The concept is similar to Betweenness Centrality, but incorporates a "percolation" value to model the level of by which a given node is infected. If all nodes in the network have the same percolation value, Percolation Centrality reduces to Betweenness Centrality. In the context of this work, during each time interval, the Defender calculates the Percolation Centrality of the DSMN. The cumulative average of these Percolation Centrality calculation are used as weights for detector placement. We employ a open-source library to calculate Percolation Centrality, as we will discuss in a later section.

5.5 Evaluation

In this section, both the refinements to the original Attacker Model and the two new Defender Models are tested.

5.5.1 Experimental Setting

In this section, the refined model is studied using similar settings employed in prior work. The algorithm across networks of various sizes - 100, 200, 300, 400 and 500 nodes. For each network size, 30 different network topologies were generated and simulation results were averaged over different network settings. There is a lack of existing models that capture the connectivity of an enterprise network at both layer 2 and layer 3, as most research concentrates on ISP-level networks. Therefore, in order to generate different network topologies, scale-free networks, known to accurately capture the connectivity in ISP networks at router level [97], were used to synthesize enterprise network topologies of different sizes.

The Python-based NetworkX 2.3 library was leveraged to generate these networks using the built-in Holme and Kim algorithm, which is a variant of Barabási-Albert (BA) model. In the BA model, new nodes are incrementally added to the network. Each new node is connected to one of the existing nodes in the graph. The probability that a new node is connected to any given existing node is proportional to the degree of the existing node. Therefore, a node with a higher degree has a higher probability of becoming the new node's neighbor, imparting the scale-free property. The Holme and Kim algorithm tends to generate networks with more clusters than typical BA networks, to more closely model enterprise networks.

The NetworkX library is also employed to calculate the network's Percolation Centrality for the purposes of implementing Percolation Cumulative Weighted Probabilistic defense model mentioned previously.

5.5.2 Refined Model Experimental Setup

In the experiments, both Compromise Reward Decay and Route Selection were implemented and their effects noted with regard to various metrics detailed below.

5.5.3 Metrics

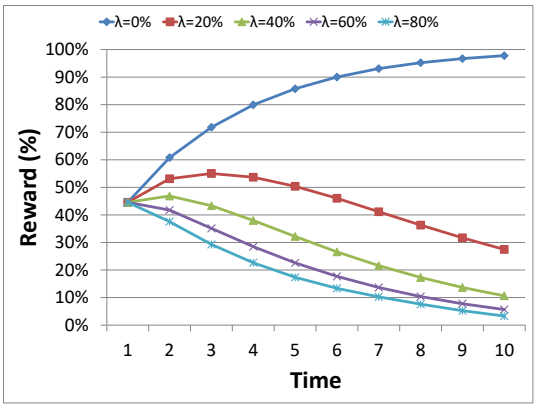
The metrics captured during the experiments are described and shown in the following sections.

- Reward - Reward values are presented as the percentage of the reward in the Malware Footprint (and thus captured by the attacker) each time step vs the total reward available in G. This facilitates comparisons across networks of different sizes.
- Footprint Size - Similarly to above, footprint sizes are presented as the percentage of the nodes in the Malware Footprint (and thus captured by the attacker) each time step vs the V/inG. This facilitates comparisons across networks of different sizes.
- Jaccard Distance - For the purposes of this paper, Jaccard Distance is employed as new metric to measure the volatility of the Malware Footprint over time. Jaccard distance measures the difference between two sets on a scale of 0 to 1, with 0 indicating no change between sets and 1 indicating that the sets share no members. The set of nodes comprising the Malware Footprint across time intervals is compared. At a given t, the Jaccard Distance is calculated using the Malware Footprint at t and $t-1$. Thus, the Jaccard Distance calculated at t_1 represents the change in the Malware Footprint between t_0 and t_1. In all cases, the Jaccard Distance at t_1 1, as the malware has compromised no nodes at t_0 and compromised some nodes by t_1. The Jaccard Distance at t_2 is the difference in the Malware Footprint between t_2 and t_1 and so on.
- Runtime - The runtime presented is the cumulative runtime over the time horizon. In comparison to the prior work, simulation runtimes are vastly more efficient.

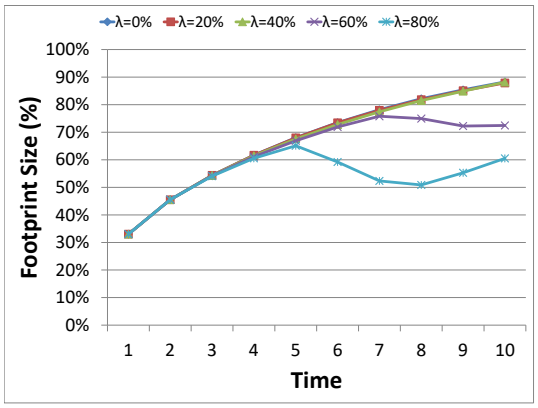
5.5.4 Reward Decay

When considering the effect of varying levels of reward decay, one should expect that higher rates of decay should result in decreased reward for the attacker over time. This intuition is shown to be true as shown in Figure 5.2, which presents the aforementioned metrics for

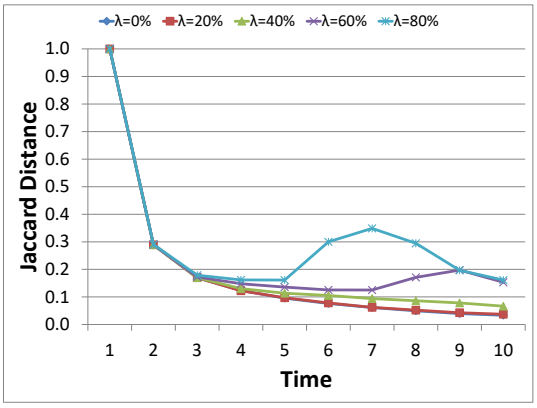
an attacker employing a Cost-Effective Strategy and Dynamic Approach with Compromise Reward Decay calculated with λ values of 0%, 20%, 40%, 60% and 80%.



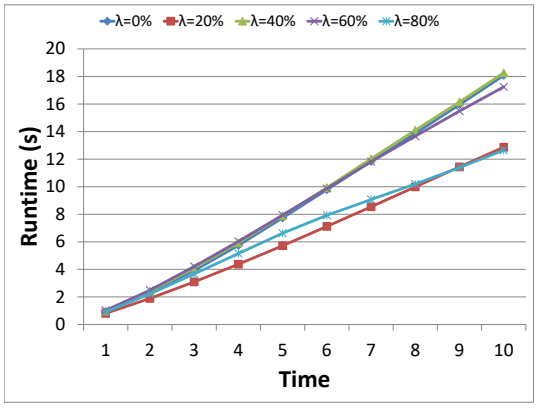
(a) Reward



(b) Footprint Size



(c) Jaccard Distance



(d) Runtime

Figure 5.2: Effects of Reward Decay

Of note, Figure 5.2b shows that the *reward* decreases at the lower rates of Compromise Reward Decay despite the Malware Footprint Size retaining the same number of nodes. At higher rates of Compromise Reward Decay, it is apparent that the Dynamic attacker is abandoning nodes which have lost their value and allocating resources to compromising "new" nodes instead as indicated in Figure 5.2c. However, as compromising these new nodes is more costly than simply maintaining previously-compromised nodes, the total number of

nodes compromised is severely limited at higher rates of Compromise Reward Decay.

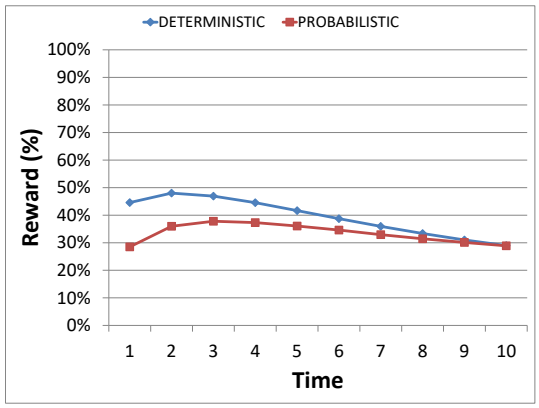
5.5.5 Probabilistic vs Dynamic Route Selection

The aim of modeling an attacker which employs Probabilistic Route Selection (as opposed to Deterministic Route Selection) was to add an element of unpredictability to the attacker model. In employing Probabilistic Route Selection, it is expected that the attacker is less effective at extracting value from the network (due to employing non-optimal decisions), but has a more volatile footprint. In reviewing the results shown in Figure 5.3, which also features an attacker employing a Cost-Effective Strategy and Dynamic Approach, these expectations hold mostly true. Curiously, Figure 5.3a shows that while Deterministic Route Selection is initially more effective at accruing *reward* on a per time interval basis, this is not true by the end of time horizon.

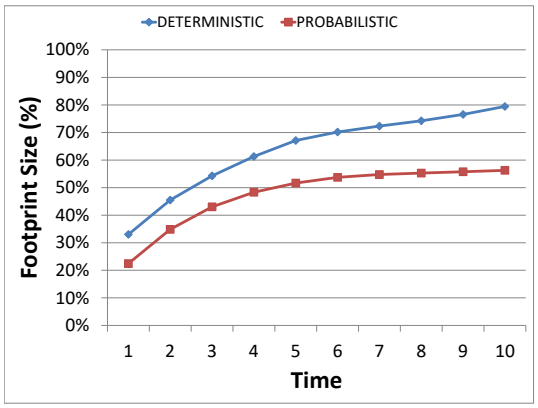
This is the result of the Deterministic attacker compromising the most optimal nodes first leaving only suboptimal nodes left by the end of the time horizon once the node value decayed. Results were calculated across all aforementioned tested values of Compromise Reward Decay. This allows us to postulate that over longer time horizons, the difference between attackers employing Probabilistic Route Selection vs Deterministic Route Selection on the basis of *reward* is negligible. However, an attacker employing Probabilistic Route Selection would maintain a smaller and more volatile and more unpredictable Malware Footprint, both are traits which are desirable to avoid detection.

5.5.6 Evaluation of Defended Networks

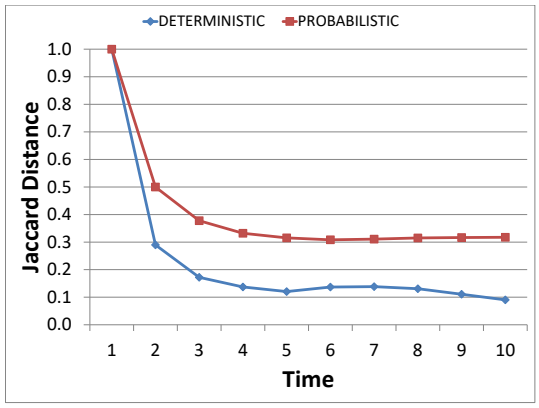
In this section, the new Defender Models are examined in comparison to both the Adaptive Betweenness Centrality described in the prior work and a comparison no defense. For these results, the parameter k is a 10% of the network size and SIM is 2 time steps to reduce computation time. Results are averaged across all route selection strategies, tree evolution approaches, and both Probabilistic and Deterministic Route Selection.



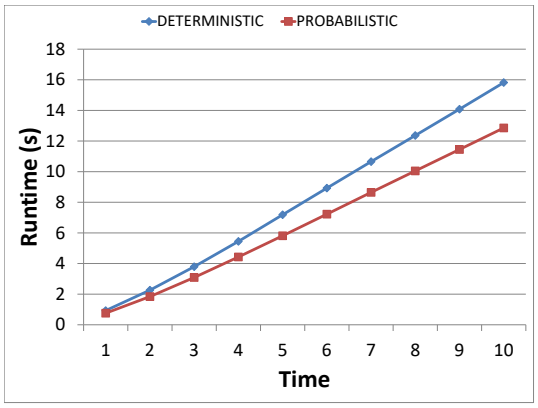
(a) Reward



(b) Footprint Size



(c) Jaccard Distance

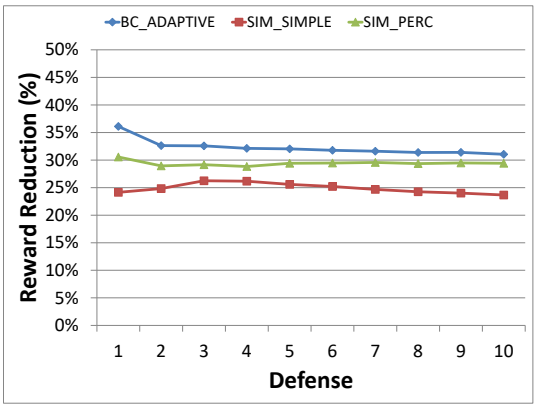


(d) Runtime

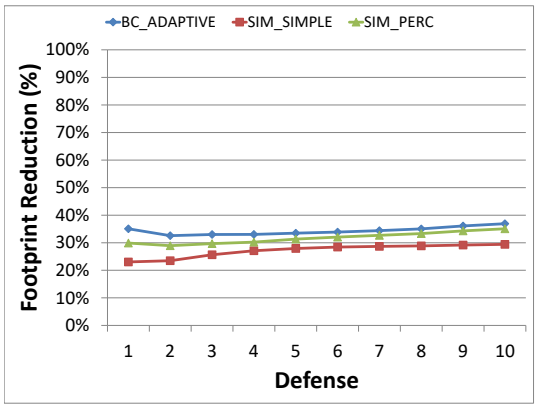
Figure 5.3: Effects of Route Selection

It is shown that in all cases, the Defender reduces the *reward* value gained by the attacker with the previously-developed Adaptive Betweenness Centrality (BC_ADAPTIVE) slightly outperforming the simple and percolation centrality-based simulated defenses (SIM_SIMPLE and SIM_PERC respectively).

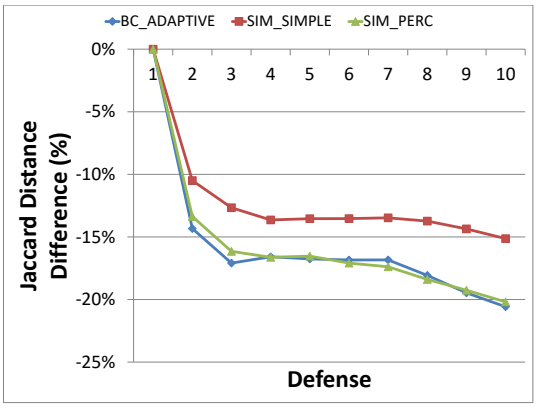
The defenses also causes the attacker to forego nodes, reducing the Malware Footprint size. Appropriately, the Jaccard Distance overall across the time horizon is also reduced, as the malware has increase difficultly spreading through the network. The Jaccard Distance between an empty set (the t_0 case) and any non-empty set (the case at t_1) is always 1. Thus, the difference in Jaccard Distance at t_1 between the defended and non-defended scenarios



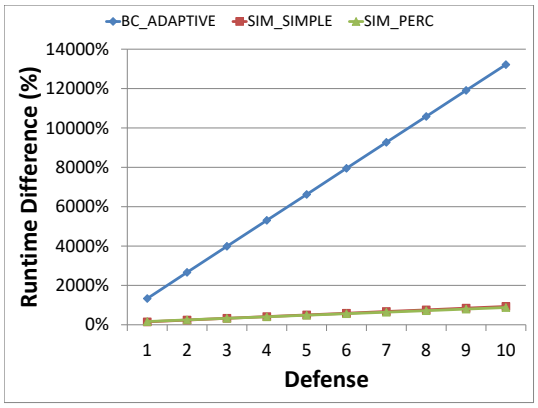
(a) Reward



(b) Footprint Size



(c) Jaccard Distance



(d) Runtime

Figure 5.4: Effects of Various Defender Models vs No Defender

is 0.

Of greater note, while the simulated defenses have similar effectiveness against the attacker, it is clear in Figure 5.4d that the Adaptive Betweenness Centrality requires by far the most computational time. Scalability across long time horizons is a more considerable concern compared to the simulated approaches proposed in this paper.

5.6 Summary

This chapter builds upon the framework detailed in the previous chapter. In addition to the improvements to the defender model, the three major contributions for the attacker are considerations for (i) how reward values may shift over time due to natural fluctation and attacker activity, (ii) non-deterministic attackers and (iii) separation of attacker and defender views. This last contribution is particularly critical for development of defenses which incorporate cyber deception.

Chapter 6: Concurrent MTDs

MTD defenses also have disadvantages. Nearly all MTDs incur a performance cost [104]. During reconfiguration, the full capabilities of the system are rarely available. Increasing the rate of reconfiguration further increases this overhead cost and reduces the availability of resources, negatively impacting system performance. This effect can be compounded by the implementation of multiple MTDs. Decisions regarding the deployment of MTDs must balance the security benefits with the corresponding MTD overhead and implementation costs. As another contribution, a method to predict availability with multiple MTDs operating concurrently is also offered.

Real-world problems, such as those posed by APTs, and limitations that exist outside the theoretical realm, require implementation and testing of MTDs against a realistic attack model. Doing so will validate the model and identify important implementation details that might otherwise be overlooked. The chapter presents an experimental framework which models multiple MTDs and an automated attacker to validate the analysis. Then, the results are applied to a utility function to determine the reconfiguration parameters that offer the best tradeoff of security and availability for the system based on the user's objectives.

The attack model employed in this paper adheres closely to the expected behavior described in literature through the aforementioned Cyber Kill Chain [105] which is also employed by APTs. The attack consists of multiple steps including several successive reconnaissance steps to identify a vulnerable target, followed by an attack against that target. MTDs can disrupt this attack at several points in the kill chain and are chosen accordingly. The attack model and the corresponding defender model are described in more detail in the following sections.

6.1 Attack Model

The attacker searches the environment for vulnerable operating systems, then searches those targets for vulnerable services, and then finally launches a corresponding exploit to establish control when such a service is discovered. The simulated attacker, operated by an automation script, performs the following actions, in order:

1. Target Scan: The attacker employs the Nmap Network Mapper tool to scan the environment. During this initial reconnaissance step, an attacker is simulated which has configured Nmap to perform a relatively non-intrusive scan. This models the behavior of an attacker which wishes to maintain stealth while probing the network for weaknesses.

   This initial scan process first attempts to detect whether a node is active on a given IP address and the operating system it is running by analyzing the pattern of open ports. Based on the response, the attacker identifies targets for further reconnaissance efforts. Unresponsive targets or those not considered to be vulnerable are not scanned further. The attacker does not attempt to perform more invasive scans such as service detection because the attacker seeks to minimize interaction to avoid detection during this step.

   For the purposes of this paper, the simulated attacker has identified the characteristics of the Metasploitable virtual machines as promising targets. Further actions would only be performed against those targets.

2. Service Detection: In this step, the simulated attacker has identified promising targets with active services and seeks to probe these targets further, performing a service scan to determine the name and version of the services operating on the node.

   Continuing the example, the Metasploitable VMs are running web services on port 80. Web services were selected due to their prevalence as a common avenue of attack. However, it is plausible that a real-world attacker may wish to target other services

based on any number of factors, such as the availability of exploits, impact of compromising the alternative service, or the belief that the compromise of such an alternative service would be more likely to be unnoticed.

If the service running is not considered to be vulnerable, the attacker proceeds to the next target previously identified during the target scan. If the attacker finds that the service running is vulnerable, it proceeds to the next step.

3. Launch Exploit: If a vulnerable service is discovered, the attacker will launch an exploit against the service. In the example, this results in a reverse connection back to the attacker. If it is not successful, the attacker proceeds to the next target identified by the target scan.

4. Attacker Success: The phase represents the end state where the now-compromised host has established a reverse connection back to the attacker. This closely corresponds to an attacker which has established full command and control over the target host. In this end state, data can be exfiltrated and other arbitrary actions undertaken. At this point, the overall attack is considered to a success.

Figure 6.1 depicts these steps in a process flow as performed against a single target IP address. After an attack is complete, if there were other hosts using the target OS found during the initial scan, these hosts are subsequently scanned and attacked in a serial fashion. This models realistic scenarios of attacker/defender behavior. An attacker would be discouraged from launching attacks indiscriminately for several reasons. Indiscriminate attacks against the network space would likely result in more rapid detection of attacker activities. More rapid identification of data breaches has been shown to result in significantly reduced costs to the defender [106].

During the attack process, reconfigurations may occur (described in the following section) which interrupt services to the attacker. In these cases, the attack is considered to be a failure, as discussed in greater detail below.

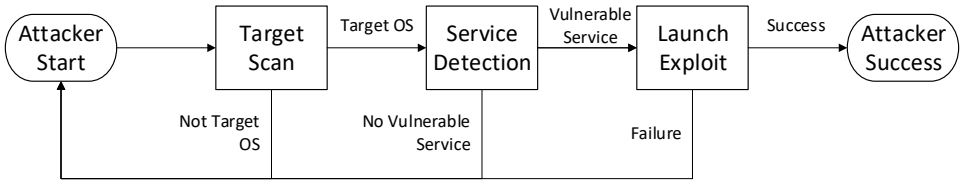


Figure 6.1: Attack Model

6.2 Defender Model

The attacker model requires the attacker to obtain several pieces of information before an attack can commence. This allows for several opportunities for deploy a MTD to disrupt the attacker. The attacker must find a vulnerable service running and must also have an IP address to attack. If the attacker does not obtain both of these, it will move on to the next potential target, if possible. If the attacker does not detect a vulnerable target, then the attack halts and is considered a failure.

Therefore, a reconfiguration which occurs during reconnaissance is likely to cause the attack to fail. Reconfigurations from each MTD occur randomly with interarrival times following an exponential distribution, which is commonly used when determining defender actions because of its memoryless property which prevents the attacker from predicting defender behavior [107, 108]. This property is leveraged during analysis. The average interarrival time for reconfigurations is denoted by μ, which may vary between MTDs. In this work also references the reconfiguration rate, which is inversely proportional to the interarrival time and is denoted by $\lambda = \frac{1}{\mu}$.

Service Reconfiguration

For the purposes of Service Reconfiguration, a simple service randomization scheme was developed, very similar to existing dynamic platform-based MTDs [38]. The MTD periodically changes the underlying implementation of the service in a manner transparent to the

typical user.

A target node employing the MTD has multiple implementations of a service available. Upon startup, the MTD randomly chooses one of the available services to run. The service remains active until the automation script randomly initiates a reconfiguration with average interarrival time μ_S. As a part of reconfiguration, the previously-running service is stopped and a new random service started. To increase randomness and prevent an attacker from predicting patterns in the service changes, the MTD may select the same service multiple times in sequence.

All web services were run on the Metasploitable VM. By default, Metasploitable contains a version of Apache containing a PHP vulnerability susceptible to a specific exploit. Then two other web services were installed which are not vulnerable to that exploit. The following web services were employed:

1. Apache (containing an unpatched PHP vulnerability)
2. Nginx
3. Lighttpd

This particular scheme is heavily influenced the MTD originally proposed by Huang [109], utilizing many of the same web services Huang selected in his own work. A diagram showing the relationship between the attacker and this MTD is shown in Figure 6.2.

IP Reconfiguration

For the second MTD evaluated, IP Reconfiguration was implemented using a simple IP randomization technique similar to other known dynamic network-based MTDs [38]. As with the Service Reconfiguration MTD, IP Reconfiguration is accomplished via an automation script. In this scheme, the VM changes its IP address with reconfigurations occurring with an average interarrival time of μ_{IP}. The pool of available IP addresses is equal to that

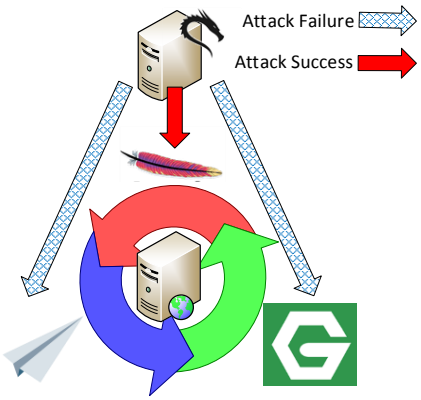


Figure 6.2: Service Randomization

of a Class C network (256), minus a small set of reserved IP addresses for the hardware environment.

Concurrent MTDs

The two MTDs must be capable of operating without interference. If the IP were to change while the service is being reconfigured, the service may not restart properly, resulting in additional loss of service. This scenario becomes increasingly likely as the reconfiguration rates increase. To ensure MTD compatibility, the two MTDs are implemented in a multi-threaded application with mutual exclusion (mutex) locks around the reconfiguration operations. This prevents the two MTDs from reconfiguring at the same time. The implementation of the MTDs working in conjunction with each other is shown in Algorithm 3.

It should be noted that these MTD defenses, as implemented, may not be practical for all web applications. In particular, web applications which require a stable, persistent connection would likely be disrupted. However, there exists a broad use case for applications employing stateless frameworks, such as the popular Representational State Transfer (REST). Authors of previously proposed MTDs [109] have developed MTDs with similar

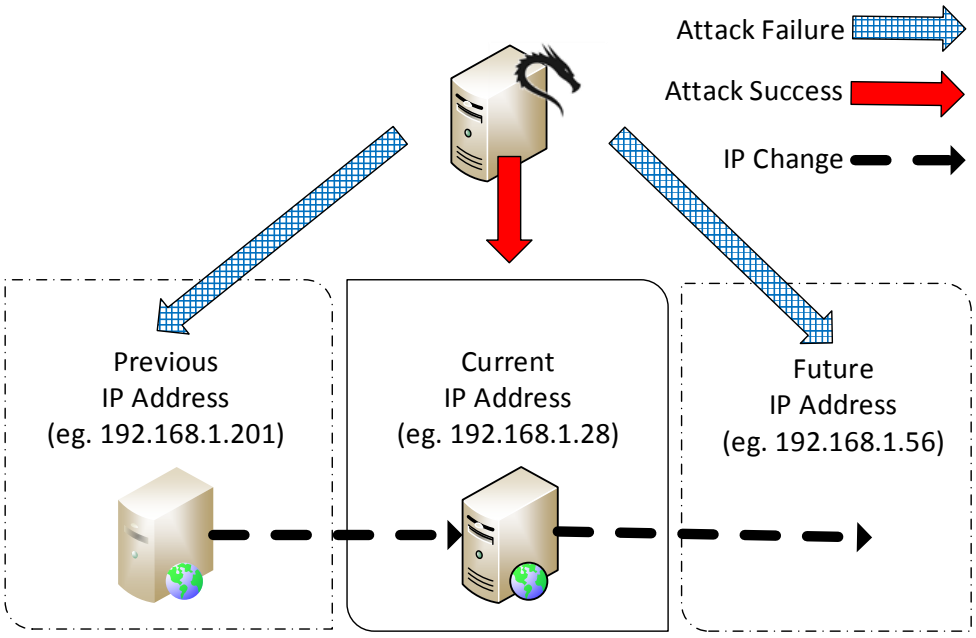


Figure 6.3: IP Randomization

operational restrictions.

6.3 Quantitative Analysis

Based on the attacker and defender model, the attacker's success rate can be determined. Individually, the MTDs might be analyzed in a manner similar to [110], where an attacker is assumed to win if they are in control for a certain period of time before a reconfiguration occurs. Likewise, an attacker in the attack model wins if can complete the attack sequence to find a vulnerable node and compromise it before being interrupted by a reconfiguration.

The availability of the system with MTD(s) in operation can also be computed. This represents the momentary loss of service caused by changing services or IP addresses and affects the attacker's success rate as well as benign users.

Algorithm 3 Concurrent MTD Implementation

```
1:  s ← Service reconfiguration rate
2:  i ← IP reconfiguration rate
3:
4:  function ServiceThread()
5:      while (currentTime − startTime > duration) do
6:          serviceWait ← random(s)                          ▷ Exponentially distributed
7:          sleep(serviceWait)
8:          lock(mutex)
9:          reconfigureService()
10:         unlock(mutex)
11:     end while
12: end function
13:
14: function IPThread()
15:     while (currentTime − startTime > duration) do
16:         IPWait ← random(i)                               ▷ Exponentially distributed
17:         sleep(IPWait)
18:         lock(mutex)
19:         reconfigureIP()
20:         unlock(mutex)
21:     end while
22: end function
23:
24: function main()
25:     init(mutex)                                          ▷ Mutex lock for threads
26:     if s > 0 then
27:         threadCreate(serviceThread)
28:     end if
29:     if i > 0 then
30:         threadCreate(IPThread)
31:     end if                                               ▷ Create threads
32:     if s > 0 then
33:         threadJoin(serviceThread)
34:     end if
35:     if i > 0 then
36:         threadJoin(IPThread)
37:     end if                                               ▷ Re-join threads
38: end function
```

6.3.1 Definitions and Assumptions

Because attacker and defender models are based on real-world constraints, the parameters of those models are defined and summarized below:

- T_a: The average time required to successfully complete an attack from start to finish,
- μ_S: The average interarrival time for service reconfigurations
- μ_{IP}: The average interarrival time for IP reconfigurations
- t_S: The average time required to complete a service reconfiguration

- t_{IP}: The average time required to complete an IP reconfiguration
- s: The total number of possible configuration states
- s_v: The number of vulnerable configuration states

T_a also includes reconnaissance steps required to perform an attack if they are also disrupted by the reconfiguration. However, some MTDs might not disrupt certain reconnaissance actions. μ_S and μ_{IP} are the parameters for interarrival times for reconfiguration, which are exponentially distributed. t_S includes the time required to stop the previous service, wait for all processes to shut down, start processes for the new service, and verify that the newly-started service is running. Likewise, t_{IP} includes the time required to bring the interface down and back up again with a new IP address. Because of the additional steps and requirement to connect externally to obtain the new IP address, verify it and ensure that the change is propagated to users, t_{IP} can be an order of magnitude larger then t_S. s represents the total number of possible states the system can be in between reconfigurations, with s_v being the number of those states that are vulnerable to an exploit.

6.3.2 Availability

Availability A is determined first, as this also affects attacker success rate. For MTDs operating individually, availability is calculated as the expected uptime per reconfiguration cycle (μ_S or μ_{IP}) divided by the total expected uptime plus expected downtime per reconfiguration cycle (t_S or t_{IP}, as appropriate), leading to the availability due to service reconfigurations:

$$A_S = \frac{\mu_S}{t_S + \mu_S} \tag{6.1}$$

and availability due to IP reconfigurations as:

$$A_{IP} = \frac{\mu_{IP}}{t_{IP} + \mu_{IP}} \tag{6.2}$$

Because the MTDs utilize mutual exclusion, the downtime from one MTD must occur during the other MTD's uptime. Therefore, overall availability is computed as the product of the two availability values:

$$A = A_S \cdot A_{IP} \tag{6.3}$$

6.3.3 Attacker Success Rate

Next, attacker's success rate is computed based on the expected attack time and reconfiguration rates. The defender model assumes that in order for an attack to be successful, it must be uninterrupted by a reconfiguration. Because of the memoryless property of the exponential distribution, as long as the system is not currently undergoing a reconfiguration, the expected time before the next configuration is equal to the respective value of μ, regardless of when the last reconfiguration occurred. Using the exponential distribution with $\lambda = \frac{1}{\mu}$, the base probability that the attack is successful is determined by finding the probability that the random variable $\mathcal{X}$, which represents the time before the next reconfiguration, is greater than the time required to execute the attack, or $P(\mathcal{X} > T_a)$. Solving using the probability distribution for the exponential function:

$$ps_S = P(x > T_a) = 1 - P(x \leq T_a) = 1 - (1 - F_x(T_a)) = e^{-\lambda_S T_a} \tag{6.4}$$

With the base probability that no reconfiguration occurred now calculated, the value based on the probability $\frac{s_v}{s}$ that it was already in a vulnerable state is adjusted, where s_v is the number of vulnerable states and s is the total number of states. In the case of service reconfiguration with three different services, $\frac{s_v}{s} = \frac{1}{3}$. In the case of IP reconfiguration, each state is considered to be equally vulnerable. If different states share common weaknesses, it would be reflected in s_v.

Using this methodology to determine attacker success rate also assumes the service is

already running at the time the attack begins. It is already established that there is an impact to availability by using MTDs, which is adjusted for by multiplying the attacker's chance of success by the availability. This further reduces the attacker's success rate and gives us an unintended benefit from an otherwise undesirable side effect of MTDs. The probabilities of attacker success for each MTD are thus:

$$ps_S = e^{-\lambda_S T_a} \cdot \frac{s_v}{s} \cdot A_S \tag{6.5}$$

$$ps_{IP} = e^{-\lambda_{IP} T_a} \cdot A_{IP} \tag{6.6}$$

When both MTDs are operating at the same time, a similar method is used to determine the attacker's success rate. If no reconfigurations are currently taking place at the time the attack starts, the attacker's probability of success is equal to the probability that both random variables $\mathcal{X}_s$ and $\mathcal{X}_{IP}$ are greater than the attack time T_a, adjusted as previously to account for number of vulnerable states and overall availability, or:

$$ps = e^{-\lambda_S T_a} \cdot e^{-\lambda_{IP} T_a} \cdot \frac{s_v}{s} \cdot A_s \cdot A_{IP} = e^{-T_a(\lambda_S + \lambda_{IP})} \cdot \frac{s_v}{s} \cdot A \tag{6.7}$$

If necessary, this method could also be further generalized for three or more MTDs.

6.4 Experimental Framework

In this section, the experimental framework used to demonstrate the analysis shown in Section 6.3 is detailed. This includes a description of the testbed environment with specific hardware and software configurations.

6.4.1 Experimental Environment

Experimental testing was performed using the Center for Secure Information Systems (CSIS) testbed environment located at George Mason University (GMU). This testbed environment is managed using XenServer 7 to quickly deploy virtual machines and associated networking services, such as DHCP, to allow the VMs to communicate.

As indicated previously, a target VM was developed with an OS image adapted from the freely-available Metasploitable and provided by Rapid7. This image is commonly used in security testing. The base VM was modified, enabling various web services for the simulated attacker to compromise.

The attacker is represented by a VM on the network which is adapted from the Rolling release version of Kali Linux, a Linux distribution developed specifically for security testing. The attacker VM was deployed with Nmap 7.50 and version 4.14.28-dev of Metasploit, a widely available security penetration testing tool. Attacks are launched using the php_cgi_arg_injection Metasploit exploit. The exploit targets unpatched versions of the Apache 2.2.8 HTTP Server, which was originally released in 2008 [111]. The exploit php_cgi_arg_injection specifically targets CVE-2012-1823 [112], an argument injection vulnerability. The vulnerability was exploited in the wild in June of 2013.

For each set of configuration parameters, 500 independent trials were conducted, consisting of a complete attack from the Metasploit node. Another process emulated a legitimate user, consistently trying to connect to the web server to perform small stateless transactions to monitor effect on availability.

6.5 Experimental Results

Results from the experiments performed are presented in this section. In addition to measuring attacker's success rate and availability, the time T_a required from start to finish of the attack where the target must undergo no reconfigurations, the average time t_s required to reconfigure a service, and the average time t_{IP} required to reconfigure an IP address

were also measured. These values are shown in Table 6.1 and are used to along with the methods in Section 6.3 to predict the attacker's success rate and availability and compare them to the collected results.

Table 6.1: Average Attack and Reconfiguration Times

Variable	**Observed Value (seconds)**
T_a	28.01
t_S	0.635
t_{IP}	9.59

6.5.1 Service Reconfiguration

As seen in Figure 6.4, adding service reconfigurations greatly decreased the attacker's chance of success. In the case where service is static, it is assumed a worst-case scenario where the only service is the vulnerable Apache service. Therefore, the largest decrease in attacker success rate came from the diversity introduced by the initial introduction of the MTD. However, as the reconfiguration rate increases, the service randomization MTD was able to prevent more than the expected 33.3% of the attacks directed against it, and these values match with the predictions.

Service reconfiguration also reduced the availability slightly, as seen in Figure 6.5. Availability decreases up to 3.3% compared to the static case at the highest reconfiguration rate.

6.5.2 IP Reconfiguration

As expected, Figure 6.6 shows adding IP randomization also decreased the attacker's success rate, although not to the same extent as the diversity-based service reconfigurations. However, the observed values follow the downward trend, as predicted.

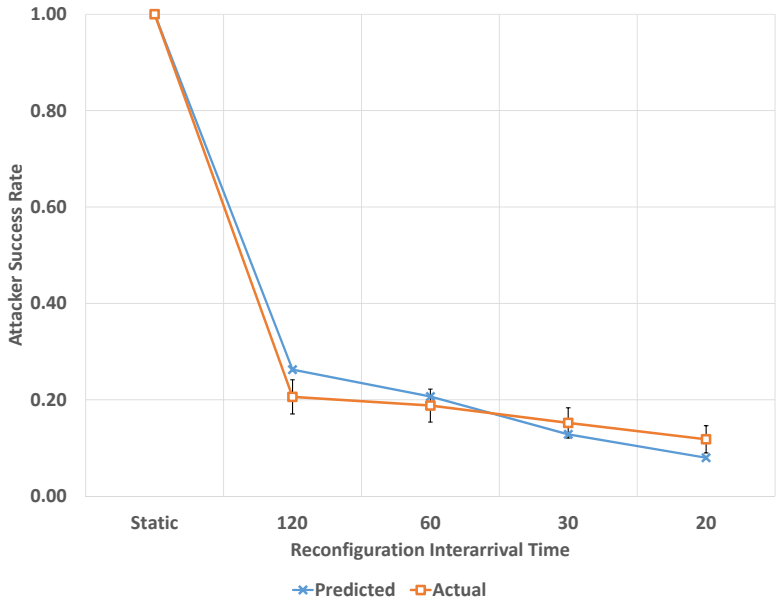


Figure 6.4: Probability of Attacker Success for Varying Service Reconfiguration Interarrival Rates

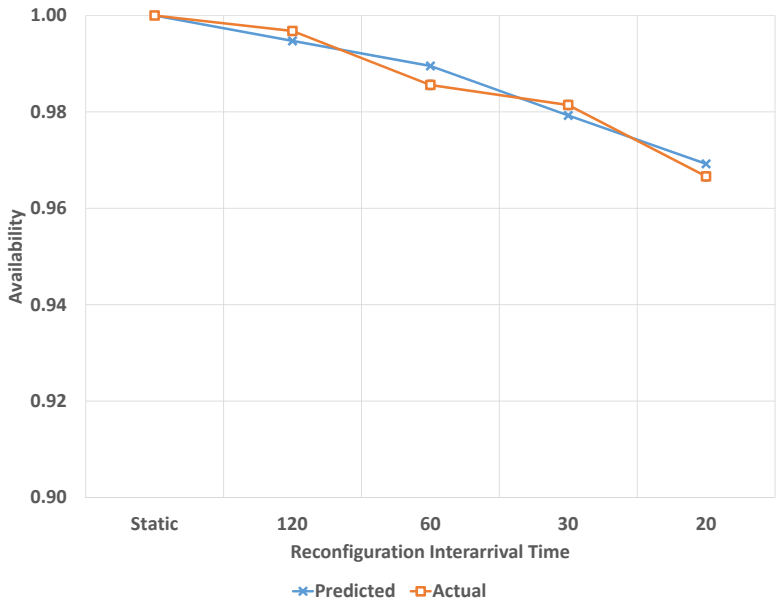


Figure 6.5: Availability for Varying Service Reconfiguration Interarrival Rates

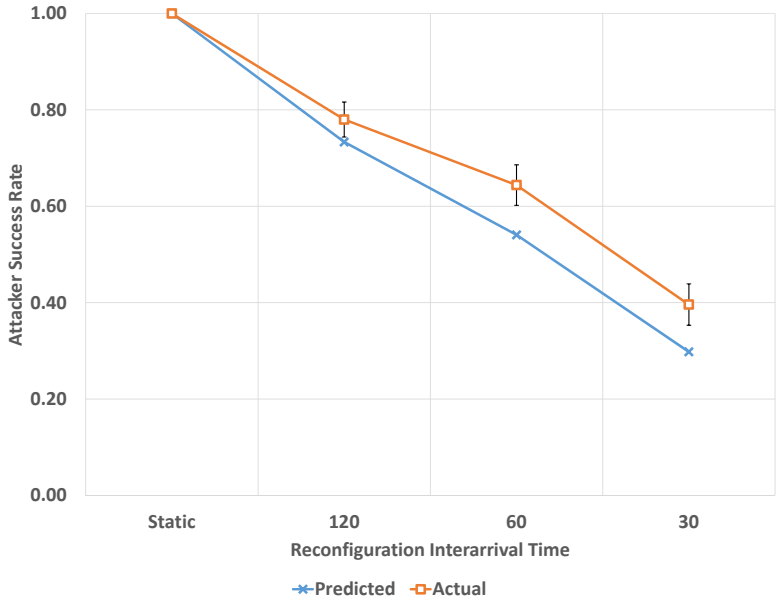


Figure 6.6: Probability of Attack Success for Varying IP Reconfiguration Interarrival Rates

Figure 6.7 shows the impact to availability when using IP reconfigurations. A much larger decrease in availability when using the IP randomization scheme is observed compared to service reconfiguration. This is due to the fact that as implemented, changing the IP address requires sending an external request for an IP address to the MTD controller and receiving a reply back. The connection to the monitor must also be rebuilt, which required a total of 9.59 seconds on average. This reduction in availability of 25% or more could mean that this method with higher reconfiguration rates may not be acceptable to users.

6.5.3 Combined Effects

The effects on attacker success rate for each combination of settings for the two MTDs is shown in Table 6.2. As service and IP reconfiguration rates increase, the attacker's success rate tends to decrease monotonically.

Using the measurements obtained for T_a, t_S, and t_{IP} during the experiments, results to the values predicted by the analysis are compared in Section 6.3, as seen in Table 6.3.

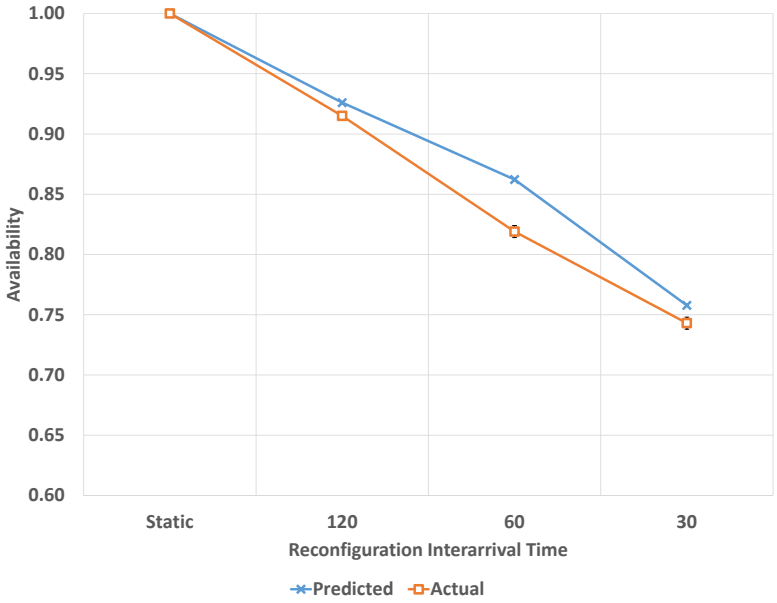


Figure 6.7: Availability for Varying IP Reconfiguration Interarrival Rates

The results are similar to the ones observed, with some larger errors present with lower interarrival times, but generally predict the behavior of the pair of MTDs.

236 1+0

Likewise, service availability for each combination of settings was measured and obtained similar results. As the interarrival time for IP reconfigurations decreases, the availability decreases. However, as the service reconfiguration interarrival time decreases and IP reconfiguration is held at a some constant rate, it was observed that the availability actually *increases* at particular values. For example, when IP reconfiguration interarrival time = 60 sec, as service reconfiguration interarrival time goes from 60 seconds to 30 seconds, availability increases from 0.782 to 0.832. This may be because the IP reconfigurations take so much longer relative to service reconfigurations and the two are mutually exclusive. This means that lengthy IP reconfigurations are delayed somewhat compared to service reconfigurations, resulting in the system behaving more similarly to that of service reconfiguration and that the two MTDs are still not wholly independent from one another.

Table 6.2: Attacker's Success Rate

		IP Reconfig			
		Static	120	60	30
Service Reconfig	Static	1.000 ± 0.000	0.780 ± 0.036	0.644 ± 0.042	0.396 ± 0.043
	120	0.206 ± 0.035	0.102 ± 0.027	0.074 ± 0.023	0.054 ± 0.020
	60	0.188 ± 0.034	0.086 ± 0.025	0.078 ± 0.024	0.046 ± 0.018
	30	0.152 ± 0.031	0.086 ± 0.025	0.056 ± 0.020	0.044 ± 0.018
	20	0.118 ± 0.028	0.034 ± 0.016	0.034 ± 0.016	0.022 ± 0.013

Table 6.3: Attacker's Success Rate (Predicted Values)

		IP Reconfig			
		Static	120	60	30
Service Reconfig	Static	1.000	0.733	0.541	0.298
	120	0.263	0.193	0.142	0.078
	60	0.207	0.152	0.112	0.062
	30	0.128	0.094	0.069	0.038
	20	0.080	0.058	0.043	0.024

6.5.4 MTD Protection Against Multiple Targets

The original set of experiments focused on a protecting a single target. However, most enterprises have multiple nodes that might require protection. This also matches more closely with a real-world example, as attackers probe a network and obtain a list of possible targets before attempting to probe further. With the virtual environment, multiple instances of MTD-protected nodes were created and results were analyzed.

The experiments were repeated using a total of six nodes in the virtual environment. Each node had identical MTD settings but ran independently. The attack script did a target scan against the entire network, followed by a more invasive scan to determine service and an attack on vulnerable target. Scans and attacks were performed sequentially on each node found during the initial scan to model an attacker not opening connections to multiple targets at the same time to maintain a stealthy presence on the network, with a total of 100 trials performed for each combination of settings due to the increased number of targets.

The effect of service reconfiguration were observed in Figure 6.8 which contains a series of histograms showing the number of times that a certain number of attacks were successful

Table 6.4: Availability

		IP Reconfig			
		Static	120	60	30
Service Reconfig	Static	1.000 ± 0.000	0.916 ± 0.003	0.819 ± 0.005	0.743 ± 0.005
	120	0.997 ± 0.000	0.909 ± 0.002	0.838 ± 0.003	0.692 ± 0.003
	60	0.986 ± 0.001	0.793 ± 0.003	0.782 ± 0.003	0.647 ± 0.003
	30	0.981 ± 0.001	0.692 ± 0.003	0.832 ± 0.003	0.711 ± 0.003
	20	0.967 ± 0.001	0.914 ± 0.002	0.794 ± 0.003	0.677 ± 0.003

Table 6.5: Availability (Predicted Values)

		IP Reconfig			
		Static	120	60	30
Service Reconfig	Static	1.000	0.926	0.862	0.758
	120	0.995	0.921	0.858	0.754
	60	0.990	0.916	0.853	0.750
	30	0.979	0.907	0.844	0.742
	20	0.969	0.898	0.836	0.734

for multiple service reconfiguration settings. For these values, all IP addresses remained static.

For example, with an service reconfiguration interarrival time of 120 seconds, a large number of trials where two or more out of six attacks were successful. However, as the interarrival time between service reconfigurations decreases, the histograms' distributions shift to the left. When the average interarrival time between reconfigurations is 20 seconds, the majority of trials resulted in zero or one out of the six available VMs compromised.

With multiple nodes, the criteria for success as the attacker or defender must be re-examined. It is assumed that an attack is considered a success if any nodes are able to be compromised. If attacks against each target are independent, the overall chance of attacker success $\hat{ps}$ would be 1, minus the probability that the individual attack with success rate ps on each of n different nodes all failed, or $\hat{ps} = 1 - (1 - ps)^n$. This constitutes an upper bound on attacker's success rate.

However, even more attacks may fail over time because the attacks are scripted to

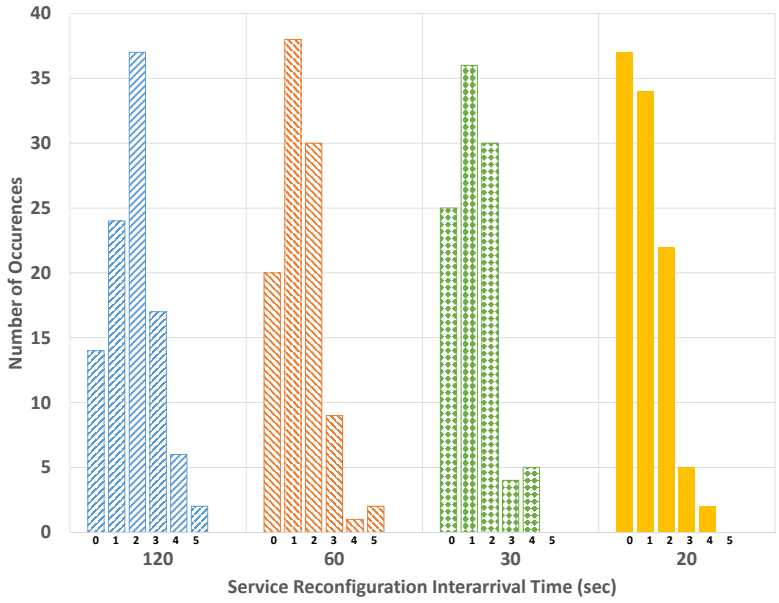


Figure 6.8: Histograms Showing Frequency of Numbers of Successful Attacks for Varying Service Reconfiguration Interarrival rates

be performed in a sequential manner. By the time a likely target is probed further and exploited, the higher the likelihood of it being reconfigured already. This is illustrated in Figure 6.9.

ps for the single target case is the likelihood a reconfiguration takes place within time period T_a; However, due to the sequential nature of the attacker probing and attempting attacks, the individual attack success probability ps_i against node i will vary due to the total time elapsed. Using the example in Figure 6.9 and Equation 6.7:

$$ps_i = e^{-T_{ai}(\lambda_S+\lambda_{IP})} \cdot \frac{s_v}{s} \cdot A \tag{6.8}$$

As T_{ai} increases on each subsequent vulnerable node found, ps_i approaches zero, as the node must remain unchanged for a longer period of time for an attack to be successful. The overall success rate $\hat{ps}$ is then:

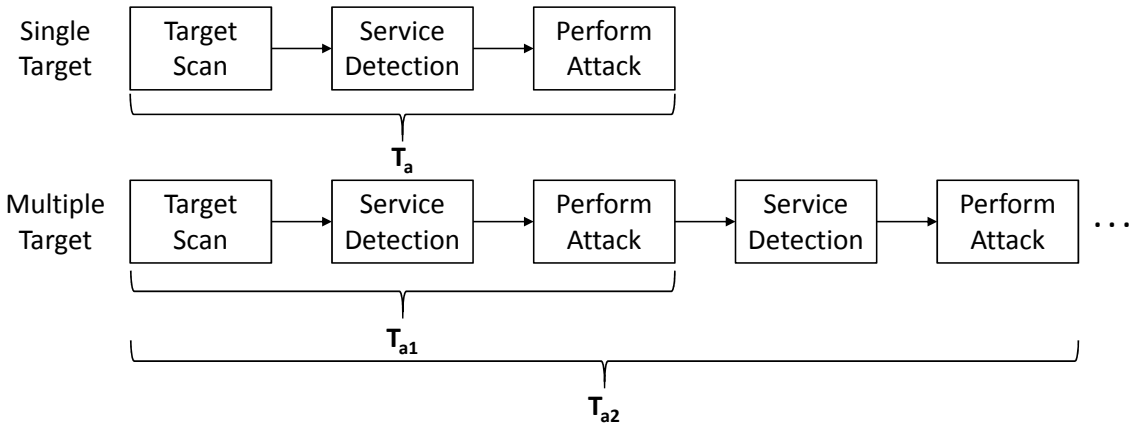


Figure 6.9: Timing Windows for Attacks on Multiple Targets

$$\hat{ps} = 1 - \left(\prod_{i=1}^{n} (1 - ps_i) \right) \tag{6.9}$$

which converges to a lower value than $1 - (1 - ps)^n$ as $ps_i \to 0$.

The complete observed results showing attacker's success rate are shown in Table 6.6.

Table 6.6: Attacker's Success Rate (Multiple Targets)

		IP Reconfiguration			
		Static	120	60	30
Service Reconfig	Static	1.00 ± 0.000	0.88 ± 0.064	0.61 ± 0.096	0.29 ± 0.093
	120	0.86 ± 0.068	0.64 ± 0.094	0.55 ± 0.098	0.22 ± 0.081
	60	0.80 ± 0.078	0.55 ± 0.098	0.43 ± 0.097	0.14 ± 0.068
	30	0.84 ± 0.072	0.55 ± 0.098	0.49 ± 0.098	0.15 ± 0.070
	20	0.63 ± 0.095	0.41 ± 0.096	0.29 ± 0.089	0.22 ± 0.081

6.5.5 Computing Utility

The analytic model presented in Sections 6.3 allows one to predict the attacker's success rate and availability. Using these results, one can answer questions such as "given the user's

objectives for security and availability, what combination of MTDs and settings maximize overall utility?"

This can be addressed by assigning utility values to the attacker's likelihood of success and availability using the following sigmoid functions:

$$U_P(ps) = \frac{e^{\sigma(-ps+\beta_P)}}{1 + e^{\sigma(-ps+\beta_P)}} \tag{6.10}$$

$$U_A(A) = \frac{1}{1 + e^{\sigma(-A+\beta_A)}} \tag{6.11}$$

where A is the availability, β_A is the availability objective, ps is the attacker's probability of success, β_P is the attacker's success probability objective, and σ is a steepness parameter for the sigmoid. Two different forms of the sigmoid function are used because a solution with optimal utility seeks to minimize the attacker's chance of success and maximize availability.

Based on the utility values derived security and availability, a global utility function, U_G, is computed as:

$$U_G = w_P \cdot U_P(ps) + w_A \cdot U_A(A) \tag{6.12}$$

where w_P and w_A are weight factors chosen such that $w_P + w_A = 1$. Different values of w_P and w_A influence the optimal choice of MTDs and reconfiguration rates. Table 6.7 shows utility values of all settings for values of $\beta_P = 0.2$, $\beta_A = 0.99$, $\sigma = 10$, $w_P = 0.25$, and $w_A = 0.75$. These values correspond to both the defender security risk appetite and tolerance for service disruption. For example, a defender which favors availability would also favor more time between reconfigurations to limit service disruption (and promote higher availability) as opposed to preventing attacks. These weight factors are assumed to be known to - or other otherwise can be determined by - defenders via means outside the scope of this work. The values chosen represent this high availability use case, reflecting many service providers, which are contractually-obligated to maintain service uptime at levels exceeding 99%.

Table 6.7: Utility Values

		IP Reconfig			
		Static	120	60	30
Service Reconfig	Static	0.394	0.241	0.118	0.089
	120	0.509	0.413	0.329	0.239
	60	0.500	0.290	0.293	0.236
	30	0.513	0.344	0.330	0.250
	20	0.505	0.344	0.303	0.245

Out of the settings evaluated, the optimal utility occurs with a service reconfiguration interarrival time is an average of 30 seconds and IP address reconfigurations is not employed at all. In the implementation, IP reconfiguration offered relatively little security benefit and relatively large loss of availability compared to the service reconfiguration technique. However, the results align with the conclusions of other researchers regarding network randomization [113], which also demonstrate relatively minor security benefits. Other pairings of MTDs might offer a more balanced result if their performance profiles were more comparable. In summary, a decision-maker presented with similar results may initiate efforts to implement service reconfiguration and also conclude that IP reconfiguration is not worth pursuing altogether; avoiding costly, but ineffective investments.

6.6 Summary

Individually, MTDs have demonstrated effectiveness against a variety of threats and attack vectors. Collectively, they offer potential of a more secure future. As one step in achieving that potential, this paper continues prior work in evaluation and performance modeling of MTDs and provides further contributions to the field. However, more research must be performed to fully realize the potential of MTDs.

The analytic work and implementation might be improved with a higher level of fidelity. More work could be done to understand specific interactions between the attacker and defender within the implementation to improve its accuracy. For example, edge cases and race conditions may exist between the attacker and defender that cause a reconfiguration

to be unsuccessful in preventing an attack.

Likewise, while the MTDs utilized would ideally be fully independent, the implementation does contain interactions between them. It is expected that other practical implementations would not be fully independent. Further work could involve development of a metric to measure the level of dependence between two or more implemented MTDs. Such a metric would allow decision makers to avoid combinations of MTDs that are highly dependent upon each other and may have undesirable interactions.

Conversely, the analysis and defender model can also be further generalized to apply to more MTDs. As different MTD techniques affect attackers in different phases of the Cyber Kill Chain, a MTD that prevents an attack earlier in the process might be more effective overall in preventing attacks. A MTD that takes effect later in the kill chain might instead provide defense by delaying the attack, ensuring the service remains protected long enough to accomplish its mission, or simply reducing the number of attempts an attacker is able to make against the system.

For example, while the attack model features an attacker which simulates nearly the entire Cyber Kill Chain, additional steps may be performed. The attacker established command and control on its target, but progressed no further. Realistic attackers leverage their control over compromised hosts over time to accomplish a range of objectives. These actions on objectives, such as data exfiltration, is already the focus of research [114]. Furthermore, there are known refresh MTD techniques [115] which may may be relatively ineffective at preventing initial compromise, but return compromised hosts to a safe state, theoretically mitigating the long-term impact of these attacks. Such an extension would likely require the development of additional attacker success metrics. As the goal of this research is determining the concurrent effectiveness of MTDs, evaluating the effectiveness of these varied MTD techniques, in combination, against the entire Cyber Kill Chain is a promising direction for future research.

Chapter 7: High-Fidelity Testing

In this section, practical underpinnings are provided to the framework by utilizing a high-fidelity cyber testbed environment to simulate attackers seeking to minimize activity on the network in order to maintain stealth.

7.1 Network Topology Discovery

It was assumed that, through privilege escalation, an attacker can gain root privileges on any compromised node, and execute local commands, including diagnostic commands, without alerting the defender. In fact, an attacker who has gained elevated privileges can potentially obfuscate local command execution or directly compromise the integrity of local logs. In contrast, the attacker seeks to minimize interactions with the network itself, as obfuscating malicious activity on the network would prove more difficult.

While some perturbation of the network is unavoidable (especially during compromise and any exfiltration activities), stealthy attackers seek to minimize this traffic. Many conventional tools and commands, such as `traceroute/tracert` and active scanning tools such as Nmap, generate network traffic. These tools are well-known and their usage could alert a vigilant defender employing a Network-Based Intrusion Detection System (NIDS). Although passive techniques might potentially limit the adversary's reconnaissance capabilities, this work demonstrates how a trove of information on compromised nodes may be used to map out parts of the network. Specifically, it is assumed that the attacker is able to access the following information:

- **Host Configuration**. Accessible through the `ipconfig` and `ifconfig` commands on Windows and Linux-based operating systems respectively, host configuration includes IP address, hostname, and gateway information.

- **Address Resolution Protocol (ARP) Cache**. The ARP cache maintains a mapping between network and physical addresses of hosts within the same Layer 2 broadcast domain.

- **Routing Tables**. Routing tables allow an attacker to discover new targets by looking at *destination* and *next hop* IP addresses.

There are many examples of advanced, stealthy malware gathering and transmitting similar information to the threat actor. As mentioned previously in Chapter 2, both Stuxnet [8] and Shamoon [116] attempt to transmit local host and network information, even though their primary objective is considered to be sabotage as opposed to espionage.

It is also assumed that the attacker is able to eavesdrop on network interfaces to collect, monitor, and capture packets which traverse compromised nodes. Using the information in the packet headers, the attacker is able to infer a network topology graph which represents its view of the network. Techniques for inferring network topologies through passive mapping with packet captures, and associated limitations, have been investigated by Hosmer [117] and Akande [118]. To address some of these limitations, and differently from previous work, in the attacker's model leverages additional sources of information, beyond network packets, to derive a more accurate and complete view of the target network.

7.1.1 Reward

Earlier chapters defined $reward(t_i, v)$, with $i \in [1, m]$ and $v \in V$, as the value gained by an adversary during the time interval $\Delta t_i = [t_{i-1}, t_i]$ by controlling node v. Accordingly, the total attacker's reward at time t_i can be computed as the sum of the rewards gained from all the nodes $v \in V^*$ in the malware's footprint, i.e.,

$$reward(t_i) = \sum_{v \in V^*} reward(t_i, v), \quad \forall i \in [1, m]. \tag{7.1}$$

Finally, the value accrued by an adversary over the entire time horizon $\mathcal{T}$ can be simply

computed as

$$Reward = \sum_{i=1}^{m} reward(t_i). \tag{7.2}$$

The reward function $reward : \mathcal{T} \times V \rightarrow \mathbb{R}^+$ is intended to capture the attacker's perceived value of each network asset $v \in V$. As different adversaries may have different objectives, one may consider $reward(t_i, v)$ as a normally distributed random variable with mean $\mu_{t_i,v}$ and standard deviation $\sigma_{t_i,v}$. In fact, the value of each network asset could be determined by averaging over multiple different reward functions, each representing a different attacker. Under such conditions, if this experiment is repeated many times, the central limit theorem implies that the distribution of the average will closely approximate a normal distribution. If one assumes independence between different network assets, then one can conclude that $reward(t_i)$ and $Reward$ are also normally distributed, i.e.,

$$reward(t_i) \sim N\left(\sum_{v \in V^*} \mu_{t_i,v}, \sum_{v \in V^*} \sigma^2_{t_i,v}\right) \tag{7.3}$$

$$Reward \sim N\left(\sum_{i=1}^{m} \sum_{v \in V^*} \mu_{t_i,v}, \sum_{i=1}^{m} \sum_{v \in V^*} \sigma^2_{t_i,v}\right) \tag{7.4}$$

In fact, the sum of two independent normally distributed random variables is normally distributed, with its mean being the sum of the two means, and its variance being the sum of the two variances.

7.1.2 Cost

As described in [101], detectability costs for individual nodes can be determined based on node characteristics that affect the level of attacker's effort required to compromise an asset and maintain it in the malware footprint, which in turn is directly related, as shown in [119],

to the risk of being detected.

$cost(t_i, v)$ is used to denote the cost for the attacker to compromise or maintain node v at time t_i. The total cost at time t_i can then be computed as

$$cost(t_i) = \sum_{v \in V^*} cost(t_i, v), \quad \forall i \in [1, m]. \tag{7.5}$$

where V^* is the set of nodes in the attacker's footprint.

Once a node v has been compromised at time t_v^*, the cost for the attacker to maintain v in its footprint during subsequent time intervals might be lower than the cost sustained for the initial compromise, due to less detectable activity on that node. Formally, the cost function *cost* for a node v compromised at time t_v^* can be defined as

$$cost(t_i, v) \begin{cases} = 0 & \text{if } t_i < t_v^* \\ = c_v & \text{if } t_i = t_v^* \\ < c_v & \text{if } t_i > t_v^* \end{cases} \tag{7.6}$$

where c_v is a constant representing the one-time cost sustained by the attacker to compromise node v.

Similarly to the prior discussion w.r.t. to the reward function, one may think of $cost(t_i, v)$ as a normally distributed random variable with mean $\mu_{t_i,v}$ and standard deviation $\sigma_{t_i,v}$. Assuming that the risk of the attacker being detected when compromising a network asset is independent of the risk of being detected when compromising other network assets, $cost(t_i)$ is also normally distributed, i.e.,

$$cost(t_i) \sim N\left(\sum_{v \in V^*} \mu_{t_i,v}, \sum_{v \in V^*} \sigma_{t_i,v}^2\right). \tag{7.7}$$

The independence assumption is particularly valid when intrusion detection is primarily host-based. When detection relies on a combination of host-based and network-based

mechanisms, such assumption may not strictly hold and Equation 7.7 would provide a rough approximation of the cost. Due to the sophisticated nature of the adversaries considered here, it is reasonable to assume that they are always successful in compromising a target node, if they choose to do so. Advanced threat actors [2] maintain stockpiles of exploits such that when a given exploit becomes publicly known and it is rendered ineffective, another exploit is used instead.

7.1.3 Attacker's Dynamics

It is assumed that the attacker maintains a view of the target network's topology and services running on each node, and updates this view as more information is gathered through probing or traffic analysis. A typical attack by an APT actor may iteratively unfold as follows:

- **Reconnaissance**. At a time t_i, starting from previously compromised nodes, the adversary analyzes data passively gathered from the target network (as described in detail in Section 7.1) and identifies a set $S \subseteq V$ of newly reachable assets.

- **Target evaluation**. For each $v \in S$, the adversary computes $reward(t_i, v)$ and $cost(t_i, v)$.

- **Target selection**. The adversary selects a subset $S^{'} \subseteq S$, such that $\sum_{v \in S^{'}} cost(t_i, v) \leq B$ and one of several possible objective functions is optimized (e.g., maximize reward or reward-to-cost ratio).

- **Compromise**. The adversary compromises nodes in $S^{'}$ and adds them to its footprint.

The steps discussed above broadly correspond to the first five steps of the cyber kill chain, i.e., reconnaissance, weaponization, delivery, exploitation, and installation [105]. A possible deceptive defensive strategy building upon this model could aim at creating and

Algorithm 4 $genNodeView(v, pcap_v, routes_v, domain_v)$

v is a node, $pcap_v$ the pcap capture from v, $routes_v$ are the network routes of v, $domain_v$ is the set of IP addresses belonging to nodes within the broadcast domain of v
Graph $G_v = (V, E)$ representing the attacker's view of the network from node v
▷ Initialize
$G_v \leftarrow (\{v\}, \emptyset)$
▷ Add nodes to graph from domain
$V \leftarrow V \cup nodesFromIPs(domain_v)$
▷ Extract packets
$p \leftarrow getNextPacket(pcap_v)$
while $p \neq \emptyset$ **do** ▷ Extract source IP (src), destination IP (dst), and packet length (vol)
    $src_p, dst_p, vol_p \leftarrow extractPacketData(p)$
    $v_{src} \leftarrow nodesFromIPs(src_p)$
    $v_{dst} \leftarrow nodesFromIPs(dst_p)$
    if $v_{src} \notin (V)$ **then**
        $V \leftarrow V \cup v_{src}$
    end if
    if $v_{dst} \notin (V)$ **then**
        $V \leftarrow V \cup v_{dst}$
        $v_{ip} \leftarrow getIPs(v)$
    else if $(src_p \in v_{ip} \wedge dst_p \in domain_v) \bigvee (dst_p \in v_{ip} \wedge src_p \in domain_v)$ **then** ▷ v and the packet src/dst are within the same domain, add edge between v and the src/dst
        $addEdge(G_v, src_p, dst_p, vol_p)$
    else ▷ packet src or dst are not within the same domain as v, create edges through next hop address
        if $dst_p \neq v_{ip}$ **then**
            $nextHop \leftarrow getNextHop(routes_v, dst_p)$
            $addEdge(G_v, v, nextHop, vol_p)$
            $addEdge(G_v, nextHop, dst_p, vol_p)$
        end if
        if $src_p \neq v_{ip}$ **then**
            $nextHop \leftarrow getNextHop(routes_v, src_p)$
            $addEdge(G_v, src_p, nextHop, vol_p)$
            $addEdge(G_v, nextHop, v, vol_p)$
        end if
    end if
    $p \leftarrow getNextPacket(pcap_v)$
end while
return G_v

deploying honeypots with characteristics such that the likelihood of them being included in the set S' is higher than for real network assets.

Algorithm 4 generates a partial view of the network from a node v, whereas Algorithm 5 updates the attacker's view of the entire network as new information becomes available by executing Algorithm 4 over newly compromised nodes.

7.2 Estimating Reward and Cost

This work is intended to provide a framework to reason about a stealthy attacker's reconnaissance behavior, incentives and deterrents. As such, the specific $reward(t_i, v)$ and

Algorithm 5 *updateGraph*(G, v_c, G_c)

Graph $G = (V, E)$ the attacker's existing model of the target network, node v_c the newly compromised node, Graph $G_c = (V_c, E_c)$ the attacker's view of the network from v_c
Graph G'
▷ nodes the attacker discovers after compromising v_c (not already in G)
$newNodes \leftarrow V_c - V$
$V \leftarrow V \cup V_c$
▷ replace edges in G which include v_c
for $e \in E$ **do**
    if $v_c \in e$ **then**
        $removeEdge(e)$
    end if
end for
for $e_c \in E_c$ **do** ▷ Extract edge source (src), destination (dst), and packet length (vol)
    $src_e, dst_e, vol_e \leftarrow extractEdgeData(e_c)$
    ▷ Add edge in G if edge source/destination is v_c or the source/destination is a "new node"
    if $src_e == v_c \bigvee dst_e == v_c \bigvee src_e \in newNodes \bigvee dst_e \in newNodes$ **then**
        $addEdge(G, src_e, dst_e, vol_e)$
    end if
end for

$cost(t_i, v)$ calculation for individual nodes may differ across organizations and operational contexts. With this caveat, the next section provides a practical estimation of reward and cost for both attackers and defenders for the purpose of this work.

Estimating Reward. For the purposes of this work, the attacker determines the reward of a node as a function of the observed volume of traffic originating from or destined to that node. Ground truth reward values would likely be determined on an organizational level through a comprehensive risk assessment. In lieu of such a risk assessment, this work employs a method similar to the one employed by the attacker – based on observed traffic in a simulated environment and other reconnaissance which minimizes network interaction. However, while the attacker calculates values of nodes within its footprint, the defender can calculate values $\forall v \in V$. Furthermore, the defender can calculate reward values based on all legitimate traffic. In contrast, the attacker utilizes only traffic observed on nodes within its footprint.

Estimating Cost. For the purpose of this work, the relationship between detectability cost and the notion of attack surface, which has been formally defined by Manadhata *et al.* [120], is considered. Systems with a greater attack surface provide more potential attack vectors to exploit the system. While a thorough, formal attack surface determination is infeasible for attackers, Manadhata's work also discusses alternate methodologies to determine attack

surface. One option is to count the number of vulnerabilities found in the system's software (including operating systems) according to public vulnerability databases. Though an approximation of a system's security posture, an attacker could be likely to employ a similar approach due to expediency.

Let be $CVE(t_i, v)$ denote the set of vulnerabilities on v at time t_i published in the MITRE CVE database. A *vulnerability detectability score* is computed, $vd : CVE \rightarrow \mathbb{R}^+$, using the CVSS *Exploitability* score and the set of public Intrusion Detection System (IDS) rules associated with the vulnerability, denoted as $IDS(cve)$. Individual organizations may employ one or more IDS according to organizational preference. For the purposes of this work, both SNORT and Suricata, two well-known, open source network intrusion detection systems that publish open repositories of publicly available rules are employed. $vd(cve)$ is defined as follows:

$$vd(cve) = \frac{|IDS(cve)|}{Exploitability(cve)}. \tag{7.8}$$

Intuitively, the more IDS rules available for a given vulnerability, the easier it is for the attacker to be detected when attempting to exploit that vulnerability. Additionally, a higher CVE Exploitability score for a given vulnerability indicates that attackers attackers are better able to utilize the vulnerability in a stealthy manner. This is due to how the CVE Exploitability score is calculated, which incorporates several subscores linked to detectability. For example, the Attack Vector (AV) subscore measures how easily an attacker can leverage a vulnerability (e.g., remote exploit possible vs. physical access required). Thus, an attacker leveraging a vulnerability with a higher Exploitability score will have more options (more attack vectors per Manadhata), and will select a vector which the attacker believes to offer the highest chance of remaining undetected.

Based on the definition above, detectability cost associated with a node v is defined as either the average detectability cost across all vulnerabilities on v, i.e.,

$$cost(t_i, v) = \frac{\sum_{cve \in CVE(t_i,v)} vd(cve)}{|CVE(t_i, v)|} \tag{7.9}$$

or as the minimum detectability cost of all the vulnerabilities on v, i.e.,

$$cost(t_i, v) = \min_{cve \in CVE(t_i,v)} vd(cve) \tag{7.10}$$

To leverage this approach, the attacker must be able to identify software operating on potential targets. An attacker may accomplish this objective through passively collecting and processing packet captures. There exist a number of tools, such as the publicly-available pOf, which are capable of performing OS and service identification based on packet capture.

7.3 Evaluation

In this section, the evaluation of the proposed framework is described. This includes both a description of the experimental setup; as well as a simple example in Section 7.3.2 which illustrates how a stealthy attacker views the network we aim to defend. Finally, we report on numerical results over larger networks in Section 7.3.3.

7.3.1 Experimental Setup

The experimental setup consists of simulated enterprise environments hosted on the CyberVAN testbed [121], which was designed to support high-fidelity cyber security research. In a CyberVAN scenario, the end hosts (such as workstations and servers) are realized through VMs running real applications while the network elements (such as routers and switches) are simulated using NS3. Experiment scenarios may run an exact replica of the target network by using an identical set of OS VMs. Human users interacting with the

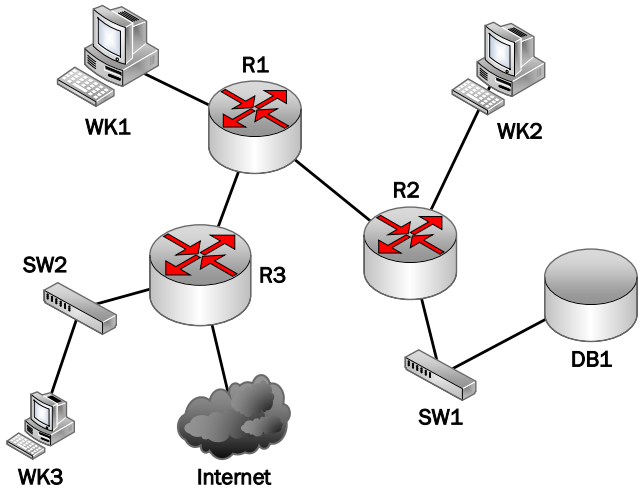
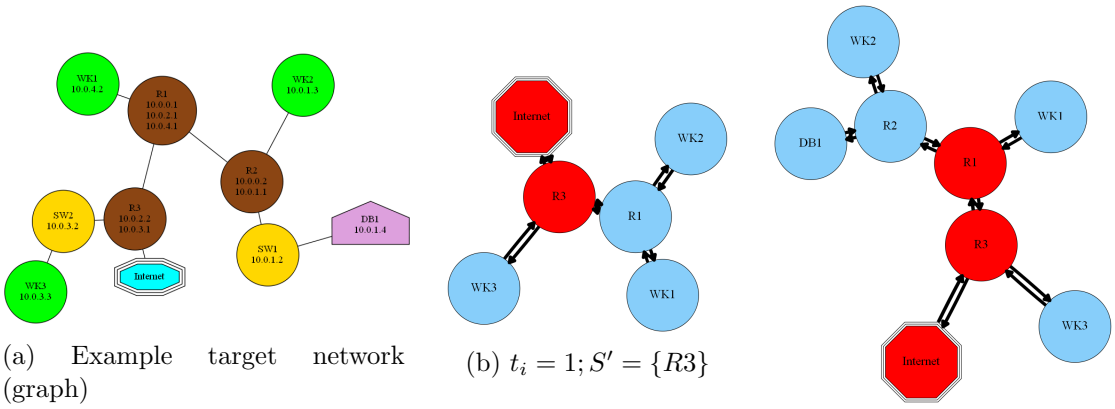


Figure 7.1: Example target network

applications on VMs are modeled by a tool called ConsoleUser, which mimics human user activities by controlling mouse and keyboard input. A sequence of user actions, such as typing a URL in a browser followed by clicking a link on the returned web page, will result in the generation of realistic network traffic. Both CyberVAN and ConsoleUser have been employed extensively for the purpose of published cyber research [122–124]. In this work,



(a) Example target network (graph)

(b) $t_i = 1; S' = \{R3\}$

(c) $t_i = 2; S' = \{R3, R1\}$

Figure 7.2: Progression of network topology discovery

ConsoleUser is also leveraged to simulate human users performing various tasks.

There is a lack of existing models that capture the connectivity of an enterprise network at both layer 2 and layer 3. Therefore, in order to generate different network topologies, this work employed synthesized enterprise network topologies of different sizes with scale-free networks properties. Such networks are known to accurately capture the connectivity in ISP networks at router level [97]. The NetworkX 2.4 library was utilized to generate these networks in an incremental fashion, using the Holme and Kim algorithm, a variant of the Barabási-Albert (BA) model. In the BA model, new nodes are added to the network one at a time. Each new node is connected to an existing node with a probability proportional to the node's current degree. Therefore, new nodes added to the network has a higher probability of becoming the neighbor of existing nodes which already have a higher degree, relative to other nodes in the network. The Holme and Kim BA variant tends to generate networks with more clusters than typical BA networks, which is intended to more closely model enterprise networks.

The networks generated and simulated in CyberVAN for the purposes of this work consist of user workstations (where ConsoleUser operates), databases, switches, routers and the external internet. Nodes with a degree greater than 1 are designated as network elements (routers/switches). The betweenness centrality of these nodes was computed. Half with higher centrality were assigned as routers and the half with lower centrality as switches to better approximate enterprise network structure. Network practices dictate that the internal network be structurally separated as much as possible from the external internet to promote defense-in-depth security, particularly with sensitive/high-value network resources. To reflect this practice, the router with the lowest centrality is designated as the internet router and the external internet is represented as a leaf node of this router. Then the longest paths were computed from the internet node. The leaf node with the longest path to the internet node is designated as a database node. Thus, this structure approximates the aforementioned network design principles to shield and isolate sensitive targets (such as a critical database) from potential external attack and offers a higher probability of detecting

exfiltration activity which must traverse a greater proportion of the overall network. Other leaf nodes are designated as user workstations. These user workstations employ ConsoleUser to communicate with either the internet node or the database and generate packet traffic accordingly.

Attackers originate from the internet node and employ the iterative decision and operating model described in Section 7.1.3 to compromise nodes in the network, building a footprint over time. The process an adversary can adopt to build a view of the network based on information collected on a single compromised node v – including captured traffic and information from other sources discussed in Section 7.1 – is described by Algorithm 4. However, utilizing any single compromised node limits the capability of the adversary to infer a topology for non-trivial networks. Thus, if multiple nodes have been compromised, the adversary aggregates data from all nodes within its footprint into an overall network topology, as described by Algorithm 5. The attacker leverages this view of the network as a primary input for the decision making process, determining which nodes to target in successive rounds. Next, a simple example was provided to illustrate both the networks generated and the the resulting progression of the attacker's network view.

7.3.2 Simple Example

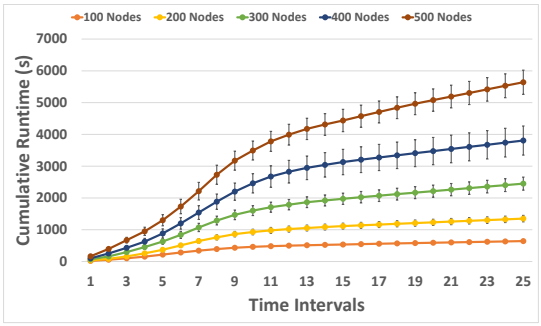
Fig. 7.2a depicts a simple network generated with the methodology described in Section 7.3.1, with routers (R1, R2, R3), switches (SW1, SW2), user workstations (WK1, WK2, WK3) and a database (DB1). Fig. 7.2b and Fig. 7.2c represent the progression of the attacker-inferred network topology, as generated by the algorithm. At the beginning, in Fig. 7.2b, the attacker, originating from the internet, initially compromises the internet router/firewall (R3). In this example, the attacker is able to reason about nodes which are not directly connected to the compromised node by understanding networking relationships. For example, while R3 is not directly connected to WK1 and WK2, the attacker is able to infer that both nodes reside in the network beyond R1, assuming that the attacker has access to both the pcap data from R3 as well as R3's network configuration, including routing

tables. As the adversary progresses through the network – thus adding more compromised nodes from which to gather data – a more complete and accurate view of network topology can be developed. Fig. 7.2c depicts how the attacker's view evolves after compromising R1. Of note, after compromising R1, the attacker becomes aware that WK2 resides behind a previously-unknown networking element, router R2. Therefore, the attacker adjusts its model accordingly. Additionally, through traffic traversing R1 from WK1 to DB1, the attacker discovers the existence of the internal database, DB1. This traffic from WK1 to DB1 would not typically traverse R3, as R3 does not lie on the shortest path from WK1 to DB1. This illustrates the importance for the attacker to compose its view from multiple sources. Furthermore, the attacker's view of the network is complete (with the exception of switches, which act as simple repeaters in CyberVAN simulations), even with a small proportion of the overall network being compromised.

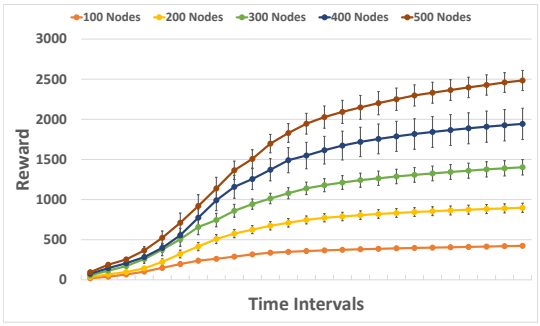
7.3.3 Experimental Results

In this section, the performance of the algorithm was examined for varying network sizes. Synthetic network topologies of different sizes (100, 200, 300, 400 and 500 nodes) were generated by scaling up smaller, real network topologies. For each network size, 30 different network topologies were generated and the results were averaged over different network settings with a 95% confidence interval. Fig. 7.3a shows the processing time for building a view of the network over 25 time intervals. When the cumulative processing time is smaller than the time horizon, the algorithm's performance can be considered real-time. For instance, the cumulative runtime for a network of 100 nodes is less than 500 seconds, or 20 seconds per time interval on average. Due to the nature of stealthy attackers, a time interval is likely to be significantly longer than the 20 seconds needed to update the view of the network and select the next target.

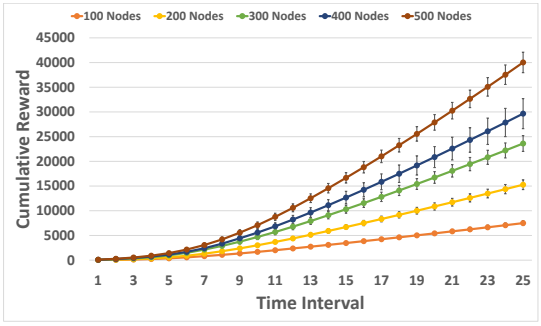
Fig. 7.3a shows the reward the attacker gains during each time interval. As mentioned previously, a number of methodologies may be use to estimate reward values, and they



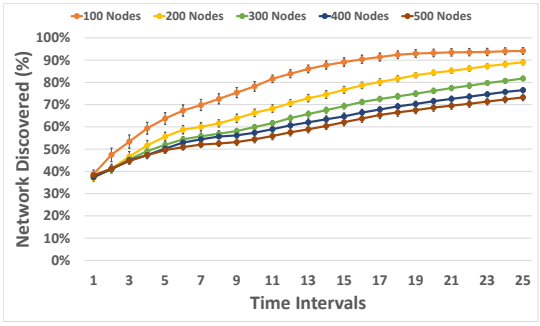
(a) Runtime



(b) Reward



(c) Cumulative Reward



(d) Percentage of network nodes discovered

Figure 7.3: Simulation results

would differ depending on the characteristics of the target network and the potential goals of the attacker. For the purposes of the work and without loss of generality, the reward of a node is assumed to be proportional to the amount of traffic observed through that node, and normalized with respect to the node with the smallest amount of traffic across all network sizes considered in the experiments. Thus, higher reward values correspond to increased exposure of target network information to the attacker. Fig. 7.3c shows the cumulative results over the time interval.

Finally, Fig. 7.3d shows how the percentage of nodes discovered by the adversary increases over time. For the purpose of this simulation and without loss of generality, a cost function was assumed in accordance with Section 7.1.2 calculated using the CyberVAN device specifications. Costs were normalized with respect to workstations, which are the most insecure. Taking this into account, databases and routers have a computed cost of

1.036924976 and 3.417790698, respectively. The chart shows that, after a number of iterations equal to 5% of the number of nodes, the attacker has discovered between 65 and 75% of the target network.

7.4 Summary

This chapter grounds the work of the previous chapters using the high-fidelity CyberVAN testbed as a proving ground, transitioning theory to practice. The work in this chapter served as a critical proof-of-concept for an attacker leveraging passive techniques to nonetheless effectively recon the network. APTs, which prioritize stealth, may use similar techniques as part of their standard operating procedure to evade detection. Understanding how this may be accomplished allows the development of defenses to interfere with these processes and hamper the overall efforts of the APT.

Chapter 8: Conclusions and Future Work

"If you know your enemies and know yourself, you will not be imperiled in a hundred battles" - Sun Tzu

8.1 Conclusions

APTs represent the pinnacle of cyber threat. In recent years, the tactics and very tools they pioneered have been copied and utilized by other entities, becoming more prevalent. Defense against these attackers thus is no longer the concern of nation-states alone, but also of businesses and other organizations which impact the daily lives of the public. Nation-states, even with all their resources, have proven to be vulnerable as well. When nation-states are affected, the flow of history shifts. Therefore, developing defenses against these threats is a critical and widespread concern. In order to develop such defenses, one must better understand the adversary.

This dissertation defines a quantitative framework to accomplish this goal of understanding the adversary - including incentives, deterrents and other factors which may influence their behavior. This framework challenges the notion in previous research which assumes that attackers are primarily focused upon relatively few "crown jewel" nodes in the network. Instead, the attacker is able to accrue some value from compromising nearly any node within the network. Coupled with the ability of these APTs to leverage vast resources and deep expertise to reconnoiter organizations and persist within their networks over time, effective defense requires new approaches and techniques.

In recent years, Adaptive Cyber Defense (ACD) and Moving Target Defense (MTD) techniques have emerged as potentially paradigm-shifting advances in cybersecurity. Thus

far, the cyber domain has been an equalizer among traditional powers, but favoring attackers [1]. ACD/MTD offer potential to shift this paradigm in favor of the defender. Thus, researchers have devoted significant effort in both developing ACD/MTD techniques [38][40][41][42][125] as well as understanding and quantifying them [126]. ACD/MTD techniques are particularly well-suited to defend against APTs by shifting configurations over time - rendering advanced reconnaissance ineffective and refreshing systems should a compromise occur - mitigating the ability to persistently perform actions-on-objectives. This work also provides notional examples of how such dynamic defenses may operate to deter the APT threat.

The quantitative framework allows defenders to (i) assess the cost incurred by APT actors to compromise and persist within a target system; (ii) estimate the value they gain by persisting in the system over time; (iii) simulate how the footprint of an APT evolves over time while constrained by limits on detectable activity to maintain stealth. Using this framework, defenders can leverage their understanding of their individual networks to make informed decisions, selecting appropriate MTDs and configuring them accordingly to provide the most effective tradeoffs to address threats.

8.2 Future Work

This work is intended to serve as a foundation upon which future defenses can be built. Although it presents a significant step towards mitigating the threat of APTs, more work is required. However, the foundation built is broad and flexible enough to be developed in numerous approaches. Several such approaches are immediately evident:

- **Refine variables and relationships:** As a quantitative framework, the conclusions provided are sensitive and dependent on input values for a variety of factors, including accurate valuations of reward and cost. Future work may refine the values to more accurately reflect real-world determinations and relationships. While this work has already been accomplished for this dissertation regarding testing and experimentation

within high-fidelity cybersecurity testbeds, more can be done to refine the framework within environments which mimic real-world scenarios as closely as possible.

- **Integrate framework with established ACD/MTD techniques:** As mentioned previously, there are now numerous ACD/MTD techniques available. As shown in Chapter 6, the effectiveness of these MTDs can be improved by adjusting a variety of parameters. A full integration of an existing ACD/MTD techniques with this quantitative framework would be a considerable next step. Developing an entirely new defense based on ACD/MTD principles which leverages this framework would be an even greater achievement.

- **Model to capture additional game-theoretic and adversarial considerations:** Cybersecurity research with game-theoretic considerations between attackers and defenders is both an established and growing area of research. In particular, this work may find commonality with concepts such as Stackelberg games.

This foundation offers a platform upon which a defense can be erected to withstand the threat posed by APTs. However, the final test remains deployment in the real world, against actual APT malware and the APT actors which employ them.

Bibliography

[1] M. R. DeVore and S. Lee, "APT (advanced persistent threat) s and influence: Cyber weapons and the changing calculus of conflict," *The Journal of East Asian Affairs*, pp. 39–64, 2017.

[2] A. Lemay, J. Calvet, F. Menet, and J. M. Fernandez, "Survey of publicly available reports on advanced persistent threat actors," *Computers & Security*, vol. 72, pp. 26–59, 2018.

[3] R. Kissel, "Glossary of key information security terms," *NIST Interagency Reports NIST IR*, vol. 7298, no. 3, 2013.

[4] M. I. Center, "Apt1: Exposing one of chinas cyber espionage units," FireEye Mandiant, Tech. Rep., February 2013.

[5] "2020 state of malware," Malwarebytes Labs, Tech. Rep., February 2020.

[6] "2019 cost of a data breach study," Ponemon Institute, IBM Security, Tech. Rep., June 2019.

[7] "Targeted cyberattacks logbook," 2018, [Online; accessed 5-March-2018]. [Online]. Available: https://apt.securelist.com/#!/threats/

[8] R. Langner, "To kill a centrifuge : a technical analysis of what stuxnets creators tried to achieve," Tech. Rep., 2013.

[9] L. Ablon and A. Bogart, *Zero days, thousands of nights: The life and times of zero-day vulnerabilities and their exploits.* Rand Corporation, 2017.

[10] T. Kristensen, "Flash 0-days," April 2011. [Online]. Available: http://blogs.flexerasoftware.com/vulnerability-management/2011/04/flash-0-days.html

[11] RFSID, "New kit, same player: Top 10 vulnerabilities used by exploit kits in 2016," December 2016. [Online]. Available: https://www.recordedfuture.com/top-vulnerabilities-2016/

[12] B. Research, "Java vulnerabilities: Write once, pwn anywhere," Bit9, Tech. Rep., August 2013.

[13] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on.* IEEE, 2010, pp. 297–300.

[14] J. A. Marpaung, M. Sain, and H.-J. Lee, "Survey on malware evasion techniques: State of the art and challenges," in *Advanced Communication Technology (ICACT), 2012 14th International Conference on*. IEEE, 2012, pp. 744–749.

[15] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proceedings of the First ACM Workshop on Moving Target Defense*. ACM, 2014, pp. 69–78.

[16] Mandiant, "M-trends 2017: a view from the front lines," FireEye, Tech. Rep., March 2017.

[17] J. Hunker, B. Hutchinson, and J. Margulies, "Role and challenges for sufficient cyber-attack attribution," *Institute for Information Infrastructure Protection*, pp. 5–10, 2008.

[18] E. M. Mudrinich, "Cyber 3.0: The department of defense strategy for operating in cyberspace and the attribution problem," *AFL Rev.*, vol. 68, p. 167, 2012.

[19] N. Tsagourias, "Cyber attacks, self-defence and the problem of attribution," *Journal of Conflict and Security Law*, vol. 17, no. 2, pp. 229–244, 2012.

[20] S. M, *Reverse Deception*. McGraw-Hill Education, July 2017.

[21] B. Schneier, "The cia's "development tradecraft dos and don'ts"," March 2017. [Online]. Available: https://www.schneier.com/blog/archives/2017/03/the_cias_develo.html

[22] S. Gallagher, "Helpful(?) coding tips from the cias school of hacks," March 2017. [Online]. Available: https://arstechnica.com/information-technology/2017/03/malware-101-the-cias-dos-and-donts-for-tool-developers/

[23] K. Ziolkowski, "Stuxnet - legal considerations," Tech. Rep., 2012.

[24] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," Tech. Rep. 6, 2011.

[25] G. McDonald, L. O. Murchu, S. Doherty, and E. Chien, "Stuxnet 0.5 the missing link," Tech. Rep., 2013.

[26] M. De Falco, "Stuxnet facts report : a technical and strategic analysis," Tech. Rep., 2012.

[27] N. Virvilis and D. Gritzalis, "The big four-what we did wrong in advanced persistent threat detection?" in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. IEEE, 2013, pp. 248–254.

[28] D. Sanger, *Confront and Conceal: Obama's Secret Wars and Surprising Use of American Power*. Crown Publishing, 2014.

[29] R. Langner, "Stuxnet's secret twin," Nov 2013. [Online]. Available: http://foreignpolicy.com/2013/11/19/stuxnets-secret-twin/

[30] T. M. Chen and A.-N. Saeed, "Lessons from stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[31] C. Raiu, "The day the stuxnet died," June 2012. [Online]. Available: https://securelist.com/the-day-the-stuxnet-died-27/33206/

[32] D. Goodlin, "A rash of invisible, fileless malware is infecting banks around the globe," Feb 2017. [Online]. Available: https://arstechnica.com/information-technology/2017/02/a-rash-of-invisible-fileless-malware-is-infecting-banks-around-the-globe/

[33] B. Schneier, "Duqu malware techniques used by cybercriminals," Feb 2017. [Online]. Available: https://www.schneier.com/blog/archives/2017/02/duqu_malware_te.html

[34] S. S. Response, "W32. duqu: the precursor to the next stuxnet," Tech. Rep. 6, November 2011.

[35] F. Hacquebord, "Two years of pawn storm: examining an increasingly relevant threat," TrendLabs Forward-Looking Threat Research (FTR) Team, Tech. Rep., 2017.

[36] ESET, "Dissection of sednit espionage group," Tech. Rep., 2016. [Online]. Available: https://www.eset.com/int/about/newsroom/research/dissection-of-sednit-espionage-group/

[37] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[38] H. Okhravi, M. Rabe, T. Mayberry, W. Leonard, T. Hobson, D. Bigelow, and W. Streilein, "Survey of cyber moving target techniques," DTIC Document, Tech. Rep., 2013.

[39] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding focus in the blur of moving-target techniques," *Security & Privacy, IEEE*, vol. 12, no. 2, pp. 16–26, 2014.

[40] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, "Moving target defense techniques: A survey," *Security and Communication Networks*, vol. 2018, 2018.

[41] J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: An architectural perspective," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, 2019.

[42] S. Sengupta, A. Chowdhary, A. Sabur, D. Huang, A. Alshamrani, and S. Kambhampati, "A survey of moving target defenses for network security," *arXiv preprint arXiv:1905.00964*, 2019.

[43] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, 2020.

[44] S. Jajodia, A. K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang, "Moving target defense ii," *Application of Game Theory and Adversarial Modeling. Series: Advances in Information Security*, vol. 100, p. 203, 2013.

[45] R. Colbaugh and K. Glass, "Moving target defense for adaptive adversaries," in *Intelligence and Security Informatics (ISI), 2013 IEEE International Conference on*, June 2013, pp. 50–55.

[46] M. I. Husain, K. Courtright, and R. Sridhar, "Lightweight reconfigurable encryption architecture for moving target defense," in *Military Communications Conference, MILCOM 2013 - 2013 IEEE*, Nov 2013, pp. 214–219.

[47] Y. Li, R. Dai, and J. Zhang, "Morphing communications of cyber-physical systems towards moving-target defense," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 592–598.

[48] V. Casola, A. D. Benedictis, and M. Albanese, "A moving target defense approach for protecting resource-constrained distributed devices," in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, Aug 2013, pp. 22–29.

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

[50] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 298–307.

[51] S. Bhatkar, D. C. DuVarney, and R. Sekar, "Efficient techniques for comprehensive protection from memory error exploits." in *Usenix Security*, 2005.

[52] L. V. Davi, A. Dmitrienko, S. Nürnberger, and A.-R. Sadeghi, "Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, pp. 299–310. [Online]. Available: http://doi.acm.org/10.1145/2484313.2484351

[53] E. D. Berger and B. G. Zorn, "Diehard: Probabilistic memory safety for unsafe languages," in *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '06. New York, NY, USA: ACM, 2006, pp. 158–168. [Online]. Available: http://doi.acm.org/10.1145/1133981.1134000

[54] G. Novark and E. D. Berger, "Dieharder: Securing the heap," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 573–584. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866371

[55] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 272–280. [Online]. Available: http://doi.acm.org/10.1145/948109.948146

[56] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, and E. Kirda, "G-free: Defeating return-oriented programming through gadget-less binaries," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 49–58. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920269

[57] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 281–289. [Online]. Available: http://doi.acm.org/10.1145/948109.948147

[58] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanović, "Randomized instruction set emulation," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 1, pp. 3–40, Feb. 2005. [Online]. Available: http://doi.acm.org/10.1145/1053283.1053286

[59] M. Thompson, N. Evans, and V. Kisekka, "Multiple os rotational environment an implemented moving target defense," in *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*, Aug 2014, pp. 1–6.

[60] R. Zhuang, S. Zhang, A. Bardas, S. A. DeLoach, X. Ou, and A. Singhal, "Investigating the application of moving target defenses to network security," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 162–169.

[61] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: A secretless framework for security through diversity," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley, CA, USA: USENIX Association, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267336.1267344

[62] H. Okhravi, A. Comella, E. Robinson, and J. Haines, "Creating a cyber moving target for critical infrastructure applications using platform diversity," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 1, pp. 30–39, 2012.

[63] D. Arsenault, A. Sood, and Y. Huang, "Secure, resilient computing clusters: Self-cleansing intrusion tolerance with hardware enforced security (scit/hes)," in *Proceedings of the The Second International Conference on Availability, Reliability and Security*, ser. ARES '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 343–350. [Online]. Available: http://dx.doi.org/10.1109/ARES.2007.134

[64] A. K. Bangalore and A. K. Sood, "Securing web servers using self cleansing intrusion tolerance (scit)," in *Dependability, 2009. DEPEND '09. Second International Conference on*, June 2009, pp. 60–65.

[65] Y. Huang, D. Arsenault, and A. Sood, "Incorruptible system self-cleansing for intrusion tolerance," in *2006 IEEE International Performance Computing and Communications Conference*, April 2006, pp. 4 pp.–496.

[66] A. J. O'Donnell and H. Sethu, "On achieving software diversity for improved network security using distributed coloring algorithms," in *Proceedings of the*

11th ACM Conference on Computer and Communications Security, ser. CCS '04. New York, NY, USA: ACM, 2004, pp. 121–131. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030101

[67] B. Salamat, A. Gal, and M. Franz, "Reverse stack execution in a multi-variant execution environment," in *Workshop on Compiler and Architectural Techniques for Application Reliability and Security*, 2008, pp. 1–7.

[68] M. Azab, R. Hassan, and M. Eltoweissy, "Chameleonsoft: A moving target defense system," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, Oct 2011, pp. 241–250.

[69] T. Roeder and F. B. Schneider, "Proactive obfuscation," *ACM Trans. Comput. Syst.*, vol. 28, no. 2, pp. 4:1–4:54, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1813654.1813655

[70] S. Vikram, C. Yang, and G. Gu, "Nomad: Towards non-intrusive moving-target defense against web bots," in *Communications and Network Security (CNS), 2013 IEEE Conference on*, Oct 2013, pp. 55–63.

[71] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL injection attacks," in *International Conference on Applied Cryptography and Network Security*. Springer, 2004, pp. 292–302.

[72] K. Trovato, "IP hopping for secure data transfer," Apr. 10 2003, uS Patent App. 09/973,311. [Online]. Available: https://www.google.com/patents/US20030069981

[73] M. Carvalho and R. Ford, "Moving-target defenses for computer networks," *IEEE Security Privacy*, vol. 12, no. 2, pp. 73–76, Mar 2014.

[74] A. Clark, K. Sun, and R. Poovendran, "Effectiveness of ip address randomization in decoy-based moving target defense," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, Dec 2013, pp. 678–685.

[75] Q. Jia, K. Sun, and A. Stavrou, "MOTAG: Moving target defense against internet denial of service attacks," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, July 2013, pp. 1–9.

[76] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2012, pp. 310–327.

[77] J. Jafarian, E. Al-Shaer, and Q. Duan, "An effective address mutation approach for disrupting reconnaissance attacks," *Information Forensics and Security, IEEE Transactions on*, vol. 10, no. 12, pp. 2562–2577, Dec 2015.

[78] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proceedings of the First ACM Workshop on Moving Target Defense*, ser. MTD '14. New York, NY, USA: ACM, 2014, pp. 69–78. [Online]. Available: http://doi.acm.org/10.1145/2663474.2663483

[79] J. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware ip address randomization for proactive agility against sophisticated attackers," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, April 2015, pp. 738–746.

[80] J. Yackoski, P. Xie, H. Bullen, J. Li, and K. Sun, "A self-shielding dynamic network architecture," in *2011 - MILCOM 2011 Military Communications Conference*, Nov 2011, pp. 1381–1386.

[81] J. Yackoski, J. Li, S. A. DeLoach, and X. Ou, "Mission-oriented moving target defense based on cryptographically strong network dynamics," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, ser. CSIIRW '13. New York, NY, USA: ACM, 2013, pp. 57:1–57:4. [Online]. Available: http://doi.acm.org/10.1145/2459976.2460040

[82] J. Yackoski, H. Bullen, X. Yu, and J. Li, *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. New York, NY: Springer New York, 2013, ch. Applying Self-Shielding Dynamics to the Network Architecture, pp. 97–115. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-5416-8_6

[83] M. Albanese, E. Battista, S. Jajodia, and V. Casola, "Manipulating the attacker's view of a system's attack surface," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, Oct 2014, pp. 472–480.

[84] G. A. Fink, J. N. Haack, A. D. McKinnon, and E. W. Fulp, "Defense on the move: Ant-based cyber defense," *IEEE Security Privacy*, vol. 12, no. 2, pp. 36–43, Mar 2014.

[85] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "Mt6d: A moving target ipv6 defense," in *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, Nov 2011, pp. 1321–1326.

[86] O. Hardman, S. Groat, R. Marchany, and J. Tront, "Optimizing a network layer moving target defense for specific system architectures," in *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 117–118. [Online]. Available: http://dl.acm.org/citation.cfm?id=2537857.2537877

[87] S. Groat, M. Dunlop, W. Urbanksi, R. Marchany, and J. Tront, "Using an ipv6 moving target defense to protect the smart grid," in *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, Jan 2012, pp. 1–7.

[88] S. Groat, R. Moore, R. Marchany, and J. Tront, "Securing static nodes in mobile-enabled systems using a network-layer moving target defense," in *Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the*, May 2013, pp. 42–47.

[89] Symantec Security Response, "Regin: Top-tier espionage tool enables stealthy surveillance," Symantec Corporation, Tech. Rep., August 2015.

[90] M. Albanese, S. Jajodia, and S. Noel, "Time-efficient and cost-effective network hardening using attack graphs," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE, 2012, pp. 1–12.

[91] M. Albanese and S. Jajodia, "A graphical model to assess the impact of multi-step attacks," *The Journal of Defense Modeling and Simulation*, p. 1548512917706043, 2017.

[92] D. S. Johnson, M. Minkoff, and S. Phillips, "The prize collecting steiner tree problem: theory and practice," in *SODA*, vol. 1, no. 0.6, 2000, p. 4.

[93] D. Du and X. Hu, *Steiner tree problems in computer communication networks*. World Scientific, 2008.

[94] M. Bateni, M. Hajiaghayi, and V. Liaghat, "Improved approximation algorithms for (budgeted) node-weighted steiner problems," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2013, pp. 81–92.

[95] S. Sadeghian Sadeghabad, "Node-weighted prize collecting steiner tree and applications," Master's thesis, University of Waterloo, 2013.

[96] A. Moss and Y. Rabani, "Approximation algorithms for constrained node weighted steiner tree problems," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 460–481, 2007.

[97] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, October 2002.

[98] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[99] N. Kourtellis, G. D. F. Morales, and F. Bonchi, "Scalable online betweenness centrality in evolving graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2494–2506, 2015.

[100] Y. Yoshida, "Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1416–1425.

[101] L. H. Pham, M. Albanese, and B. W. Priest, "A quantitative framework to model advanced persistent threats," in *Proceedings of the 15th International Conference on Security and Cryptography (SECRYPT 2018)*. Porto, Portugal: SciTePress, July 2018, pp. 282–293, - Best Paper Award.

[102] M. Piraveenan, M. Prokopenko, and L. Hossain, "Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks," *PloS one*, vol. 8, no. 1, p. e53095, 2013.

[103] L. Lü, D. Chen, X.-L. Ren, Q.-M. Zhang, Y.-C. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *Physics Reports*, vol. 650, pp. 1–63, 2016.

[104] D. A. Menascé, "Security performance," *IEEE Internet Computing*, vol. 7, no. 3, pp. 84–87, May/June 2003.

[105] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," Lockheed Martin Corporation, 2010.

[106] "2017 cost of data breach study," Ponemon Institute, Tech. Rep., June 2017.

[107] H. Maleki, S. Valizadeh, W. Koch, A. Bestavros, and M. van Dijk, "Markov modeling of moving target defense games," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. ACM, 2016, pp. 81–92.

[108] N. Nasiriani, Y. Shan, G. Kesidis, D. Fleck, and A. Stavrou, "Changing proxy-server identities as a proactive moving-target defense against reconnaissance for ddos attacks," Dec 2017.

[109] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," in *Moving target defense*. Springer, 2011, pp. 131–151.

[110] H. Okhravi, J. Riordan, and K. Carter, "Quantitative evaluation of dynamic platform techniques as a defensive mechanism," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 405–425.

[111] Apache Software Foundation, "Apache," 2008. [Online]. Available: https://archive.apache.org/dist/httpd/

[112] U. S. C. E. R. Team, "CVE-2014-0160." Available from MITRE, CVE-ID CVE-2012-1823, Dec. 3 2012. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1823

[113] K. A. Farris and G. Cybenko, "Quantification of moving target cyber defenses," in *SPIE Defense+ Security*. International Society for Optics and Photonics, 2015, pp. 94 560L–94 560L.

[114] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia, "A moving target defense approach to disrupting stealthy botnets," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. ACM, 2016, pp. 37–46.

[115] A. K. Bangalore and A. K. Sood, "Securing web servers using self cleansing intrusion tolerance (SCIT)," in *Dependability, 2009. DEPEND'09. Second International Conference on*. IEEE, 2009, pp. 60–65.

[116] C. Morales, Feb 2017. [Online]. Available: https://blog.vectra.ai/blog/an-analysis-of-the-shamoon-2-malware-attack

[117] C. Hosmer, *Python passive network mapping: P2NMAP*. Syngress, 2015.

[118] A. J. Akande, C. Fidge, and E. Foo, "Limitations of passively mapping logical network topologies," *Intl. Journal of Computer Network and Information Security*, vol. 9, no. 2, pp. 1–11, February 2017.

[119] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proc. of the 1st ACM Workshop on Moving Target Defense (MTD 2014)*. Scottsdale, AZ, USA: ACM, November 2014, pp. 69–78.

[120] P. Manadhata and J. M. Wing, "Measuring a system's attack surface," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 2004.

[121] R. Chadha, T. Bowen, C.-Y. J. Chiang, Y. M. Gottlieb, A. Poylisher, A. Sapello, C. Serban, S. Sugrim, G. Walther, L. M. Marvel, E. A. Newcomb, and J. Santos, "CyberVAN: A cyber security virtual assured network testbed," in *Proc. of the 2016 IEEE Military Communications Conference (MILCOM 2016)*. Baltimore, MD, USA: IEEE, November 2016, pp. 1125–1130.

[122] T. Bowen, A. Poylisher, C. Serban, R. Chadha, C.-Y. J. Chiang, and L. M. Marvel, "Enabling reproducible cyber research - four labeled datasets," in *Proc. of the 2016 IEEE Military Communications Conference (MILCOM 2016)*. Baltimore, MD, USA: IEEE, November 2016, pp. 539–544.

[123] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *Proc. of the 2013 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA, USA: IEEE, May 2013, pp. 98–104.

[124] B. Lindauer, J. Glasser, M. Rosen, and K. Wallnau, "Generating test data for insider threat detectors," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 5, no. 2, pp. 80–94, June 2014.

[125] S. Venkatesan, M. Albanese, K. Amin, S. Jajodia, and M. Wright, "A moving target defense approach to mitigate ddos attacks against proxy-based architectures," in *2016 IEEE conference on communications and network security (CNS)*. IEEE, 2016, pp. 198–206.

[126] W. Connell, M. Albanese, and S. Venkatesan, "A framework for moving target defense quantification," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2017, pp. 124–138.

Curriculum Vitae

Luan "Keith" Pham is a cybersecurity professional with experience serving as a U.S. federal contractor with the U.S. Department of Defense (DoD), U.S. Department of Homeland Security (DHS), and the U.S. Intelligence Community. He received two prior degrees from George Mason University: a Bachelor of Science in Information Technology and Master of Science in Information Security and Assurance in 2007 and 2010, respectively. His career assignments has led him to travel throughout the continental United States conducting various forms of security assessments. At the conclusion of his studies, completing the Doctor of Philosophy in Information Technology at George Mason in 2020, he is scheduled to return to federal contracting in the service of the U.S. Department of State and U.S. Agency for International Development (USAID).