ACTIVE CYBER DEFENSE PLANNING AND ORCHESTRATION

by

Md Mazharul Islam

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

Charlotte

2021

ABSTRACT

MD MAZHARUL ISLAM. Active Cyber Defense Planning and Orchestration.
(Under the direction of DR. JINPENG WEI)

The overwhelming number of recent data breaches reported that hundreds of terabytes of highly sensitive information, including national, financial, and personal, have been stolen from different organizations, indicating a clear asymmetric disadvantage that defenders face against cyber attackers. Modern attackers are well organized, highly stealthy, and stay persistent in the network for years; therefore, they are known as advanced persistent threats (APT). Existing detection and prevention based cyber defense techniques usually approach the target for specific, known attack signatures, descriptions, and behaviors. However, APT attackers can easily avoid such detection techniques by employing reconnaissance, fingerprinting, and social engineering. It is often very challenging and sometimes infeasible for defenders to prevent the information gathering of the adversary and patch all the vulnerabilities in the system. Therefore, a proactive defense approach is needed to break such asymmetry.

Active Cyber Defense (ACD) is a promising paradigm to achieve this goal. ACD can proactively mislead adversaries and enables a unique opportunity to engage with them to learn new attack tactics and techniques. ACD enhances real-time detection, analysis, and mitigation of APT attacks. ACD can be achieved through cyber agility and cyber deception. Cyber Agility, such as moving target defense (MTD), enables cyber systems to defend proactively against sophisticated attacks by dynamically changing the system configuration parameters (called mutable parameters) in order to deter adversaries from reaching their goals. On the other hand, Cyber Deception is an intentional misrepresentation of the system's ground truth to manipulate adversaries' actions.

Although cyber deception and MTD have been around for more than decades, static configurations and the lack of automation made many of the existing techniques easily discoverable by attackers and too expensive to manage, which diminishes the value of these technologies. Sophisticated APTs are very dynamic and thereby require a highly adaptive and embedded defense that can dynamically create honey resources and orchestrate the ACD environment appropriately according to the adversary behavior in real-time.

To overcome these challenges, this dissertation introduced an autonomous resilient ACD framework, having the following aspects: (1) developing multistrategy ACD policies that leverage an optimal dynamic composition of various MTD and deception techniques to maximize the defense utility, (2) a policy specification language and an extensible rich API integrated with a synthesis engine for developing different MTD techniques without consulting about the low-level network and system configuration management, (3) a theoretical framework and implementation for an autonomous goal-oriented cyber deception planner that optimizes deception decision-making.

DEDICATION

To my mom, everything she does is just to bring a little smile to my face.

ACKNOWLEDGEMENTS

I owe a great debt of gratitude to Dr. Ehab Al-Shaer who was my advisor for the last four years of my Ph.D. career. He moved to the School of Computer Science at Carnegie Mellon University in June 2020 as a Distinguished Career Professor. However, he remotely supervised me to finish my dissertation from CMU. His in-depth knowledge, interests, and passion for network and system security research, especially in the cyber deception paradigm, make him an exemplary researcher and mentor. Without his supervision and dedication, I would not succeed in completing my work in this dissertation. I am very thankful to my current advisor Dr. Jinpeng Wei, who help me to shape and finalize my dissertation. He continuously helps me to write and publish papers, especially throughout the whole dissertation writing process. I would also like to express my sincere gratitude to my dissertation committee members: Professor Bill Chu, Professor Badrul Chowdhury and Professor Matthew Whelan, for serving on my dissertation committee and their precious comments and suggestions.

I want to thank my family, my mom, my brothers, and my sister, who support me no matter what I have gone through. Ph.D. is a long journey, and I often faced difficulties that could be unbearable without their support. A big shout out to all my friends and colleges who are my constant companion in this long journey. Thank you Ashu, Maruf, Mohi, Purba, Sakib, Madiha and Ehsan. A special thanks to Wasi, Sazzad vai and Bony vai, who always believe in me.

Last but not least, I would like to thank the faculty members and the Department of Software and Information Systems staff in the University of North Carolina at Charlotte, my fellow graduate students, and my research collaborators for their academic and administrative support throughout my doctoral study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: Introduction

The recent data breaches reports showed that hundreds of terabytes of highly sensitive information, including national, financial, and personal, have been stolen from more than 150 organizations in 2021 [1]. In 2020, the FBI's Internet Crime Complaint Center released a report stating that \$13.3B has been lost due to cybercrime where adversaries stole information from organizations and individuals by spear phishing, ransomware, and information-stealing malware [2]. These attacks are well organized, highly stealthy, and stay persistent in the organization for more than a year on average [3]; therefore, they are known as advanced persistent threats (APTs). A myriad amount of research has been done from academia and industry to mitigate APT attacks. However, every two seconds in the US, people are losing their sensitive data to information stealer [4], and every minute almost six malware attack happens [5]. For instance, the May 2021 ransomware attack in Colonial Pipeline Co. cost them millions of dollars of losses [6]. This indicates APT attackers are capable of penetrating existing security systems as they have asymmetric leverage in cyber warfare- defenders require to protect all vulnerabilities, yet; the attackers need a few susceptibilities to exploit.

Active Cyber Defense (ACD) provides a proactive cyber security technique that can reverse the asymmetry between adversary and defender in cyber warfare. Traditional reactive intrusion response depends on adversary engagement, detection of the attack vectors (finding malicious code, malware, etc.), generating patches, and finally applying them into the network and systems. This lack of addressing many key challenges, including advanced adversary adaptation, changes into attack strategies, or small modifications into the malware, leading them to bypass the existing security patches easily. Moreover, patching requires days, weeks or even months and often, many vulnerabilities remain unpatched because of recourse constraints [7, 8]. Therefore, to precede advanced and sophisticated adversaries in cyber warfare, the defender must practice proactive and adaptive techniques rather than static reactive defense. ACD plays a game-changing role in proactive defense. ACD can be leveraged by moving target defense (MTD) and cyber deception techniques.

MTD enables cyber systems to defend proactively against sophisticated attacks by dynamically changing the system configuration parameters, often known as mutable parameters, in order to deter adversaries from reaching their goals. On the other hand, cyber deception is an intentional misrepresentation of the ground truth of real systems to manipulate adversaries' actions [9]. However, developing and deploying adaptive cyber deception and MTD techniques in real-life operational networks is an extremely complex and time-consuming task due to the extensive efforts required to implement the underlying network infrastructure configuration functions that are necessary to support active deception and mutation operations, including observing, planning, and deploying honey resources at real-time. Moreover, existing ACD techniques mostly suffer from a single point of failure, which means if one defense action fails, it is highly likely that adversaries will achieve their goals or discover the honey resources. Besides, ACD requires expensive infrastructure and management resources to maintain attraction and believability. ACD techniques frequently change low-level network and system configurations. These are costly operations because the modified configuration involves conflict resolution and correctness verification to ensure the mission's integrity.

Most importantly, ACD techniques that use static configuration and planning can be detected and avoided by skilled attackers. Therefore, a dynamic and adversary adaptive MTD and deception planning need to be incorporated to defeat sophisticated attackers. This requires an intelligent framework that can make defense decisions against probable adversary actions by observing the attack actions and make sequential decisions. Therefore, ACD policymakers in this field often spend significant time and effort building such infrastructural functions rather than focusing on developing sophisticated strategies for MTD and cyber deception applications.

To overcome these challenges, this dissertation introduced an autonomous resilient ACD framework, having the following aspects: (1) developing multistrategy ACD policies that leverage an optimal dynamic composition of various MTD and deception techniques to maximize the defense utility, (2) a policy specification language and an extensible rich API integrated with a synthesis engine for developing different MTD techniques without consulting about the low-level network and system configuration management, (3) a theoretical framework and implementation for an autonomous goal-oriented cyber deception planner that optimizes deception decision-making.

1.1 Research Objectives

The main objective of this dissertation is to build an ACD framework that can be used to develop different MTD and deception techniques without considering the continuous low-level system configuration, optimal planning and safe deployment of the defense techniques. This section discusses the individual objectives of each chapter of the dissertation.

- The first objective of this dissertation is to deter the most effective adversary tactic in the kill chain, which is initial access to the system through spear-phishing attacks. Most of these spear-phishing attacks happen in the form of email communication, known as email spear-phishing attacks, where adversaries attach malicious links (URL) or files (malware) into the email and send the phishing email to targeted victims. Email spear-phishing attacks have been one of the most devastating cyber threats against individual and business victims in recent years. Using spear-phishing emails, adversaries can manage to impersonate authoritative identities in order to incite victims to perform actions that help adversaries to gain financial and/hacking goals. Many of these targeted spear-phishing can be undetectable based on analyzing emails because, for example, they can be sent from compromised benign accounts (called lateral spear-phishing attacks). The current state of the art for detecting spear-phishing emails limits by analyzing email payloads/headers. However, adversaries can easily evade detection by mimicking users' behavior and avoiding bad signatures [10]. To address these limitations, I introduced a novel moving target technique called sender email address mutation to protect against lateral spear-phishing and spoofing attacks proactively (chapter 2).

- Cyber agility and deception techniques enable cyber systems to defend proactively against sophisticated attacks by dynamically changing the system configuration parameters in order to deceive adversaries from reaching their goals, disrupt the attack plans by forcing them to change their adversarial behaviors, and/or deterring them through prohibitively increasing the cost for attacks. However, developing and deploying cyber agility such as moving target defense and adaptive cyber deception techniques in real-life operational networks is extremely complex and time-consuming tasks due to the extensive effort required to implement the underlying network infrastructure configuration functions. These functions are necessary to support active cyber deception and MTD operations, including real-time observing, planning, and deploying honey resources. Therefore, developers in this field often spend significant time and effort building such infrastructural functions rather than focusing on developing sophisticated strategies for cyber defense applications. The second objective of the dissertation is to address these challenges by providing a framework for automating the creation of configuration-based defense techniques rapidly and safely. In chapter 3, such a framework is presented that provides a high-level cyber agility policy language specification and a controller for implementing configuration-based

MTD and deception techniques. It also provides an extensible rich API that can be used to observe adversary actions, compose multi-strategy defense plans, and ensure safe yet quick deployment of such plans by automatically managing the network configuration.

- Cyber deception is a promising defense that can proactively mislead adversaries and enables a unique opportunity to engage with them to learn new attack tactics and techniques. Although cyber deception has been around for more than a decade, static configurations and the lack of automation made many of the existing deception techniques easily discoverable by attackers and too expensive to manage, which diminishes the value of this technology. Sophisticated APT attackers are highly dynamic and thereby require a highly adaptive and embedded deception that can dynamically create honey resources and orchestrate the deception environment appropriately according to the adversary behavior in real-time. The final objective of this dissertation is to provide optimal planning of deception course of actions and their automatic orchestration in the production environment in real-time. Therefore, the defender can design specific deception scenarios to deceive APT attacks in order to achieve different deception goals. In chapter 4, a framework named CHIMERA is presented that generates a sequence of optimal deception action planning based on the APT attacks and deception goals by analyzing the adversary tactics leveraging MITRE ATT&CK [11] and using Sequential Decision-Making techniques such as Partially Observable Markov Decision Processes (POMDP).

1.2 Background

ACD is a cyber resiliency capability that dynamically orchestrates security architectures to prevent attacks proactively, while adapting security policies and configurations based on active investigation of threat observables. This dissertation focuses

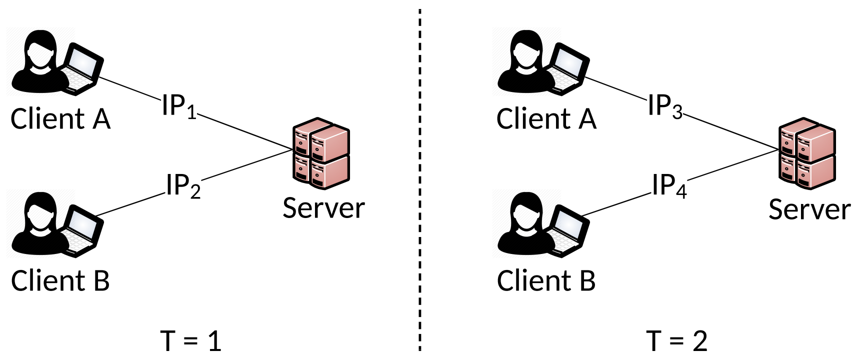


Figure 1.1: IP mutation example.

on MTD and cyber deception to leverage ACD against APT attacks.

1.2.1 Moving Target Defense

Moving Target Defense (MTD) is a systematic periodical reconfiguration of network resources parameters in order to dynamically change the available attack surface (such as IP address, network route, and so on). MTD enables proactive defense by deterring the adversary from reaching its goal. For example, IP address mutation can deter network reconnaissance attackers by changing the IP address of the critical resources in a periodic fashion. Figure 1.1 shows an example of how IP mutation works. In IP mutation, each critical resource is assigned with an ephemeral IP address, which periodically gets changed. Thus, the accumulation of IP address information in a reconnaissance attack gets invalidated after that period. For instance, according to the figure 1.1, in time $T = 1$, each client communicates with the server through IP1 and IP2, respectively. However, in $T = 2$, these ephemeral IP gets changed to IP3 and IP4. Therefore, if any of the clients store the server's IP address, she cannot use that information later to reach (attack) the server. Such IP mutation techniques enforce network scanning attackers to probe the network each time period, making them less stealthy and eventually get exposed.

1.2.2 Cyber Deception

Cyber Deception is an intentional misrepresentation of real systems' ground truth to manipulate adversary's course of actions under the premises of the defenders' rules [9]. Cyber deception can be used to *divert* the adversary away from the real target to a false or no target when the adversary is already in the system—e.g., providing honey files [12] while the attacker searches for sensitive files. The defender can *distort* the adversary's perception of the infrastructure by adding ambiguity into the system, e.g., running fake services with obvious vulnerabilities (honey-patches [13]). Deception can *deplete* adversary's computational power and resources to delay the attack propagation—for example, honey encryption [14] of the credential files, which the adversary needs to decrypt. Finally, the defender can *discover* new attack tactics, techniques, and procedures (TTPs) by letting them execute different attack actions in contained honey resources.

1.2.3 Spear-phishing Attack

To launch an attack, the first step of the APT actor is to initiate the attack vector (malware, email attachments, etc.) into the system or victim's machine. This tactic is known as initial access, which is often performed by techniques such as a spear-phishing attachment. In email phishing attacks, adversaries send generic emails with malicious attachments or links to a massive number of users indiscriminately, hoping that someone will take the bait. However, spear-phishing attack is more targeted. In spear-phishing attacks, adversaries impersonate key personnel or authoritative identities to incite victims to perform such actions that help them gain certain goals. Adversaries carefully select their targets and send them well-crafted phishing emails that closely resemble what they usually receive. Mostly, these attack emails mimic a familiar style and signature of a known person to the victims, where both the email headers and body merely deviate from benign emails. Yet, the email contains a 'lure' that is convincing enough to engage the victim into an ‘exploit’ [15].

A common technique for spear-phishing attack is to spoof the name or email address of the sender, known as *source spoofing*, to convince the victim that the email is sent from a trusted source [15]. Solutions like SPF [16], DKIM [17], and DMARC [18] that verifies the sender authenticity may prevent email source spoofing [19]. However, a more stealthy variation of spear-phishing attack bypasses such sender authentication protocols, known as *lateral spear-phishing* attack, where adversary uses compromised email accounts of someone either socially or professionally connected with the victim to initiate the phishing email [10]. These phishing emails are very hard to detect because of the cleverly crafted content created by deep analyzing the prior conversation with the victim from that compromised account. Therefore, adversaries inherently win the cyber game against defenders in the lateral spear-phishing attack by evading any existing security regarding sender email authentication as the phishing email coming directly from a legitimate account and defeating behavioral anomaly detectors by accessing human anchoring as the email seemingly composed by a trusted entity [10].

1.3 Research Challenges

This section describes the research challenges addressed in the dissertation.

- Email spear-phishing attack is the most common tactic adversaries use to launch an attack because it is hard to detect. Existing spear-phishing detectors depend on either signature or behavioral analysis of the email content/header. Advanced attackers can easily bypass signature-based analysis by modifying the attack vector (e.g., changing in malware code) and behavioral analysis by clearly crafting the phishing email content. Therefore, a proactive approach is required to provide an alternative solution in detecting email spear-phishing attacks.

- Email security protocols such as SPF, DKIM, and DMARC can verify the sender's authenticity. These protocols are effective against email spoofing attacks. However, the lateral spear-phishing email is sent from benign but compromised email accounts. Thus, the above protocols can not prevent the lateral spear-phishing attack.

- Other cryptographic approaches such as PGP [20], S/MIME [21], Two-factor authentication (2FA) [22] can be used to authenticate email senders and prevent email account hijacking. However, these techniques are not widely used in practice due to transparency, usability, and management challenges [23–25]. For instance, PGP signature and encryption obfuscate the plaintext emails into cyphertext, immediately losing the content's visibility. Therefore, existing IDS and content-based behavioral analysis tools can not work with PGP encryption. Furthermore, PGP requires user training on Public key infrastructure (PKI) and maintains complex key management systems. Moreover, PGP signatures in email can be spoofed [26].

- Most of the existing frameworks lack of providing multi-strategy MTD and deception action composition because these policies can conflict with each other. Moreover, lack of automation may lead to misconfiguration of system parameters that can create new vulnerabilities.

- An efficient ACD composition framework needs continuous network monitoring to observe adversary activities, optimal planning for cost-effective decision-making, and safe deployment without breaking the mission integrity.

- The framework should provide an extensible rich API to create new defense strategies. The implementation of the framework needs to be scalable and easily extendable for integrating future defense techniques.

- Designing a deception environment to deceive APT attacks requires considering both the attacker's action and deception action. The true strategy of the attacker is unknown to the defender. In addition, the alerts defender receives from the system are noisy and often high false positive.

- Synthesizing an optimal deception planning (deception CoAs) under uncertain attack behavior and with limited observability can incur high state space and computation complexity.

- The deception environment should be dynamic to cope with adaptive adversaries in order to provide defense in real-time. Moreover, the risk of a deception action failure need to be accounted, because if the deception action fails, the adversary will achieve their goal.

- An embedded deception strategy needs to be implemented to provide real-time deception in the operational environment because standalone deception environments such as Virtual Machisen or Sandbox cannot be used to deceive the malware if it is already running into a production machine.

1.4 Contributions

This dissertation presents the following key contributions:

- The first chapter presents a novel proactive defense technique using sender Email address Mutation to protect a group of related users against lateral spear-phishing. In Email Mutation, the sender email address get frequently changed randomly that can only be verified by trusted peers, without imposing any overhead or restriction on email communication with external users. Email mutation technique is transparent, secure, and effective because it allows users to use their email as usual, while they are fully protected from stealthy spear-phishing. The

Email mutation technique (algorithm and protocol) is well described and a formal model is developed to verify its correctness. The processing overhead due to mutation is a few milliseconds, which is negligible with the prospective of end-to-end email transmission delay. A real-world implementation of the Email mutation technique is also shown which works with any email service providers such as Gmail, Apple iCloud, Yahoo Mail, and seamlessly integrates with standard email clients such as Gmail web clients (`mail.google.com`), Microsoft Outlook, and Thunderbird.

- The second chapter presents a cyber agility synthesis framework that contains a formal ontology, MTD policy language, and a controller synthesis engine for implementing configuration-based moving target defense techniques. The policy language contains the agility specifications required to model the MTD technique, such as sensors, mutation triggers, mutation parameters, mutation actions, and mutation constraints. Based on the mutation constraints, the MTD controller synthesis engine provides an MTD policy refinement implementation for SDN configuration with provable properties using constraint satisfaction solvers. The framework also provides an extensible rich API for developing advanced cyber deception applications. The API can be used to observe adversary actions, compose multi-strategy deception plans, and ensure safe yet quick deployment of deception plans by automatically managing the network configuration and operational tasks. The framework is developed over OpenDaylight SDN controller as an open programming environment to enable rapid and safe development of sense-making and decision-making MTD and deception actions.

- The third chapter presents a theoretical framework and implementation for an autonomous goal-oriented cyber deception planner, called CHIMERA, that optimizes deception decision-making. CHIMERA agents can reside in any pro-

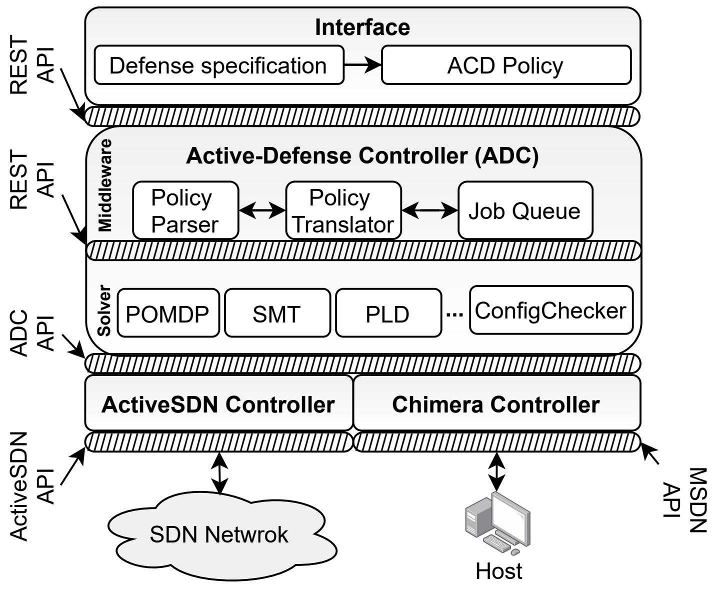


Figure 1.2: Active Cyber Defense Framework.

duction machine/server and automatically create and orchestrate the deception ploys (actions) to steer and mislead the malware or APT to the desired goal without human interaction. The deception ploys are dynamically composed based on the deception planning while ensuring safe yet fast deployment and orchestration of deceptive course-of-actions. CHIMERA is evaluated with real APT attacks for information stealing, ransomware, Remote Access Trojans (RAT), and others. In different case studies with 4,578 real malware samples, CHIMERA's adversary-aware dynamic deception strategies were able to effectively accomplish the deception goals within a few seconds and with minimum cost.

Figure 1.2 shows an overview of the ACD framework. It has the following components:

Interface: The ACD framework provides an interface for ACD policy creation leveraging a high level language specification described in figure 3.3. Authentic users can use the interface to generate defense policies for rapid deployment of MTD defense or deception with full concurrency and safety. The policy creation interface provides an easy description of how to follow the language specification to create the correct MTD policy.

Active Defense Controller: The Active Defense controller (ADC) in the framework is the central orchestrator that handles the end-to-end processing of the cyber deception from initiation by the interface to the safe deployment in the network or systems. ADC provides an open playground that enables prototyping or building advanced deception planning rapidly and safely. ADC leverages its facilities by providing an extensible API, called ADC API, that gives access to sophisticated cyber deception and system management functions using the distribute controller (ActiveSDN and Chimera). The ActiveSDN controller incorporates defense actions related to the software-defined network, and the Chimera controller incorporates defense actions for an individual machine.

Besides, ADC incorporates with a decision-making synthesis engine called solver, that is capable of solving computationally hard problems to optimize deception policy actions. ADC composes the defense triggering by the interface, ensure safe low-level configuration changes, and deploy the planning into the network.

Middleware: The middleware parses and translates the high-level defense specification to intermediate controller interfaces (ADC API). It incorporates the solver through REST for solving constraint problems in order to optimize deception planning. Middleware creates a back-and-forth communication bridge to ADC with the user interface through REST API and the network or system through ADC API.

Solver: The solver is to optimize constraint problems to generate a feasible and practically deployable deception or MTD configuration. The solver is designed as a plug-and-play model in the architecture. Therefore, various deception and configuration optimization solution can be added with ADC as required. Different solver is integrated with the framework such as Satisfiability modulo theories (SMT) [27] to optimize anonymity and diversity of concealment configuration [28], ConfigChecker [29] to solve reachability constraints, partially observable Markov decision process (POMDP) [30] for sequential decision making, etc. ADC incorporates with the solver through middleware via REST API.

CHAPTER 2: Email Address Mutation for Proactive Deterrence Against Lateral Spear-phishing Attack

2.1 Motivation

In recent years, email spear-phishing becomes the most effective cyber threat against individual and business victims. It has been reported that 90% of data breaches in 2017-2018 included a phishing element, 74% of public sector cyber-espionage, and 64% of organizations' attacks involve spear-phishing attacks, where 30% of these attacks are targeted [31]. Over $26B has been lost to spear-phishing and account takeover in 2019 [32]. Only in the US, 71.4% of phishing attacks and data breaches associated with nation-state or state-affiliated actors used spear-phishing [33].

Unlike phishing, where adversaries send generic emails with malicious attachments or links to a massive number of users indiscriminately hoping that someone will take the bait, spear-phishing is more targeted [34]. In spear-phishing attacks, adversaries impersonate key personnel or authoritative identities in order to incite victims to perform such actions that help them to gain certain goals. Adversaries carefully select their targets and send them well-crafted phishing emails that closely resemble what they usually receive. Mostly, these attack emails mimic a familiar style and signature of a known person to the victims, where both the email headers and body merely deviate from benign emails. Yet, the email contains a 'lure' that is convincing enough to engage the victim into an 'exploit' [15].

A common technique for spear-phishing attack is to spoof the name or email address of the sender, known as *source spoofing*, to convince the victim that the email is sent from a trusted source [15, 35]. Solutions like SPF [16], DKIM [17], and DMARC [18] that verifies the sender authenticity may prevent email source spoofing [19]. However, adversaries are continuously evolving and adapting their attack strategies. As a result, a more stealthy variation of spear-phishing attack has been encountered recently, known as *lateral spear-phishing* attack, where adversary uses compromised email accounts of someone either socially or professionally connected with the victim to initiate the phishing email [10]. These phishing emails are very hard to detect because of the cleverly crafted content created by deep analyzing the prior conversation with the victim from that compromised account. Therefore, adversaries inherently win the cyber game against defenders in the lateral spear-phishing attack by evading any existing security regarding sender email authentication as the phishing email coming directly from a legitimate account and defeating behavioral anomaly detectors by accessing human anchoring as the email is seemingly composed by a trusted entity [10]. These facts motivate our research to develop a proactive mechanism for protecting the number one targeted attack vector, email.

The current state of the art for detecting lateral spear-phishing emails mainly depends on email headers and body [10, 15, 36–42]. These defense techniques require users' historical data to model a behavioral profile for the sender or receiver in order to detect the anomalous behavior of the phishing email. They also depend on analyzing the content of phishing emails searching for malicious URLs or attachments, domains with low reputation, etc. However, spear-phishers can easily evade detection by mimicking users' behavior (from previous emails) and avoiding the use of bad features [10].

To address these limitations, a novel moving target defense technique called sender Email address Mutation (EM) is developed to proactively protect a group of users against lateral spear-phishing and spoofing attacks. EM is developed as a cloud-based service that can be easily integrated with existing email infrastructure for any organization to offer scalable email protection against spear-phishing with minimal management overhead. It deploys a secure gateway in the cloud that works transparently between end-users and email service providers. EM defends a group of socially or professionally connected members, called the VIP users. It creates a number of random *shadow* email addresses (accounts) associated with each VIP user besides their actual email address. These shadow email addresses are used as the sender for email delivery but are hidden to both end-users.

While two VIP members communicate with each other through EM, the email first goes to the secure email mutation gateway (EMG) in the cloud, where EMG translates the sender email address to a shadow email address corresponding to the sender before forwarding it. Similarly, when the receiver VIP user fetches that email, the EMG verifies the shadow email address and delivers the email to the recipient, if the verification is successful. Therefore, knowing the public email address will not be sufficient to attack the VIP users. Spear-phisher adversaries must correctly guess the current shadow email being used by each individual user in order to successfully impersonate a VIP user in an email sent to another VIP user. While EM achieves this protection between VIP users, it also maintains the email open communication model by allowing VIP users to receive and send emails to any external users without any restriction. Thus, EM can protect a group of socially or organizationally connected (VIP) users from any phishing emails that impersonate a VIP user even if the email is coming from a compromised VIP email account, without impacting users' usability or interaction with external users.

PGP [20], S/MIME [21], Two-factor authentication (2FA) [22] can be used to authenticate email senders and prevent email account hijacking. However, these techniques are not widely used in practice due to many users' transparency, usability, and management challenges [23–25]. For instance, PGP signature and encryption obfuscate the plaintext emails into cyphertext, immediately losing the content's visibility. Therefore, existing IDS and content-based behavioral analysis tools can not work with PGP encryption. Furthermore, PGP requires user training on Public key infrastructure (PKI) and maintains complex key management systems. Moreover, PGP signatures in email can be spoofed [26]. EM provides an alternative proactive mechanism for the majority of email users who are not using PGP and/or 2FA to protect against spear-phishing without compromising transparency, usability, or manageability. Therefore, although EM does not use a cryptographic approach like PGP or 2FA, it can provide comparable protection while maintaining high usability and deployability.

Our key contribution is three-fold:

- First, a novel protocol called EM is introduced, as a proactive defense against highly stealthy lateral spear-phishing attacks.
- Second, the verification of the EM protocol is shown so that it can be integrated with any existing email service provider.
- Third, the EM system is implemented (code available on GitHub) and deployed it in a real-world environment without imposing any usability or performance overhead on users or service providers [43].

2.2 Problem Statement

Given a group of user email accounts associated with organizationally or socially, the goal of Email Mutation is to protect the users from lateral spear-phishing attacks (including spear-phishing and spoofing attacks) by changing the sender email address to shadow email address frequently without breaking reachability and transparency among them with minimal system overhead.

Shadow Email Address. The shadow emails are a list of pre-created email accounts assigned to all VIP users but being kept hidden from them. These accounts are only used in email transmission as a sender address. Only EMG conducts with these email accounts. Depending on the impacts, the number of shadow email addresses assigned to a VIP user varies. EM is flexible to the creation of shadow email accounts as long as the shadow email domain is the same as the real email domain. However, in experiment, a prefix “sid" (shadow ID) is used in the shadow email address to make a clear difference with the real email address. A possible shadow email address may look like: *real.email.address.x@domain*, where x is at least 16 byte long random alphanumeric sequence. For instance, *alice.sid8aiy5vgia0ta4uec@org.com* can be one of the shadow email addresses for Alice’s real email address *alice@org.com*, where x = *sid8aiy5vgia0ta4uec*.

Reachability. Reachability ensures that EM does not obstruct email communication between users. EM users have their regular email communication with all other users but protected from any lateral spear-phishing attack.

Transparency. Transparency means the EM users do not lose the visibility of their emails, as EM doesn’t modify the header or body of the email on the client-side. For instance, PGP encryption encrypts the email body, losing the plaintext email visibility from the users. Besides, the end-users are oblivious to the whole mutation techniques.

2.3 Related Work

A vast amount of research has been done to detect phishing [19, 37, 44–46] and spear-phishing attacks [10, 15, 38–42]. The majority of these works depend on email content or headers. For instance, Ho et al. presented a learning-based classifier to detect lateral spear-phishing by seeking malicious URLs embedded in the phishing email [10, 15]. However, these solutions will not work against motivated adversaries trying to evade detection simply just by not adding any malicious URL or no URL at all in the phishing email.

Spear-phishing detectors like EmailProfiler [40] and IdentityMailer [41] also depend on email headers and body to build behavioral profiles for each sender based on stylometric features in the content, senders writing habits, common recipients, usual email forwarding time, etc. New emails get compared with the profile to measure the deviation determining whether they are spear-phishing email or not. These solutions can not detect lateral spear-phishing emails when the contents are carefully crafted to avoid deviation from the norm. Moreover, they show a high false-positive rate (10%), which becomes unacceptable when it comes to the large volume of emails in a real-world enterprise network.

Gascon et al. [38] proposed a context agnostic spear-phishing email detector by building behavioral profiles for all senders in a given user mailbox. They create such profiles from the common traits a sender follows while composing an email, like attachments types, different header fields, etc. However, in the lateral spear-phishing attack, email headers do not deviate from the usual structure as it is coming from a legitimate account. In addition, building profiles for each sender can induce massive overhead in large scale deployment.

Existing sender authentication protocols such as SPF [16], DKIM [17], and DMARC [18] can not detect lateral spear-phishing emails because they are not spoofed and composed from valid email accounts. Other solutions, such as signing emails with PGP [20], S/MIME [21], or 2FA [22] can prevent the lateral spear-phishing attack. Unfortunately, these techniques are not widely used because of usability, manageability, and transparency issues [23–25]. Moreover, a recent study showed that PGP signatures in the email could be spoofed as well [26].

2.4 Threat Model

2.4.1 Attack Taxonomy

In the lateral spear-phishing attack, adversaries send phishing emails to victims from a compromised account. To make such attacks trustworthy and effective, adversaries carefully choose those compromised accounts that are closely related to the victims, such as employees from the same organization [10]. Therefore, the attacker

```
From: Alice <alice@org.com>
To: Bob <bob@org.com>
Subject: February, 2020 Meeting Budget (Event venue booking)
Hi Bob,
Process wire transfer of $100,543 to Trudy (account no. 5648132796,
routing no. 026001234) to finalize upcoming  event venue bookings.
Send me an invoice of that transaction ASAP, thanks.
Alice
CEO, org.com
```

Listing 1: A carefully crafted lateral spear-phishing email sends to Bob from a compromised account Alice, without any malicious attachments or URLs.

easily bypasses traditional email security systems like sender authentication, as the email is come from a legitimate account and make the victim fall for the attack, as it is seemingly composed by a person they already trust. Listing 1 depicts an example of lateral spear-phishing email. Adversary Trudy compromises the email account of Alice, CEO of an organization (`org.com`). By examining her inbox, Trudy obtains that Alice directed Bob, finance department head of `org.com`, to make some wire transactions for arranging an upcoming business meeting. Exploiting this analysis, Trudy composes a phishing email from Alice's email account to Bob, directing him to make a wire transaction in Trudy's bank account. These types of lateral phishing are crafted carefully by observing previous emails and may not contain any malicious attachments or URLs that make it very hard to detect.

2.4.2 Email Mutation Attack Model

The EM protocol detects lateral spear-phishing and spoofing attacks where adversaries send phishing emails to the victim from a compromised benign email account or impersonate a benign person that the victim already trusts. Compromising an email account means adversary gain access only to that email account, not the physical machine such as laptop, desktop, or cell phone itself of the user. Moreover, any compromised account who never communicated with the victim before can hardly be

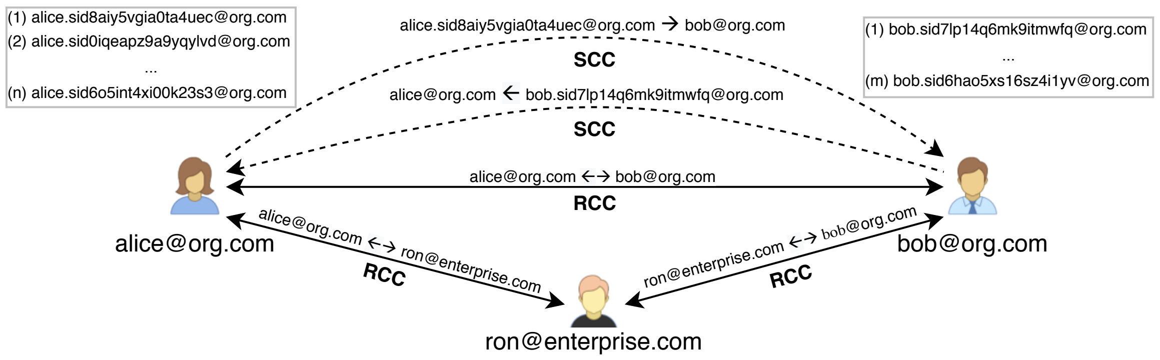

(1) alice.sid8aiy5vgia0ta4uec@org.com
(2) alice.sid0iqeapz9a9yqylvd@org.com
...
(n) alice.sid6o5int4xi00k23s3@org.com
alice.sid8aiy5vgia0ta4uec@org.com → bob@org.com
SCC
alice@org.com ← bob.sid7lp14q6mk9itmwfq@org.com
SCC
(1) bob.sid7lp14q6mk9itmwfq@org.com
...
(m) bob.sid6hao5xs16sz4i1yv@org.com
alice@org.com ←→ bob@org.com
RCC
alice@org.com
bob@org.com
alice@org.com ←→ ron@enterprise.com
RCC
ron@enterprise.com ←→ bob@org.com
RCC
ron@enterprise.com

Figure 2.1: Email Mutation overview. Alice and Bob send emails using their shadow email addresses (dashed line, single arrow).

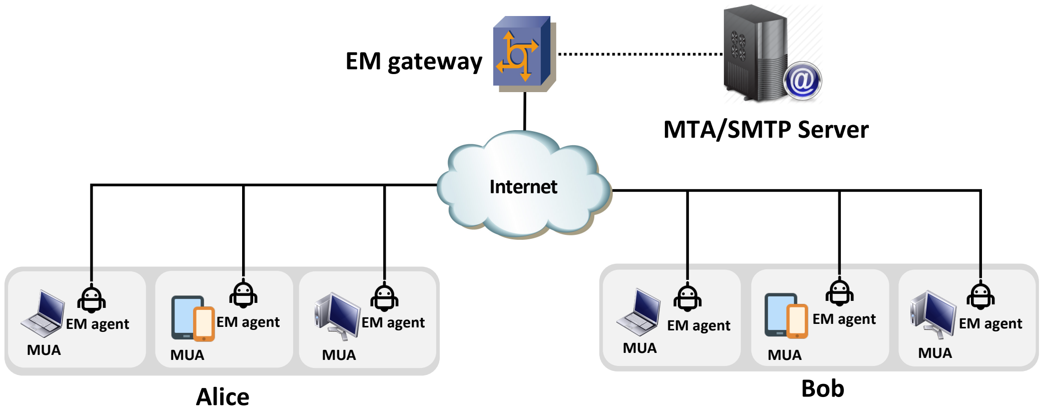


Figure 2.2: Email mutation architecture.

successful in exploiting the victim. Therefore, EM solely focuses on compromised accounts and impersonated entities that are connected with the victim, e.g., employees from the same organization or different organization, but communicates frequently. The people who use EM to protect themselves against spear-phishing attacks are denoted as VIP users. To launch a lateral spear-phishing attack, an adversary needs to send the phishing email from a compromised account, Alice, for instance, whom Bob already connected with. EM can protect Bob against such an attack if both Bob and Alice are agreed prior to use EM; therefore, they are in the VIP user list. EM also protects VIP users from spoofing if the adversary impersonates any of the VIP users in the phishing email.

2.5 Email Mutation System Overview

Figure 2.1 depicts an overview of sender email address mutation, where two VIP users Alice and Bob from an organization (`org.com`), agreed to use EM to protect themselves against lateral spear-phishing attacks. Previously, they communicate with each other using their real email address (double arrow solid line), called Real channel communication (RCC). However, after EM starts, each VIP user gets a list of shadow email accounts. For instance, Alice and Bob get n and m number of shadow email accounts (addresses), respectively. Thus, each new email Alice composes for Bob now uses a different shadow email address as a sender instead of her real email address, and the modified (mutated) email is forwarded to Bob (dashed line single arrow). This is called Shadow channel communication (SCC). Sending an email in SCC by mutating the sender email address is known as *mutation*.

When Bob receives such an email, the shadow email address gets verified for lateral spear-phishing detection. This is called *verification*. Similarly, when Bob composes a new email for Alice, the email is forwarded through SCC by mutating Bob's real email address to one of his shadow email addresses. Although Alice and Bob use shadow email addresses as the sender to communicate with each other through SCC, they use their real email address to communicate with external users (non-VIP), e.g., Ron from `enterprise.com`. VIP users group can comprise people from different organizations having different domains. For instance, John (not shown in the figure) from `gov.com` can be a VIP member with Alice and Bob from `org.com`.

2.5.1 Architecture

The *mutation* and *verification* happens in the cloud by mutation gateways (EMG). Figure 2.2 illustrates the architecture of EM. Clients can use multiple devices such as laptops, desktop, or cell phones for accessing emails; therefore, EM provides an EM agent (EMA) for each of the devices. While sending an email, the agent delivers

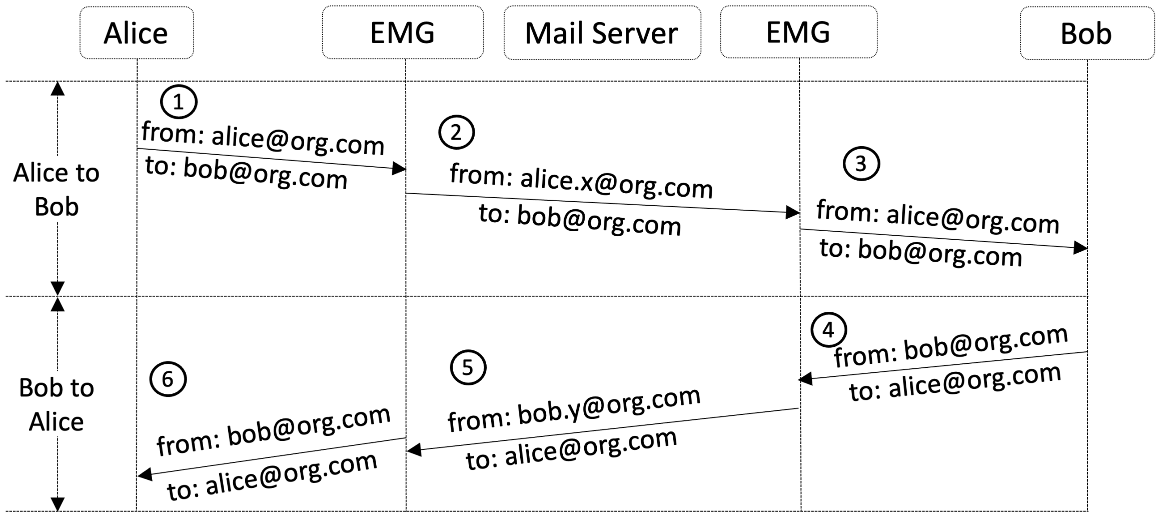


Figure 2.3: Benign email communication between two VIP members using EM.

the email to the EMG for mutation. After mutation, the EMG forwards the mutated email to corresponding mail servers (SMTP/MTA). Similarly, while fetching a new email, the agent receives it from the EMG. The EMG first gets the email from the mail server and then verifies it to detect a lateral spear-phishing attack before responding to the agent. In large enterprise networks, multiple EMGs can be used for load balancing.

2.5.2 Algorithm

The VIP users supposedly send emails to each other, which use as ground truth G next time they send any new emails. For instance, when a VIP user i sends an email to another VIP user j, the last l emails between them will be used as ground truth ${}_iG_j$ to generate a mutation ID, *mID*. By indexing the *mID*, a shadow email address gets selected from a secret arrangement of shadow email addresses S_i assigned for the sender i. The shadow email address is then used to forward the email. Similarly, as the receiver j has the identical ground truth, j can generate the exact *mID* to find the same shadow email address from S_i for verification.

Algorithm 1 Shadow Selection

```
1: procedure SELECTSHADOW($_iG_j$, $S_i$)
2:     $h \leftarrow$ SHA-512($_iG_j$)
3:     $mID \leftarrow h\ mod\ len(S_i)$
4:     $shadow \leftarrow S_i[mID]$
5:     return $shadow$
6: end procedure
```

Algorithm 1 shows the pseudocode of shadow email address selection. A hash function SHA-512 is used to get the digest of $_iG_j$, which then modulo with the size of S_i to select the current shadow email index, *mID*. Although from a compromised VIP user account, the adversary can achieve the ground truth $_iG_j$ to calculate *mID*, yet can not get the correct shadow email address because of not having the secret arrangement of S_i. Therefore, the adversary can not send an email with the right shadow email address, which immediately gets detected by the EMG.

2.5.3 Protocol

2.5.3.1 Communication between VIP users.

Figure 2.3 explains the EM protocol through email communication between VIP users Alice and Bob. (1) Alice composes an email to Bob *{from: alice@org.com, to: bob@org.com}*. (2) Alice's EMA delivers the email to EMG, where EMG uses the ground truth between them in algorithm 1, to select a shadow email address for Alice. Assume that the selected address is *alice.x@org.com*. Therefore, the EMG forwards the email to the mail server as *{from: alice.x@org.com, to: bob@org.com}*. (3) When Bob's EMA fetches for a new email, EMG receives the email from the mail server and deduce that the sender address is one of Alice's shadow email addresses. Therefore, EMG uses the ground truth between Bob and Alice in algorithm 1 to select the current shadow email address for Alice. If the retrieved address matches *alice.x@org.com*, the email is benign. Otherwise, it is phishing. EMG respond the benign email to Bob's EMA as *{from: alice@org.com, to: bob@org.com}*. Bob receives the email as how Alice composed it, making the whole EM mechanisms transparent to end-users. The replies from Bob to Alice is similar to the above three steps. The only secret in the protocol is the arrangement of the sender shadow emails.

2.5.3.2 Communication with External Users.

EM only protects VIP users. Therefore, the EMG bypasses the following emails with non-VIP users to decrease the overall email traffic processing:

No Mutation. A VIP user sends an email to a non-VIP user. Formally:

$$\{sender: x,\ recipient: y;\ where,\ x \in R\ and\ y \notin R\}$$

where R is the list of real email addresses of all VIP users.

No Verification. A VIP user receives an email from a non-VIP user. Formally:

$$\{sender: x,\ recipient: y;\ where,\ x \notin R\ and\ y \in R\}$$

2.5.4 Identifying Lateral Spear-phishing Attack

In EM, the legitimate email communication between VIP users happens through SCC, where the sender address is always a valid shadow email address. Therefore, EMG detects an incoming email as phishing while fetching new emails that have a real email address of a VIP member as the sender's address. However, the adversary may send phishing emails by guessing or randomly generating a shadow email address to bypass EM. Such phishing attempts is called as EM engineering attack. To formalize the detection process, let's assume that R is the real email address list, and $\overline{S}$ is the set of shadow email address lists of all VIP users.

Lateral Spear-phishing and Spoofing Attack. By compromising a VIP user's email account or impersonating a VIP user, the adversary sends a phishing email to another VIP user. Formally:

$$\{sender : x, recipient : y; where,\ x, y \in R\}$$

That means both the sender and receiver email address is enlisted in the VIP user list. EMG immediately detects such an email as a phishing email.

EM Engineering Attack. The adversary sends a phishing email to any VIP user by randomly *guessing* a shadow email address as the sender's address. Formally:

$$\{sender : x, recipient : y; where,\ x \in \overline{S}\ and\ y \in R\}$$

To evade EM, adversaries may randomly guess or mimic the mutation mechanism to generate a legitimate shadow email address. However, EM creates shadow email addresses from a space of at least 16 byte long alphanumeric sequence. Therefore, the probability of guessing a correct shadow email address is $1/2^{128}$, which is nearly zero.

2.6 Email Mutation Research Challenges

2.6.1 Handling Multiple Shadow Email Accounts

In EM, each VIP user has a set of shadow email accounts for sending emails to another VIP user. However, VIP users only discern about their real email account. Therefore, sending emails from multiple shadow accounts and keeping track of all sent emails into one real email account is challenging. EM overcome this challenge by following means. First, shadow email accounts are only used for sending emails while the receiver email address will always be the real email address. Therefore, each VIP user receives all emails into their real email account inbox. Second, while forwarding an email from a shadow account, the EMG uses an IMAP `APPEND` command to populate that email into the real email sent-box after a successful email delivery to the recipient. Thus, the real account gets a trace of email delivery. If an email gets bounced, the EMG sends the bounce email back to the real email account.

2.6.2 Improving Email Mutation Usability

Existing phishing detectors similar to EM mostly suffer because of low usability. For instance, PGP requires user training on public-key cryptography and the PGP tool itself [23,24]. PGP encryption removes the visibility of the email from end-users. Whereas, EM does not distort the generic user experience of using emails. Every operation such as mutation, verification, and shadow email address communication is entirely segregated from the end-users and processed by the cloud EMGs. Users only need to use EMAs, for instance, sending an email by pressing the new "Send email with mutation" button beside the regular "Send" button in the Gmail web client (`mail.google.com`) illustrated in figure 2.4a. EM does not modify or add anything in the email body or headers makes it transparent to mail servers as well. Therefore, EM can be used with any email service provider and email clients without further usability and configuration overhead. The transparency of EM also makes it compatible to work combining with other email security solutions (even with PGP [20] or S/MIME [21]) and cyber agility frameworks [47].

2.6.3 Preserving User Privacy

The secure cloud-based gateways in the EM does not violate the end-to-end user email privacy because of the following reasons. Firstly, EMG does not keep any copy of the email. It just either mutates or verifies the sender's email address if the communications happen between two VIP users. All other emails get bypassed. Secondly, EMA connects with EMGs through secure channels (SSL/TLS) to avoid unauthorized data access during transmission. Finally, the organization of the VIP members can maintain their own EMGs to preserve data privacy. The secret shadow email lists can not be retrievable from any EMGs. Therefore, in a cross-enterprise EM system, EMG from one organization can not reveal the shadow email list of another organization. In recent days cloud-based secure email gateways are becoming popular because of their swift, robust, and effective attack detection with minimal management overhead [48]. Therefore, many organizations are adopting such solutions from Cisco, Microsoft, Barracuda, Mimecast, and others [49].

2.6.4 Adding Custom Email Fields is Insufficient

Adding just a new field in the email headers (such as X-headers [50]) or custom trace on the email body for sender authentication will not help to detect the lateral spear-phishing attack. Because adding any extra information into an email will immediately eliminate its transparency to both the client and the mail server. Second, adversaries may corrupt such additional data by adding noises into it, which will cause an interruption in regular email communication, raising high false-positive rates, and opening new security loopholes into the detection system. Finally, motivated adversaries can craft such fields by carefully observing historical emails. To overcome these challenges, EM uses a random selection of the sender email address (shadow address) for each new email delivery without adding anything into the email. This makes the solution transparent to both end-users and mail servers, but hard for adversaries to guess a correct shadow email address.

2.6.5 Addressing Asynchronous Ground Truth Problem

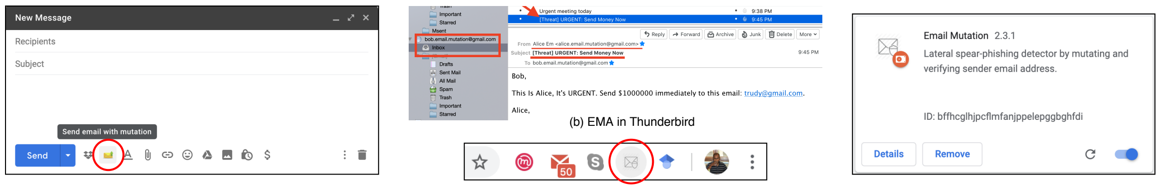
If a VIP user deletes any email from his inbox/sentbox that was sent by another VIP user, then the ground truth between them becomes asynchronous. To solve this problem, EMG keeps the hashed (SHA-512) digest of the last l number of emails between two VIP users. Therefore, EMG stores a maximum of $(v-1)$ hashed ground truth for each VIP user, where v is the number of all VIP users, to prevent asynchronous ground truth problems. Moreover, sender and receiver EMGs perform this hash storing operation asynchronously while sending and/or receiving an email. Thus, the *synchronization* does not require any communication between EMGs.

2.6.6 Handling Insider Attack

A novel contribution of EM is that it can protect VIP users from insider attacks. For instance, John is a VIP user who stoles Alice's email account credentials. Then, John uses his EMA to send a phishing email to Bob impersonating Alice. Formally, an attacker i compromises an email account j and uses i's EMA to send emails to k impersonating j, where $i, j, k \in L$ and L is the list of all VIP users. EM solves this problem by following: every VIP user's EMA is synchronized with its corresponding EMG instance through a unique authentication token (see section 2.7.1 for details). That means John's EMA can get only his instance of EMG; therefore, it will work only for John himself, not for Alice or Bob. The EMG keeps track who is forwarding the email by examining the authentication token of the EMA, and then verifies if that EMA is associated with the user or not. If the EMA is not assigned for that particular user but delivers an email anyhow, then EMG identifies that email as an insider attack.

2.6.7 Minimizing Shadow Email Account Overhead

The shadow email accounts are only meant to send emails as sender. These accounts do not receive any emails. Creating multiple shadow accounts for VIP users does not increase any inbound email traffic. Besides, these accounts do not impose any memory overhead. Therefore, they create negligible overhead on the service provider. The shadow email addresses can be selected from a 16-byte long (at least) sequence. This ensures no collision between shadow email with real email accounts. Additionally, unique keywords such as “sid" (section 2.2) can be used in shadow accounts creation to make a fine difference from real email accounts



Urgent meeting today 9:38 PM
[Threat] URGENT: Send Money Now 9:45 PM

From Alice Em <alice.email.mutation@gmail.com>
Subject [Threat] URGENT: Send Money Now 9:45 PM
To bob.email.mutation@gmail.com

Bob,

This Is Alice, It's URGENT. Send $1000000 immediately to this email: trudy@gmail.com.

Alice,

(b) EMA in Thunderbird

Email Mutation 2.3.1
Lateral spear-phishing detector by mutating and verifying sender email address.

ID: bffhcglhjpcflmfanjppelepggbghfdi

(a) Chrome Extension EMA adds new button for EM (c) EMA icon at Chrome Menu Bar (d) EMA description in Chrome Extension Panel

Figure 2.4: EMA in Google Chrome Browser and Thunderbird, (a) “Send email with mutation" button in `mail.google.com`, (b) EMA in Thunderbird, (c) EMA icon at Chrome Menu Bar, and (d) EMA in Chrome Extension Panel.

2.7 Scalable Implementation and Security Measurement

Evaluating EM in a large scale network requires a scalable implementation that is compatible with any existing email clients and email service providers. Moreover, the security measures of each component are necessary to reduce the risk factor in terms of user privacy and data breach. These matrices are measured for the primary components of EM: EMA and EMG.

2.7.1 Email Mutation Agent

Security Measures. EMA operates alongside with regular Mail user agent [51] or email clients only to deliver or receive new emails from EMGs. It does not communicate with the mail or SMTP server. Therefore, EMAs neither have any storage to collect emails nor requires the user email account credential. The communication between EMA and EMGs happens through a secure channel (SSL/TLS) to protect data breaches during the transaction.

Implementation. Usually, clients use multiple devices, such as cell phones, laptops, desktop, and more, to access their email accounts. Therefore, EMA needs platform-oriented implementation to work on different devices as well. Three types of EMA are implemented for three different platforms, 1) browsers extension for web clients that will work in laptops and desktop, where a web browser can run. 2) Shell and python scripts to configure email clients such as Outlook, Thunderbird, etc. and 3) email client (android/iOS) app for cell phones and tablets. Figure 2.4 shows different implementation of EMAs, such as browser extension (2.4a) and Thunderbird client (2.4b). The Chrome browser extension adds a new button called “Send email with Mutation" (figure 2.4a) alongside with the regular “Send" button that `mail.google.com` provides.

Distribution of EMA. A VIP user can get an EMA from their system admin or download it from the web. The admin gives a unique authentication token to each VIP user for their first use of EMA to subscribe with the EMGs. Later on, using that token, they can connect with their corresponding EMG from different EMAs. This ensures users' flexibility to use EMA from different devices and different locations (e.g., public networks). Users can reset the token anytime.

2.7.2 Email Mutation Gateway

Security Measures. EMG inspects email for detection and modifies for mutation. It does not maintain any mailboxes for users. When a VIP user subscribes with EMG, it creates an instance for that user. So that later on, the same user can connect with the EMG from EMAs in different devices. EMG keeps the arrangement of the shadow email lists secret. Therefore, from an EMA, VIP users can not retrieve their shadow email list. After a certain mutation interval t seconds, EMG rearranges all the secret shadow email lits of VIP users randomly. Besides, an instance of EMG given to a VIP user is not shareable by other VIP users through EMAs, meaning Alice can not use the EMG instance of Bob from her EMA. This protects the insider attacks because using her EMA and EMG instance, Alice can not send emails as Bob, considering that Alice compromises Bob's email account. Cloud-based solutions like EMGs are secured, and many providers like Amazon, Microsoft, Cisco, Barracuda, Mimecast, and more are nowadays providing secure cloud email gateways for phishing detection [49].

Implementation. The EMGs are implemented as an inbound and outbound Mail transfer agent [51] that works as an SMTP relay to mutate outgoing emails and proxy gateway to check incoming emails for spear-phishing detection. Python libraries such as `smtpd` and `imaplib` are used to implement the relay server and Django [52] framework to make EMG a web service. The code is available on GitHub.

2.8 Email Mutation Verification and Evaluation

2.8.1 Verification

EM is a new technique; therefore, it is necessary to ensure the design correctness before implementation and deployment over real network. Model checkers help to formally specify, model, and verify a system in the early design phase to find any unknown behavior if it exists. This section presents the modeling of individual components, their interaction, and verification of EM using model checking tool UPPAAL [53]. The comprehensive system is modeled using timed automata and verified against user-defined temporal properties. The components of EM have been illustrated by using the state machine diagram of the system modeling language, where the circle represents the state, and the arrow shows the transition among the states. The transitions are annotated with channels, guards, and actions. Interaction between components using channel $(!, ?)$ where, “!" and “?" means the sending and receiving signals, respectively.

Modeling of Client and Mail Server. Figure 2.5a illustrates the functionality of a client. The client uses the sending signal $\mathit{Send}(i, j)!$ to forward a newly composed email to the EMG, where i is the sender address, and j is the receiver address. Using $Fetch\_Email(j)!$, client j request to fetch any new email from EMG. As a response, the client receives a new email from EMG by the receiving signal $\mathit{Receive}(i, j)?$, where i is the sender address, and j is the receiver address. Figure 2.5b shows the basic functionality of a mail server. The channels $Send\_To\_Server(i, j)?$, $Fetch\_From\_Server(j)?$, and $Response\_From\_Server(i, j)!$ represent the transitions for receiving a new email, receiving fetching request for new emails, and re-

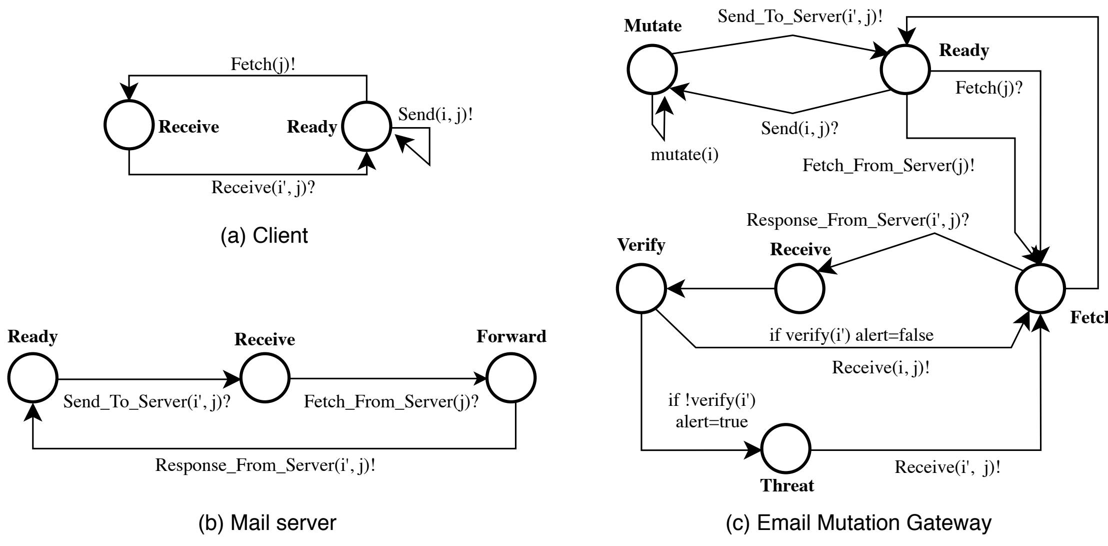


Figure 2.5: State machine diagrams of different components in EM.

sponding new emails respectively.

Modeling of Gateway. Figure 2.5c describes the functionality of the mutation gateway (EMG). EMG receives a new email from clients through the receiving signal $send(i, j)$? and mutates the sender address i to i' using the function $mutate(i)$. Then it forwards the email to the mail server using signal $Send\_To\_Server(i', j)$!. EMG enters the *Fetch* state after receiving a $Fetch(j)$ signal from client j and seeks new emails from the mail server by the signal $Fetch\_From\_Server(j)$!. As a response, EMG receives new emails by the receiving signal $Response\_From\_Server(i', j)$?, where i' is the (mutated) sender address, and j is the receiver address. After that, EMG verifies i' by the function $verify(i')$. If verification pass, then the email will be delivered to the client through the $Receive(i, j)$?, where i is the real address of i'. Otherwise, the email gets flagged as a threat. In case of suspicious email, the invariant *alert* is set to true; otherwise, it is set to false. Here, the state *Threat* is presented to flag the email.

Modeling of Adversary. Using channel $Send\_To\_Server(l, j)$? adversaries send email to recipient j, where l is the adversary chosen sender address. If adversaries

Table 2.1: Temporal properties to verify the correctness of Email Mutation system.

Property	**CTL**	**Result**
Reachability	$A[] := \forall\ i \in all\_emg$ $(emg_i.verify\ \wedge\ !emg_i.alert) \rightarrow (emg_i.ready)$	*satisfied*
Liveness	$A[] := \forall\ i \in all\_emg\ (emg_i.alert\ \rightarrow emg_i.threat)$	*satisfied*
Deadlock-freeness	$A[] := \forall\ i \in all\_emg\ (!emg_i.deadlock)$	*satisfied*

make a successful guess, their email will be delivered to the client.

Property Verification. UPPAAL takes the model, and user-defined temporal properties as input to verify the model and generates output, either satisfied or not satisfied. UPPAAL defines temporal properties in the form of computational tree logic (CTL), which is a branch of symbolic logic. To verify EM, the following temporal properties have been defined: liveness, reachability, and deadlock-freeness. Table 2.1 describes the properties along with its UPPAAL supported CTL and the results of these properties. *Reachability* describes that every good state is reachable, and every bad state is unreachable. In EM, every benign email should be delivered to the destination. *Liveness* describes the system is progressing to achieve a specific goal. For instance, every suspicious email should be flagged as a threat. *Deadlock-freeness* ensures that the system is not stopped, and it is always progressing. The system is in deadlock sate when it stops in a state and does not proceed to other states. The CTL operator $A[]$ represents that every single state on all paths should satisfy the properties, all benign emails are delivered to the destination and every suspicious email is flagged. The reachability property ensures that no suspicious email will be delivered without a threat flag. In reachability and liveness properties, $(\forall\ i \in all\_emg)$ is used for iteration and verification of property against every single email mutation gateway (emg_i).

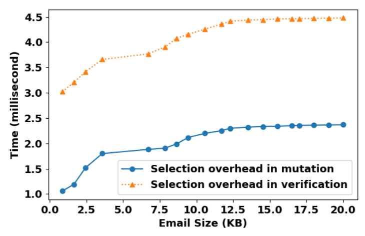
2.8.2 Evaluation

The performance of EM is measured in terms of overhead added into existing email infrastructure. EM has a 100% lateral spear-phishing and spoofing detection rate. EM is compared with similar existing solutions to measure the necessity and effectiveness of the system.

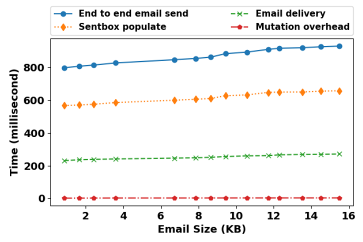
Experiment Setup. EM is evaluated in large scale enterprise networks for more than six months and protected 5,000 VIP members over five different organizations. Among them, 46 VIP member was voluntarily from Jet Propulsion Laboratory, NASA (JPL). The JPL red team sends more than half a million attack emails to evaluate the system. The VIP members use different mail service providers, including Gmail, Microsoft Exchange, Apple iCloud, and email clients like `mail.google.com`, Outlook, Thunderbird, and so on. The shadow email addresses for each VIP user was between ten to one hundred. The mutation interval t was set with different values between 60 seconds to 2 hours to rearrange the secret shadow email lists randomly. All evaluation metrics have been achieved from real-time email communications.

Shadow Email Selection Overhead. EMG computes shadow email addresses for mutation and verification using algorithm 1. The selection overhead is measured using a single email over different email sizes. Figure 2.6a shows the selection overhead for mutation is 2.5 milliseconds, and verification is 5 milliseconds for email size range 10-20KB without attachments. Figure 2.7a shows the overhead is 3 milliseconds, and 7 milliseconds for email size range 7-12MB with attachments.

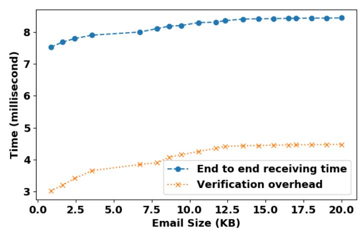
Mutation Overhead. While forwarding an email, EMG 1) mutates the email, 2) delivers the mutated emails to the provider mail server, and 3) populates the real email account sentbox of the sender to keep track of successful email deliveries. Figure 2.6b and 2.7b shows the overall forwarding delays over email sizes. Emails in 10-16KB sizes need 3 milliseconds for mutation, 250 milliseconds for delivery, 650 milliseconds for sentbox population, and overall 950 milliseconds to forward the email. For email sizes 7-12MB, mutation delay is 4.5 milliseconds, and overall sending time is 1.5 seconds. In both cases, the mutation overhead is 0.5% compared to the end-to-end email forwarding delay.



(a) Shadow computation.

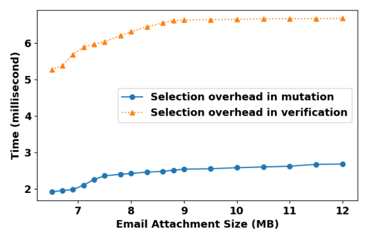


(b) Email mutation time.

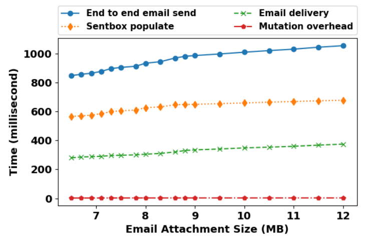


(c) Email verification time.

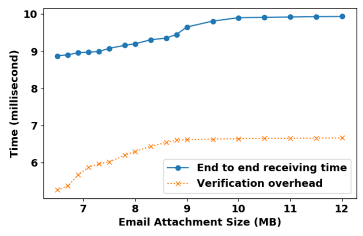
Figure 2.6: EM gateway performance for mutation-verification of individual emails without attachments.



(a) Shadow computation.



(b) Email mutation time.



(c) Email verification time.

Figure 2.7: EM gateway performance for mutation-verification of individual emails with attachments.

Verification Overhead. Figure 2.6c and 2.7c shows the end-to-end email receiving time with verification over different email sizes. Emails in 10-20KB sizes without attachments have overall 8.5 milliseconds receiving delay where the verification delay is 4.5 milliseconds, and emails in 7-12MB sizes have overall 10 milliseconds receiving delay where the verification delay is 7 milliseconds

Email Processing Rate by an EMG. The overall performance of an EMG is measured by sending more than thousands of emails at a time to a single EMG to process mutation and verification simultaneously. Figure 2.8 and 2.9 shows the average processing delay of EM for sending an email with mutation is 1.1 seconds and receiving an email after phishing detection is 10.9 milliseconds, respectively while dealing with 5000 emails per second.

Cross-Enterprise Architecture Overhead. Adding different organizations into the EM system does not increase additional overheads because the operational cost

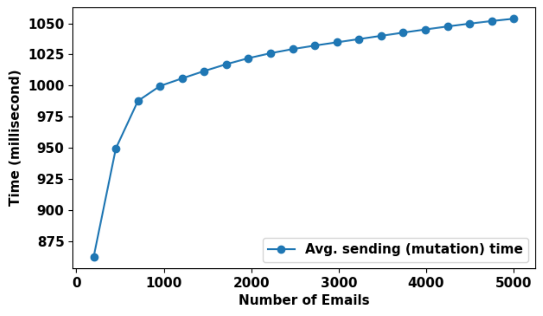


Figure 2.8: Multiple email processing overhead for mutation.

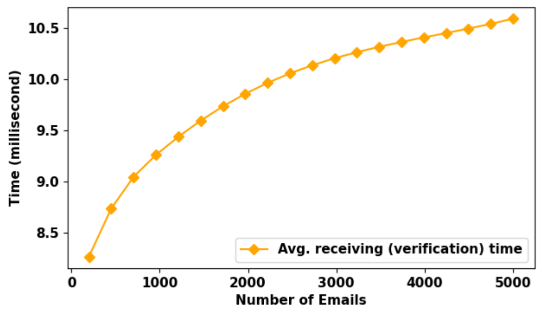


Figure 2.9: Multiple email processing overhead for verification.

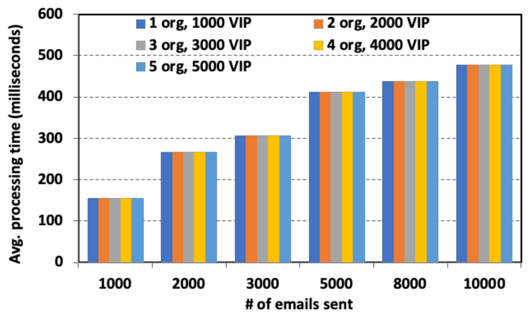


Figure 2.10: EMG overhead in cross-enterprise architecture. The increasing number of organization or VIP members do not impact the overall processing time. The number of emails dealt at a time determines the overall delay.

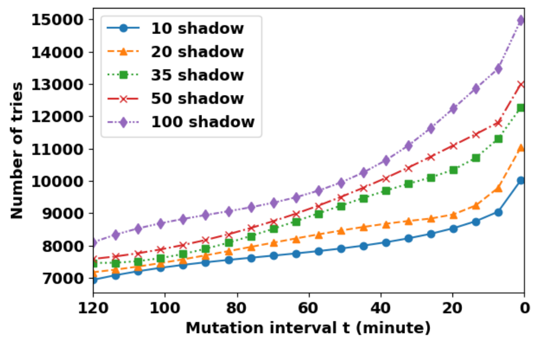


Figure 2.11: EM engineering attack, the minimum number of tries adversary needs to break the EM for sending their first successful phishing email.

of each EMG only depends on the total number of emails get processed at a time, not on the size of the VIP user list. Figure 2.10 depicts that the overall email processing time is the same for one organization having 1,000 VIP users to five organization having 5,000 VIP users. The delays increased when the total number of emails dealt at a time increases. Multiple EMGs can be used to balance these increasing delays.

EM Engineering Attack. If adversaries send a phishing email directly from compromised VIP user accounts or by impersonating a VIP user, they have 0% chance to evade EM. However, adversaries may try to phish by randomly guessing a shadow email address, known as EM engineering attack. A shadow email address is a 16-byte long random alphanumeric sequence, which is practically impossible to guess. There-

Table 2.2: Detection results of EM.

Metric	**Data**
Total attack emails	516,000
Lateral spear-phishing attack	153,207
Spoofing attack	145,291
EM engineering attack	201,437
Integrity attack	16,065
EM engineering attack missed	3
Integrity attack missed	167
L. spear-phishing detection	100%
Spoofing detection	100%
EM engineering detection	99.99%
EM engineering false negative	0.0015%
Integrity attack false negative	1.04%

fore, for the sake of the evaluation, all valid shadow emails are informed to the red team before launching the attack. Figure 2.11 depicts the detection results. With different setups of mutation parameter, the minimum number of tries adversary needs to break EM for sending their first successful phishing email varies. For instance, it takes 7,000 tries to send the first phishing email if a VIP user has 10 shadow emails and 120 minutes mutation interval. However, the tries dramatically increase to 14,500 (probability 0.000069) if the number of shadows and mutation interval changes to 100 and 1 minutes, respectively. This indicates that based on user impact, EM can increase the level of protection swiftly.

Detection Results. Table 2.2 summarizes the performance metrics for EM. From a total of 516,000 attacks, EM detected all of the lateral spear-phishing and spoofing emails with no false positive and false negative rates. Out of 201,437 EM engineering attacks, three were successful, as I inform prior to the attack generator red team about all valid shadow email addresses. The purpose of EM is to detect lateral spear-phishing and spoofing attacks. However, EM can detect any integrity violation in the email while two VIP user communicates. EM calculates the current shadow email address based on the hashed value of the email and prior l emails (ground

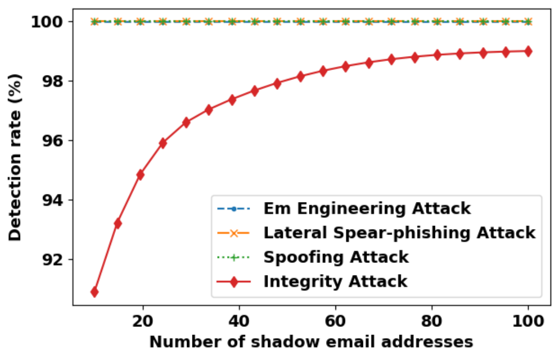


Figure 2.12: The detection rate of EM over different shadow email address assigned to a VIP user. Lateral spear-phishing and spoofing detection rate is 100% for all values of shadow email addresses. The integrity attack can be detected with 99% accuracy by using 100 shadow email addresses.

truth). Therefore, any changes in the email during transmission will desynchronize the ground truth at the receiver side. Figure 2.12 shows the integrity attack detection rate is 99% when the shadow address for a user is one hundred. Although this attack is explicitly out of our attack model, however, I add this into the detection results to show the capabilities and completeness of EM.

Comparison with Existing System: *Learning-based.* Existing lateral spear-phishing detectors are mostly learning-oriented; they learn attack signatures, benign users' behavior from the historical data, and create a model to detect phishing emails [10, 15, 38, 40, 41]. These solutions require myriad historical data for training, distinct attack signature (e.g., malicious URL or attachment), sufficient number of features, and often shows high false positive and false negative rates because of any lacking of these requirements. False positive means benign emails detected as phishing and false negative means a phishing email detected as benign. For instance, Ho et al. [10] trained their model over 25.5 million emails, and their attack model is limited to malicious URLs embedded in the email body. Gascon et al. [38] showed high false positive (10%) and high false negative rates (46.8%) against detecting lateral spear-phishing attacks. EM does not require any training; it is independent from

Table 2.3: Comparison between existing popular PGP tools and EM.

Tool	Overhead (milliseconds)			
PGP	GnuPG	Autocrypt	Enigmail	Mailvelope
	760.356	680.65	852.12	785.65
EM	Mutation	Verification		
	3	7		

Table 2.4: Existing email authentication standards failed to detect lateral spear-phishing (LSP) attacks.

Auth. protocol	**Attack data**	**LSP detect**
SPF	100%	0%
DKIM	100%	0%
DMARC	100%	0%
EM	N/A	100%

email content or header and can detect lateral spear-phishing and spoofing attacks with zero false positive and false negative rates.

Agent-based. PGP, S/MIME can ensure sender authenticity by digitally signed the email. However, these solutions are not widely used because of low usability, low transparency, and high manageability issues [23, 24]. The end-users require prior knowledge regarding public-key cryptography and proper training to use PGP tools. The encrypted cyphertext eliminates the visibility of the emails, making it incompatible to work with other security extensions such as IDS. Moreover, PGP signatures can be spoofed [26]. In contrast, EM is transparent, has a low management overhead, and highly flexible to use. The end-users do not need any prior knowledge or training to use it. Table 2.3 shows a comparison of EM with the existing popular PGP solutions in terms of overhead added in a generic mail transfer system. The overhead of EM is negligible (3-7 milliseconds) compared to PGP signature and encryption operations with RSA 2048 bit keys, which is vital for large scale enterprise networks where email processing rate is higher.

Authentication-protocol. Standard email spoofing detection protocols such as SPF, DKIM, and DMARC can not detect lateral spear-phishing attacks. Table 2.4 depicts that, all of our attack data has these security extensions. However, the lateral spear-phishing emails bypass these standard techniques as they are sent from legitimate accounts. Nonetheless, EM detects them all. Therefore, in the prevailing lacking of alternative protection against lateral spear-phishing attacks, the EM system is a valuable extension to existing defenses.

2.9 Summary

This chapter presented a novel approach using sender email address mutation to proactively defend against the most devastating and stealthy spear-phishing called lateral spear-phishing attacks. Our EM system guarantees the phishing emails sent from trusted users will be immediately detected. EM integrates well with existing email infrastructures, and it requires no special handling by users. EM requires an agent to be deployed on the client-side for every user and a central gateway in the cloud. The agent can be a simple plugin installed in email clients. I implemented and evaluated EM in a large scale real-world enterprise network with well-known email service providers (Gmail, for example). The evaluation showed that EM causes 0.5% overhead on overall email transmission while detecting lateral spear-phishing and spoofing attacks. Moreover, I showed that it is very hard to break EM (probability 0.000069). Unlike the existing spear-phishing detectors, which are limited on malicious content or links, our EM can work beyond email content and headers to detect most stealthy lateral spear-phishing attacks that exploit compromised email account.

CHAPTER 3: Active Cyber Deception and Agility Synthesis

Dynamic composition for security planning necessitates a framework to enable cyber agility and deception into the system for proactive defense. Therefore, the purpose of this chapter is to build the core of the framework that leverages the automatic creation of defense actions rapidly and safely. This chapter focuses on developing a high-level cyber agility policy language specification and distributed controller architecture for implementing configuration-based MTD and deception techniques.

3.1 Motivation

In modern cyber warfare, the prevalence of cyber asymmetry between the adversary and the defender is that the defender needs to protect all susceptibilities into the infrastructure. In contrast, the adversary requires few vulnerabilities to exploit. Existing reactive cyber defense techniques response after the attack has been launched and usually approaches the target for specific, known attack descriptions or signatures [54]. This often lets the adversary remain stealthy enough to learn the system and discover further vulnerabilities and possible lateral movement. In addition, skilled attackers can easily avoid static signature-based detection through exhaustive reconnaissance, fingerprinting, and social engineering [28]. Therefore, a proactive approach must be used by defenders to break this asymmetry. Active Cyber Defense (ACD) is a promising technology to achieve this goal. ACD can be leveraged through cyber agility such as moving target defense (MTD) and cyber deception.

Cyber agility allows the system to defend proactively against a wide-scale vector of sophisticated attacks by dynamically changing the system parameters and defense strategies in a timely and economical fashion. It can provide robust defense by deterring attackers from reaching their goals and disrupting their plans via changing adversarial behaviors. Cyber Deception is an intentional misrepresentation of real systems' ground truth to manipulate adversary's course of actions under the premises of the defenders' rules [9]. However, the goal of cyber deception is beyond just to mislead adversaries. Cyber deception can *deflect* adversary away from their target to false or no target, *distort* their perception of the infrastructure by adding ambiguity and decoys into the network. Deception can *deplete* adversary consuming their computational power, delaying attack propagation by storming the static ground truth to a probabilistic state, for instance, increasing the number of probing by mutating the static IP addresses of critical resources. Finally, deception can engage with the adversaries to stir down under defenders' premises to *discover* their hidden tactics and techniques, which leads the defender to protect future zero-day attacks. In summary, deception is a rising paradigm in cyber defense that works beyond traditional detect-then-prevent techniques while effectively discovering new adversary tactics, consuming their resources, slowing down attack propagation, and learn adversary intention for future defense.

The current state of art depicts the increasing adoption rate of ACD because of its evident success over the existing reactive defense [9,28,55–60]. By 2022, it is expected that the global cyber deception market's expense will grow up to \$2.3 billion [61]. However, developing ACD techniques in real networks is a highly complex task. It requires significant effort in implementation and network configuration management. Efficient and adaptive deception and MTD needs continuous network monitoring to observe adversary activities, optimal planning for feasible implementation, and safe deployment without breaking the integrity of the system. As a result, few ACD frameworks are developed and validated in the real-life operational environment.

To overcome these challenges, an ACD framework (figure 3.1) is developed in this chapter. The framework has two part, MTDSynth for cyber agility synthesis and an extensible rich API to build sophisticated MTD and cyber deception applications. The goal of framework is to make MTD and deception infrastructure as services through API that shields the defender from all intricate details about the low-level primitives implementation, orchestration and deployment, which eventually block them from developing novel and innovative defense applications.

MTDSynth allows Agility developers for creating Agility control programs using a high-level cyber agility policy language (HAPL) that provides the required constructs for configuration-based Agility techniques including the following: (1) *mutation triggers* which can be time-based or event-based using user-defined network *sensors*, (2) Agility *mutable system parameters* that will be dynamically changed based on the trigger, (3) *configuration parameters* that depend on the mutable parameters, (4) *mutation functions* and *mutation constraints* that dictates the methodology to compute and optimize the selection of new mutation values, (5) *mutation attributes* that can be used to define the mutation scope or domain.

The API creates a new dimension to make cyber deception and MTD as a service. It is extensible and available on GitHub; therefore, more functions can be added by the community based on requirements. The high-level API specifies the HoneyThings (honeypots, decoys, etc.) [9] configuration parameters and misrepresentation mechanisms of the system. While deployment, these parameter configurations can be done by low-level network management API. Thus, the framework has a comprehensive low-level network management API. The high-level security API and low-level network configuration API make the framework an easy to develop multi-strategy ACD composition and deployment in a timely and economical fashion.

The framework is developed over Software-defined Networking (SDN). SDN provides a robust mechanism for dynamic and disruptive network management. SDN enables programmatic ability into the network configurations. The fundamental facility achieved from SDN is to manage the network configuration dynamically from a central controller for quick response and diagnosis. The framework offers the API through ActiveSDN, a SDN controller developed over OpenDaylight [62] to facilitate the synthesis process. ActiveSDN supports the implementation of the sensors and agility actions defined in the HAPL at a high level without the need to focus on any low-level OpenFlow [63] configuration. In addition, ActiveSDN incorporates the Satisfiability Modulo Theory (SMT) constraints satisfiability solver [64], to optimize the Agility and agility actions (selection of parameters and configurations values) at real-time.

3.2 Problem Statement

Given ACD requirements in terms of deception and MTD techniques, the goal is to define a specification language and a rich set of API that can be used to design multi-strategy ACD techniques and deploy them into the network safely and rapidly.

Specification Language: The high-level specification language will give structured instructions on how to use the HoneyThings in order to create several ACD policies. The ACD framework provides an extensible list of MTD and deception API. However, to use them properly in order to create a complete deception or MTD game without breaking the mission integrity, a structure should be maintained. Using the language, ACD developers can create multilevel deception and MTD actions without consulting low-level network or system configurations.

HoneyThings: HoneyThings consist of honey-registry, honey-files, honey accounts, honey traffic, and fake configurations that resemble real system resources but are actually fake. The purpose of HoneyThings is to mislead the adversary away from the real target. The factory that can manage the HoneyThings is known as HoneyFactory.

3.3 Related Works

To illustrate the framework, the following existing MTD techniques are used: (1) Temporal Random Host IP Mutation (Temporal RHM) [65], (2) Spatial Random Host

IP Mutation (Spatial RHM) [66], (3) Random Route Mutation (RRM) [67], and (4) MTD to disrupting stealthy bots [68].

Temporal RHM. Temporal RHM can turn end-hosts into untraceable moving targets by mutating their IP addresses in an intelligent and unpredictable fashion without sacrificing network integrity, manageability or performance. In RHM, moving target hosts are assigned virtual IP addresses that change randomly and synchronously in a distributed fashion over time without disrupting active connections.

Spatial RHM. In spatial RHM, to reach each destination host h_j, each source host h_i is associated with an ephemeral IP (eIP), such that this eIP could be only used by h_i to reach h_j. The distribution based on which these new mappings are determined can be either *uniform* or *deceptive (adversary-adaptive).* The mutation uses a strategy selection algorithm to determine the appropriate way at any given time by analyzing the behavior of potential adversaries in the network.

Random Route Mutation (RRM). RRM allows for switch routes in the network periodically or based on feedback from network monitors. The main goal of RRM is to change the route between a given source and destination address randomly to disable the attack capabilities to launch an effective eavesdropping or DoS attacks on the specific node or link in the route.

Disrupting Stealthy Bots. Disrupting Stealthy Bots is a MTD approach for placing detectors across the network in a resource constrained environment and dynamically and continuously changing the placement of detectors over time to defend against stealthy bots.

The notion of mutable networks as a frequently randomized changing of network addresses and responses was initially proposed in [69]. The idea was later extended as part of the MUTE network which implemented the moving target through random address hopping and random fingerprinting [70].

Existing IP mutation techniques include Dynamic Network Address Translation (DYNAT) [71–73], Applications that Participate in their Own Defense (APOD) [74], Address Routing Gateway (ARG) [75], Network Address Hopping (NAH) [76], Random Host IP Mutation (RHM) [65], OpenFlow Random Host IP Mutation (OF-RHM) [77], etc.

DYNAT is a technique developed to dynamically reassign IP addresses to confuse any would-be adversaries sniffing the network. They obfuscate the host identity information (IP and Port) in TCP/IP packet headers by translating the identity information with preestablished keys. BBN ran series of red-team tests to test the effectiveness of DYNAT, while Sandia's DYNAT report [72,73] examines many of the practical issues for DYNAT deployment.

Spatio-temporal Address Mutation (STORM) [66] can defend against collaborative scanning worm and APT attacks. It can distort attackers' view of the network by causing the collected reconnaissance information to expire as adversaries transition from one host to another or if they stay long enough in one location.

The work in [67] provided a general formalization for RRM with various operational and QoS constraints. The route selection is random and the new constraints can be added conveniently. In practical networks, the number of disjoint paths is usually very small [78], so the work in [67] analysed the MPE with non-disjoint paths for RRM. The work in [28] presented a cyber deception framework, called CONCEAL, as a composition of mutation, anonymity, and diversity to maximize key deception objectives.

Developing cyber agility framework is a complex task because of the automatic yet fast orchestration and management of network configuration without breaking the mission integrity [79, 80]. Agility framework requires comprehensive metrics for the safe deployment of mitigation techniques [81–83].

The current state of the art in cyber deception is enriched due to its effectiveness over existing reactive defense policies. Different theories and approaches are being used for optimized and powerful cyber deception planning. Game-theoretic approaches such as partially observable stochastic games [58], zero-sum Stackelberg game [84], Bayesian game [85, 86] provides adversary adaptive deception.

Sequential decision making such as Partially Observable Markov Decision Process (POMDP) [87, 88], game theory with Q-learning [89], and MDP [90] provide interactive cyber deception planning with the adversary in real-time. Probabilistic logic deception (PLD) [57], satisfiability modulo theories (SMT) [28,56] provides optimized deception planning.

Deception framework like [28, 55, 91] provides a practical implementation for optimizing complex deception plannings. However, these solutions do not focus on building a multi-strategy deception goal simultaneously. Software-defined networking (SDN) has been used for developing many cyber deception techniques for network securities against reconnaissance attacks [28, 47, 55, 59, 92–94].

3.4 Active Cyber Defense Framework

Figure 3.1 shows the major components of ACD framework: the user interface to initiate cyber deception or MTD and the active defense controller (ADC) to deploy the deception or MTD along with the OpenDaylight ActiveSDN controller over the SDN network. The Chimera is the deception action pallner, which is described in chapter 4.

3.4.1 Interface

The interface provides an easy way to play with the deception and MTD API (ADC API) to build highly complex defense policies. For instance, the admin wants to

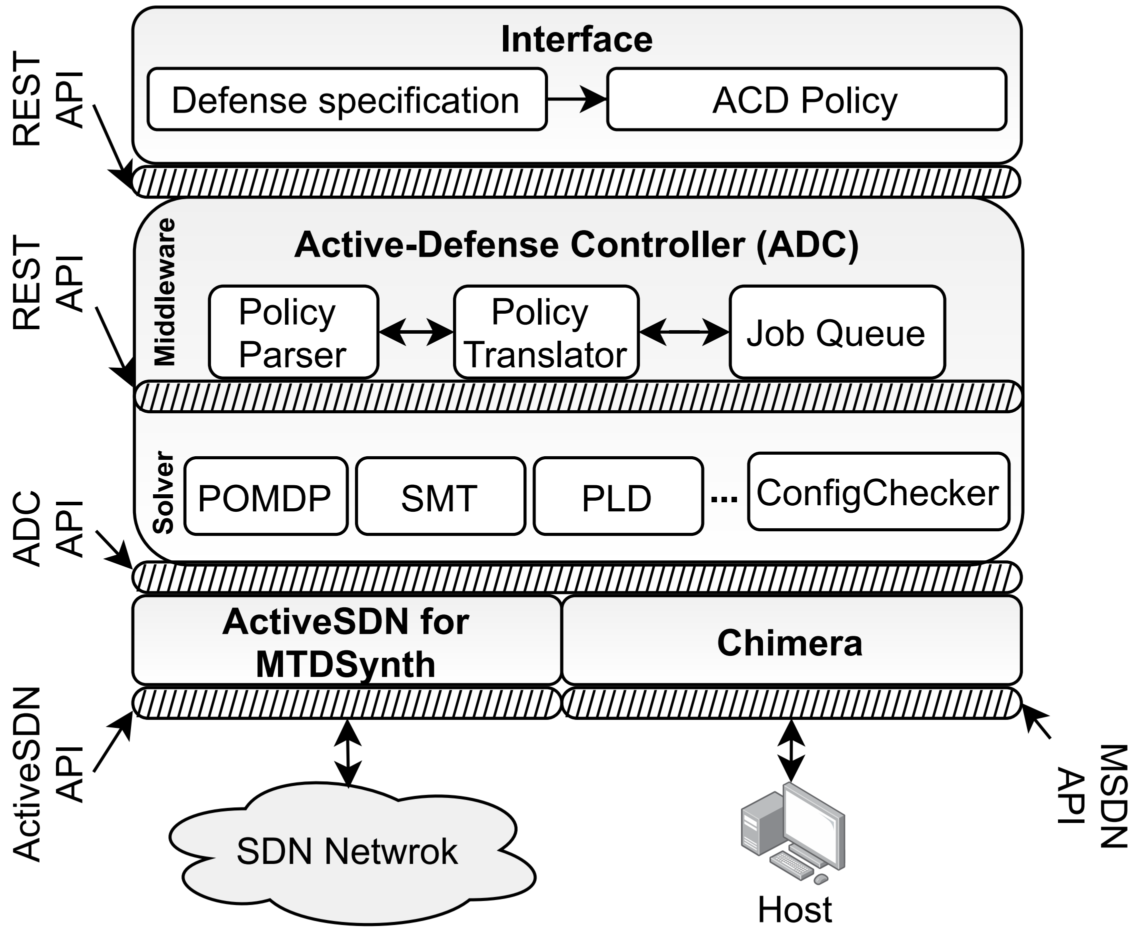


Figure 3.1: MTD Controller Synthesis using MTDSynth.

initiate a deception strategy to proactively protect critical resources into the network while also wants to learn the network behavior for any future attacks. Therefore, she starts a multi-strategy deception technique, such as distort adversary, by creating anonymity and diversity into the network while diverting malicious traffic to a decoy machine for unknown attack tactics discovery. Such a multi-strategy deception can be launch swiftly and effortlessly using the interface shown in listing 2. Listing 2 depicts a JSON request for launching deception by the API *createHoneyNetwrok()*. The details of the JSON and the API is discussed in section 3.4.2.5. The interface delivers the deception triggering JSON request to the middleware through REST API.

Listing 2 JSON request to launch deception by *k-anonymity* and *l-diversity* through API *createHoneyNetwork()*.

```
{"input": {
        "api": "createHoneyNetwork",
        "target": "r1",
        "impact": "high",
        "k": 2,
        "l": 3,
        "trigger": "activate"
    }
}
```

3.4.2 Active Defense Controller

The Active defense controller (ADC) in the framework is the central orchestrator that handles the end-to-end processing of the deception and MTD techniques from initiation by the interface to the safe deployment in the network. It provides an agility language specification HAPL to create agility control programs (agility policies for MTD and cyber deception). ADC enables an open playground that enables prototyping or building advanced ACD techniques rapidly and safely using SDN. ADC leverages its facilities by providing a ADC API that gives access to sophisticated cyber deception, MTD and OpenFlow management functions using the OpenDaylight controller. Besides, ADC incorporates with a decision-making synthesis engine called solver, that is capable of solving computationally hard problems using constraint satisfaction solvers (SMT Z3) [27], ConfigChecker [29], etc. to optimize deception policy actions. ADC composes the deception triggering by the interface, ensure safe low-level configuration changes, and deploy the ACD technique into the network. Therefore, ADC makes ACD technique as a service that users can access without taking any low-level configuration management headaches yet mitigate attacks proactively.

3.4.2.1 Middleware

The middleware translates the high-level ADC API to ActiveSDN (OpenDaylight) API. It incorporates the solver through REST for solving constraint problems in order to optimize deception planning. Middleware creates a back-and-forth communication

Listing 3 Middleware generates the ActiveSDN (OpenDaylight) API deception event from user input (Python implementation).

```
class ActiveDeceptionApiHandler:
    def event_generator(self, request):
    ...
        deception_event = EventBuilderFactory
                        .build(request)
        output = ActiveDeceptionServiceImpl
               .defenseByDeception(deception_event)
```

Listing 4 ADC solves optimal deception planning configuration to create honey network (OpenDaylight Java implementation).

```
public class ActiveDeceptionServiceImpl implements ActiveDeceptionService
    ...
    public Future<RpcResult<DefenseByDeceptionOutput>>
        defenseByDeception(DefenseByDeceptionInput input){
        ...
        DeceptionPlan optimalConfiguration = Solver.solve(input.getConstraints())
        Future<RpcResult<CreateHoneyNetworkOutput>> status =
            createHoneyNetwork(optimalConfiguration);
```

bridge to ADC with the user interface through ADC API and the SDN network through ActiveSDN API. The middleware is developed in python and run as a daemon server along with the ADC controller. The middleware uses a factory pattern to interpret user input and create an event correspondingly, shown in listing 3. Then it invokes the ADC service, which implements the ActiveSDN API to launch the deception or MTD techniques. Listing 4 shows the ActiveSDN implementation of the deception deployment.

3.4.2.2 Policy parser

The policy parser is the first step to process the MTD policy created by the interface. It parses the given policy according to the language specification, generates a parsing tree and checks any syntax error. Then the parse tree is delivered to the translator. If any error occurs while processing the policy, the parser provides feedbacks to the interface; therefore, the user can make necessary correction in the policy. The MTD policy parser also measures the semantic correctness of the given policy from the parse tree. It checks whether the policy uses any primitives, sensors, mutation functions, or constraints that are not supported by ActiveSDN. Moreover, it checks the correct use of arguments in different primitives, type mismatch while comparing the output of different primitive constrains with each other, etc. If the validation fails, the reason behind the failure is returned to the interface as feedback.

3.4.2.3 Policy translator

The translator module translate the MTD policy to a python script with the ActiveSDN API. The ADC expose the primitives to create MTD policy from all the actions, sensors, and constraints ActiveSDN engine has in the ActiveSDN API. The translator generates a complete python script from the given policy based on that API, therefore executing the script will deploy the policy into the network through ActiveSDN engine. The translator knows how to invoke each function the engine provides correctly and by correct, it means the proper argument selection, providing appropriate response to any notification, etc. The script can run as a daemon server to communicate back and forth with the engine. The reason behind that, some API will generate output that can be used as input to another API (e.g., the output of *getCriticalLink()*, *l* can be used as input in *isLinkFlooding(l, ...)*). Thankfully, ActiveSDN engine is developed over OpenDaylight, which supports multi threading to handle multiple request at a time from the translator program

3.4.2.4 Solver

The solver in ADC is to optimize constraint problems to generate a feasible and practically deployable ACD configuration. The solver is designed as a plug-in-play model in the architecture. Therefore, various configuration optimization solution such as POMDP [30], SMT [27], and more can be added with ADC as required. The SMT solver is used to optimize anonymity and diversity of concealment configuration [28], ConfigChecker to solve reachability constraints [29]. ADC incorporates with the solver through middleware via REST API.

Table 3.1: Active cyber defense framework sensors, management and constraint API.

Sensor API	**Management API**	**Constraints API**
isHostScanning()	block()	getRouteRisk()
isLinkFlooding()	inspect()	overlap()
chekTrafficRate()	throttling()	isIncludeSwitch()
checkElephantTCP()	splitInspect()	getAvailableBandWidth()
getFlowStatistics()	priorityForwarding()	checkUniqueIP()
checkNewComers()	installFlowRule()	checkNonRepeateIP()
getCriticalLinks()	installNetworkPath()	checkSpatialCollision()
getAllFlowRules()	sendPacketOut()	getMinDetectionProb()
findNeighbors()	createTunnel()	getAttackUncertainity()
detectBot()	subscribeEvent()	canReach()
getPortID()	removeAllFlows()	getShortestPath()

Table 3.2: Active cyber defense framework sensor, management and constraint API description.

	Name	Descriptions
Sensors	isHostScanning()	If any IP address sending SYN packets grater than the threshold th in a certain time window t, this function generates a true positive alarm.
	isLinkFlooding()	If the bandwidth consumed by the flows going to that link l, is greater than the the bandwidth threshold th, this will generate a true positive alarm.
	chekTrafficRate()	Calculate the average rate of a specific flow f of type (UDP and ICMP) flows.
	checkElephantTCP()	Calculate the percentage of large-size TCP traffic from a given flow list <f> .
	getFlowStatistics()	To get the complete information about a flow f such as: number of packets matched with f, bytes captured by f, time window for those packets, traffic type (ICMP, TCP, UDP) etc.
	checkNewComers()	Calculate the ratio of new IP source addresses from a given flow list <f> that has not been seen before recently in a given time window t.
	getCriticalLinks()	This function returns the critical links may generated in the topology based on the flow data path.
	getAllFlowRules()	This function retrieves all the flow rules available into a switch s.
	findNeighbors()	Returns all the neighbor hosts of the given IP address h that connected with the given switch s.
	detectBot()	If the signature of the examined packets satisfies the condition of bot traffic, return true.
ADC API	createHoneyNetwork()	Dynamically creates a honey network with decoy/shadow hosts and services to analyze adversary for unknown TTP discover or distort them to delay attack propagation.
	reDirect()	Redirect traffics to a given destination (can be a decoy or false target) and tunnel the packet to a proxy to generated trusted response.
	reRoute()	Change the old path between a source and destination pair to a new path to avoid possible link flooding or other security measures.
	pathMutate()	Change the path frequently of active flow(s) to another satisfiable path based on event or time.
	ipMutate()	Randomizing real src/dst IP addresses to virtual src/dst IP addresses for depletion, so that real IP is used for routing but end hosts always uses virtual IP to communicate.
	migrateService()	Create new machine with same services of the current target then migrates all benign traffic to the new machine.
	spatialMutation()	Randomize the real IP of given hosts so that each host reach the same destination with a different IP address. Therefore, the view of the network is different for different host.
Management API	block()	Block any incoming traffic from the given host IP address.
	inspect()	Inspect header and limited prefix data (application header), can be redirected to given IDS for advance inspection.
	throttling()	Policing the rate (traffic shaper) of the given traffic.
	splitInspect()	Divide the traffic and apply deep packet inspection.
	priorityForwarding()	Use priority queuing to forward traffic (gold, bronze and silver services)
	installFlowRule()	This function installs a single flow rule into a switch.
	installNetworkPath()	This function installs a complete data flow(s) path between source and destination host.
	sendPacketOut()	Forward the packets to the given destination (switch, host or controller).
	createTunnel()	Creates a tunnel where gateways at both ends only changes the source IP address to new IP address so that man in the middle get false information about source IP address.
	subscribeEvent()	Subscribes an event. An event can be a flow trace having type TCP, particular rate or given src-dst pair.
	removeAllFlows()	Removes all flow rules from a switch.
Constraints API	getRouteRisk()	Calculates the risk of a route based on the probability of each risk get attacked.
	overlap()	Calculates overlapping links between two route.
	isIncludeSwitch()	Checks if a particular switch available in a route.
	getAvailableBandWidth()	Checks the assgined bandwidth of a link.
	checkUniqueIP()	Checks if two sets of IPs are unique or not.
	checkNonRepeateIP()	Checks if the given IP is not used by other already assigned IP.
	checkSpatialCollision()	Checks the collisions in eIP in spatial mutation,
	getMinDetectionProb()	Calculates the minimum probability to detect an event based on given signature or specification.
	getAttackUncertainity()	Calculates the probability of a host, link or switch in a route that will be attacked with a given probability.
	canReach()	Find all reachable sources or destinations to/from a specific source s and destination d.
	getShortestPath()	Calculates the shortest path source s and destination d.

Table 3.3: Active cyber defense framework Deception and MTD API

Name	Descriptions
createHoneyNetwork()	Dynamically creates a honey network with decoy/shadow hosts and services to analyze adversary for unknown TTP discover or distort them to delay attack propagation.
reDirect()	Redirect traffics to a given destination (can be a decoy or false target) and tunnel the packet to a proxy to generated trusted response.
reRoute()	Change the old path between a source and destination pair to a new path to avoid possible link flooding or other security measures.
pathMutate()	Change the route frequently of active flow(s) to another satisfiable route based on event or time.
ipMutate()	Randomizing real src/dst IP addresses to virtual src/dst IP addresses for depletion, so that real IP is used for routing but end hosts always uses virtual IP to communicate.
migrateService()	Create new machine with same services of the current target then migrates all benign traffic to the new machine.
spatioTemporalMutation()	Randomize the real IP of given hosts so that each host reach the same destination with a different IP address. Therefore, the view of the network is different for different host.
createShadow()	Creates an identical fingerprint (shadow) of a given host in the honeypot.
createDecoy()	Creates a decoy host. If the decoy is specified for a target host without specifying any services, then arbitrary but the same type of services will be created in the decoy, e.g., an FTP server but with a different vendors.

3.4.2.5 ADC Deception and MTD API

ADC provides a comprehensive API for developing complex and multi-strategy deception and MTD techniques. The API list is divided into four classes: a high-level public API for deception and MTD known as ADC API. The other three are low-level APIs. Sensor API for collecting network behavior to observe adversary actions. Management API to configure cyber resources such as switches, links, hosts, services, etc. The constraint API calculates risk, overlaps, reachability, availability while configuring honey networks. Table 3.1 shows the low-level API list, and table 3.2 describes their functionalities.

The novelty of framework is the extensible ADC API that can be used to build sophisticated multi-strategy deception and MTD techniques in a cost-effective and timely fashion. It eliminates the challenges of deploying effective cyber deception policies that require frequent but complex low-level network configuration management. Because of the robust and expressive ADC API, deception and MTD management is orchestrated automatically with minimal overhead. Table 3.3 describes a selective list of active deception API.

3.5 Agility Policy Language specification

The language specification allows defender to program MTD techniques in high-level and the ADC will deploy that technique into the network *safely* by orchestrating

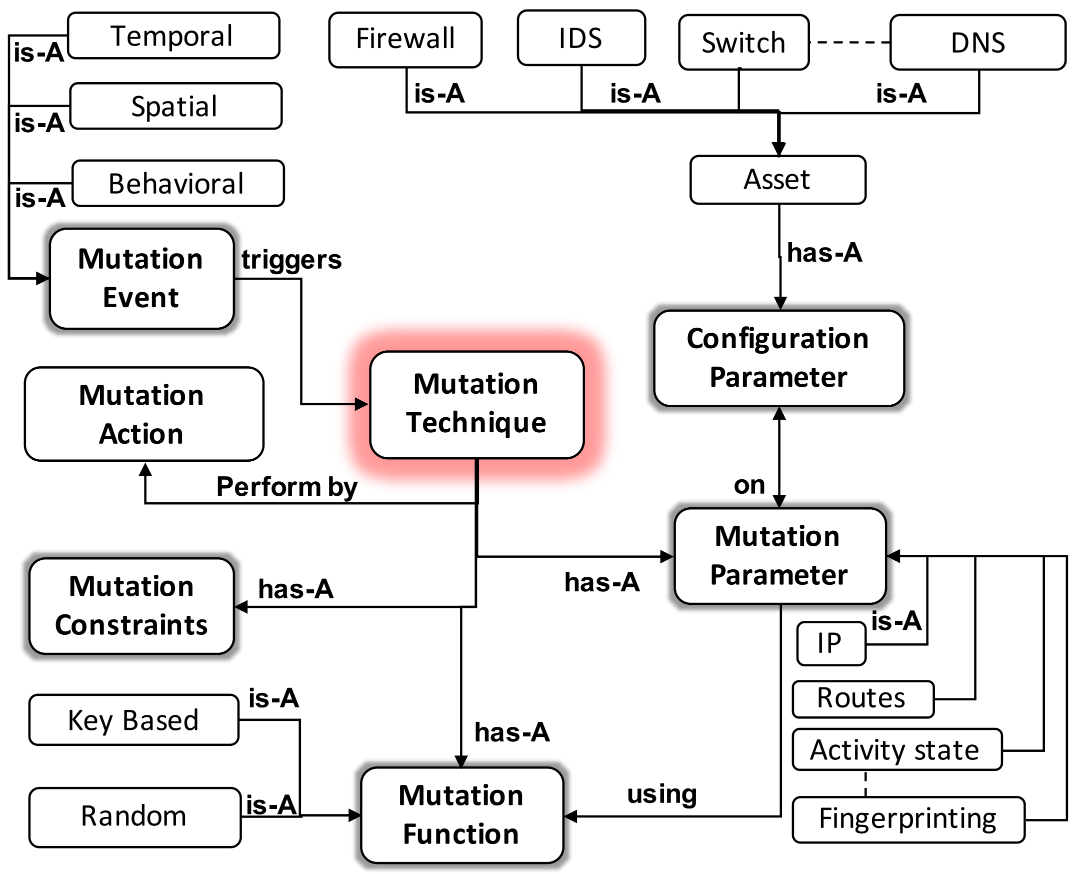


Figure 3.2: Cyber agility policy ontology for MTDSynth.

low-level network configurations (e.g., DNS record, flow rules in switch tables) automatically. By *safely*, it means that the deployment of MTD techniques doesn't violate the mission integrity as the automatic configuration of such sophisticated techniques can jeopardize the reachability, liveness, and fairness of the network communication. Therefore, MTDSynth provides minimum effort in the development of MTD techniques and puts safeguard onto it by considering the constraints. To define a fine grain MTD specification, an ontology is needed that can best describe the formal syntax of MTD techniques creation. Therefore, I developed an MTD ontology and an MTD language syntax around it.

3.5.1 Ontology

Figure 3.2 shows the ontology for the MTD techniques. The mutation technique will be triggered by an event which can be temporal, special, or behavioral event. The temporal or spatial events are time oriented, meaning after a certain period, the mutation techniques will be triggered or continue triggering. Behavioral events depend on network behavior such as packet dropping, network scanning, link flooding, etc. Based on the event, an agility specification will be triggered. The agility specification of mutation depends on several factors like mutation parameters such as the IP address of specific hosts, configuration parameters like DNS-entry or switch flow tables, mutation functions like key-based or random distribution etc. The mutation technique get performed by the mutation actions. The event trigger time interval can initiate a periodic triggering of the mutation techniques. The ontology enforces the safety of the MTD techniques by considering the proper declaration of individual constraints that requires for each mutation strategy.

3.5.2 Language Specification

The specifications of MTD contain the following aspects as shown in Figure 3.3.

MTD name N is a *string* that can be used to define an MTD rule. Using *N*, the same rule can be used later on. Besides, the naming will help the user to keep track while creating multiple rules.

Mutation Event E is the trigger that initiates the mutation. The triggering event can be *time* based or network system *behavior* oriented. For example, the system admin can start IP mutation to protect critical resources in a timely fashion. Therefore after a certain period of time, the mutation technique will start and/or repeat. Here the triggering event is *time interval*, Δ. However, the admin may want to mutate a route if any of the links in the route get flooded. In that case, the mutation triggering event is network behavior oriented, which is if a link gets flooded.

$$
\begin{aligned}
\textit{Agility Rule } \Pi &::= N:\ E \rightarrow \Lambda \\
\textit{MTD Name } N &::= \textbf{\textit{STRING}} \\
\textit{Mutation Event } E &::= \Delta \mid \alpha \\
\textit{Time Interval } \Delta &::= \textbf{\textit{NUMBER}} \\
\textit{Sensor Alert } \alpha &::= \textit{isHostScanning()} \mid \textit{isLinkFlooding()} \mid \textit{isBotDetected()} \mid \\
&\quad\ \ \textit{checkUDPICMPPRate()} \mid \textit{getAvailableBandwidth()} \mid \\
&\quad\ \ \textit{checkNewComers()} \mid \textit{checkElephantTCP()} \mid \\
&\quad\ \ \textit{getRouteLength()} \mid \textit{getCriticalLink()} \mid \textit{getRouteRisk()} \mid \\
&\quad\ \ \textit{getFlowStatistics()} \mid \textit{getAllFlowRules()} \mid \textit{getFlowRate()} \\
\textit{Agility Spec. } \Lambda &::= \textbf{\textit{MUTATE}}\ p\ id\ \textbf{\textit{OF}}\ \{attr\}\ \textbf{\textit{USING}}\ f\ \textbf{\textit{ON}}\ g\ \textbf{\textit{BY}}\ m \\
&\quad\ \ \textbf{\textit{WHILE}}\ c \\
\textit{Mutation Param. } p &::= \textbf{\textit{ROUTE}} \mid \textbf{\textit{IP}} \mid \textbf{\textit{STATE}} \\
\textit{Identifier } id &::= [A - Z] \\
\textit{Attribute } attr &::= rattr \mid ipattr \mid sattr \\
\textit{Route Attr. } rattr &::= id.src \rightarrow IPAddress, id.dst \rightarrow IPAddress \\
\textit{IP Attr. } ipattr &::= id.IP \rightarrow \langle list\ of\ IPAddress\rangle \mid h \\
\textit{Host Name } h &::= \textbf{\textit{STRING}} \\
\textit{State Attr. } sattr &::= id.IP \rightarrow \langle list\ of\ IPAddress\rangle \mid s \\
\textit{Activity Status } s &::= \textbf{\textit{UP}} \mid \textbf{\textit{DOWN}} \\
\textit{Mutation Func. } f &::= \textit{random}(\eta, \eta) \mid \textit{key-based} \\
\textit{Numeric } \eta &::= \textbf{\textit{NUMBER}} \\
\textit{Configuration } g &::= r \mid r, g \\
\textit{Resource } r &::= \textbf{\textit{DNS-entry}} \mid \textbf{\textit{switch-table}} \mid s \\
\textit{Mutation Action } m &::= \textit{ipMutate} \mid \textit{pathMutate} \mid \textit{spatioTemporalMutation} \mid \\
&\quad\ \ \textit{createShadow} \mid \textit{reDirect} \mid \textit{migrateService} \\
\textit{Constraint Spec } c &::= \beta \mid \beta\ ;\ c \\
\textit{Agility Const. } \beta &::= \alpha \mid \gamma \mid \delta \mid \delta\ op\ v \\
\textit{Mutation Const. } \gamma &::= id_{t+1}\ op\ \eta \mid id_{t+1}\ op\ id_t \mid attr_{t+1}\ op\ \eta \\
&\quad\ \ attr_{t+1}\ op\ attr_t \\
\textit{Network Func. } \delta &::= \textit{includeSwitch()} \mid \textit{excludeSwitch()} \mid \textit{overlap()} \mid \\
&\quad\ \ \textit{canReach()} \mid \textit{getAllPaths()} \mid \textit{getShortestPath()} \mid \\
&\quad\ \ \textit{getMinDetectionProb()} \mid \textit{getAttackUncertainity()} \\
\textit{Value } v &::= \eta \mid \textbf{\textit{TRUE}} \mid \textbf{\textit{FALSE}} \\
\textit{Operator } op &::= \ > \ \mid \ < \ \mid \ \leq \ \mid \ \geq \ \mid \ = \ \mid \ \neq \ \mid \ \cap \ \mid \ \cup \ \mid \forall \mid \exists \mid \\
&\quad\ \ + \mid - \mid \times \mid / \\
\textit{IPAddress} &::= (([0-9]|[1-9][0-9]|1[0-9]\{2\}|2[0-4][0-9]| \\
&\quad\ \ 25[0-5]) \setminus .)\{3\}([0-9]|[1-9][0-9]|1[0-9]\{2\} \\
&\quad\ \ |2[0-4][0-9]|25[0-5])
\end{aligned}
$$

Figure 3.3: HAPL Syntax for MTDSynth.

***Sensor Alert* $\boldsymbol{\alpha}$** are the sensors in ActivesSDN that can be deployed into the network to collect and measure any behavioral facts, like host scanning, link flooding, bot detecting, critical links finding, etc. The sensor primitives are implemented as functions so that the user can directly use them to trigger MTD events or create constraints for the safe deployment of MTD techniques.

***Agility Specification* $\boldsymbol{\Lambda}$** defines the actions that are taken for the agility technique. For example, the actions of IP mutation is to mutate the IP address of the hosts in the valid address space, and the actions of route mutation are to mutate the route of the specific network flows. Therefore, the agility action specification defines to start an MTD *action* (ipMutate, pathMutate etc.) on a *parameter* (e.g., IP, route) using an mutation *function* which requires automated orchestration of network *configuration* fulfilling specific *constraints* to maintain mission integrity.

Mutation Parameters p the mutable parameter can be IP address, route or state of an service that will be mutated over time. The mutation parameter has *attributes* that provides granularity in defining the exact mutation elements. For example, the IP parameter has attributes like host IP address and host name, the route parameter has source and destination host, the state parameter has status such as service is up or down, etc. Therefore, defining a parameter also requires proper setting of its attributes.

Mutation Function f bounds the mutation space and defines how the mutation will happen. Providing a range in f will limit the mutation space obtained from the solution of constraints in that range. Besides, it will also be used to choose the next mutation parameter from that space. MTDSynth uses two mutation functions from ActiveSDN, random and key-based. In random function, the mutable parameter (e.g. IP address) will be chosen in a random fashion. For key-based function, a proper hash key needs to be provided for selecting the mutation parameter.

Configuration Parameters g include the system parameters that should be correctly configured for the mutation. For instance, the DNS record entry, flow rules in the switch table, and so on. Note that, MTDSynth orchestrates all these low-level configurations automatically.

Mutation Action m are the primary functionality MTDSynth uses for MTD that ActiveSDN provides. Based on an action, that particular class of MTD will be executed, for example, *ipMutate* will mutate the IP addresses, *pathMutate* will mutate the active route of a given flow, *migrateService* will dynamically change the specified services over time, etc. The user does not need to configure these actions, rather just mention the name in the policy. MTDSynth will handle the corresponding configuration of such actions. These actions are complicated to configure by hand, therefore MTDSynth will configure such actions atometically with safety.

Constraints Specification c ensures the safe deployment of the agility rule so that the mission goal remains uninterrupted. For example, while doing mutation (IP, route, or any other parameter), the reachability of the network components must not be interrupted. ActiveSDN provides a comprehensive constraint specification that can be used to define fully qualified constraints to prohibit any conflict or misconfiguration which may occur while deploying MTD rules into the network. This helps the user not to jeopardize the mission integrity while providing maximum security. To define a complete constraints specification, a series of constraints may need to be executed sequentially. This includes sensors, mutation parameter, the attributes of the parameter along with the primitives ActiveSDN API provides. To define *mutation constraints*, the current (t) attribute values will be used to determine the next (t+*1*) mutation parameter attribute values in the constraints. Therefore, a constraint specification is a combination of multiple network constraints primitives, mutation parameters, mutation parameters attributes, numerical or boolean values, that are combined with arithmetic, relational, or logical operators. MTDSynth provides a vast number of network constraints primitives from ActiveSDN API; however,

```
Path Mutation:
isLinkFlooding(1, 0.2) →
    MUTATE route R of {R.src → IP_1, R.dst → IP_2}
    USING random(1..N) ON switch-table BY pathMutate
    WHILE
        $(R_t \cap R_{t+1})/R_t \geq 0.7$;
        includeSwitch($R_{t+1}, [s_2]$) = TRUE;
        excludeSwitch($R_{t+1}, [s_6]$) = TRUE;
        getRouteLength($R_{t+1}$) ≤ 5;
        getAvailableBandwidth($R_{t+1}$) > getFlowRate($IP_1$, $IP_2$)×1.2;
        getRouteRisk($R_{t+1}$) ≤ 0.25
```

Figure 3.4: Route Mutation Policy.

user can program any constraint they want with the specification to fulfill the safety while deploying any MTD techniques.

Network Function $\boldsymbol{\delta}$ are the primitives ActiveSDN provides to generate a complete constraint specification for different MTD actions. Note that, different primitives return different values, for example, *Boolean*, numeric, or list of objects. While comparing the output of such primitives, the type must be matched, otherwise, semantic error may generate in MTD policy parser module.

3.5.3 Policy Examples

This section describes how MTD policy can be created using the HAPL Syntax from figure 3.3.

3.5.3.1 Route Mutation

Let's assume, the user wants to mutate the route between two hosts h_1 and h_2 if any of the links in the current route of these hosts get flooded. The user has the following constants: the IP address for source host h_1 is IP_1 = 10.0.0.1, the IP address for destination host h_2 is IP_2 = 10.0.0.2; deep packet inspection node = s_2. Assume there's a critical link $l(s_6, s_7)$ that is in the current route h_1 and h_2 using where l can be a target that the user wants to avoid. The agility policy for route mutation is shown in figure 3.4.

R_t is the current route and R_{t+1} is the next mutated route chosen by MTDSynth following the *Agility Action*. The *isLinkFlooding(l, th)* method will check the packet drop rate in link l with the threshold th and if the rate is greater than 20%, the mutation will be triggered. ActiveSDN has a agility primitive called *pathMutate* for the route mutation. The *pathMutate* will mutate the *data path* of a specified flow between source h_1 and destination h_2. The next mutated route R_{t+1} will be selected *randomly* from the *available route space* between h_1 and h_2. The *available route space* will be selected by solving the constrains mentioned in the *WHILE* loop. The only *configuration* changes will be done in *switch* flow rule tables. The framework will automatically orchestrate this flow rule updates in corresponding switches. To choose the next mutated route R_{t+1}, the following constraints will be executed sequentially. The constraint overlap will find the common link ratio between the current route R_t and the next chosen route R_{t+1} by intersection, *isIncludeSwitch* check whether the given route R_t contains the given switches (s_2), *excludeSwitch* ensures the given route must not contain the given switches (s_6), *getRouteLength* returns the links count in the next chosen route which helps to define a maximum hop count in a chosen route, *getAvailableBandWidth* measures the given link bandwidth and the *getFlowrate* checks the number of packets any flow (h_1, h_2) sends per second. The primitive *getRouteRisk* measures the risk of a given route that may get attacked in a probabilistic way. For example, assume that a route having n number of links, where the probability of each link get attacked is p_i where $i \in n$. Then the risk that a route will get attacked is:

$$1 - \prod_{i=1}^{n}(1 - p_i)$$

3.5.3.2 Spatio Temporal IP Mutation

Lets assume the user want to mutate the communication IP between hosts h_1, h_2, ... h_m. The MTD policy is shown in figure 3.5, where N is the size of available address

```
Spatial IP Mutation:
isHostScanning(100, 5) →
    MUTATE IP P of {P.IP → [h_1, h_2, ..., h_m]}
    USING random(1..N) ON DNS-entry, switch-table BY
        spatioTemporalMutation
    WHILE
        $\frac{m\times(m-1)}{N} \leq 0.1$;
        $\forall_{i,j\in N} P_t.h_i \neq P_{t+1}.h_i$
```

Figure 3.5: Spatio Temporal IP Mutation Policy.

```
Temporal IP Mutation:
timeInterval = 5s →
    MUTATE IP P of {P.IP → [h_1, h_2, ..., h_n]}
    USING random(1..N) ON DNS-entry, switch-table BY ipMutate
    WHILE
        $\forall_{i,j\in(1,N)} P_{t+1}.h_i \neq P_{t+1}.h_j$;
        $\forall_{i,j\in N} P_t.h_i \neq P_{t+1}.h_i$
```

Figure 3.6: Temporal IP Mutation Policy.

space, m is the number of mutating hosts. $\frac{m\times(m-1)}{N}$ is the probability of two distinct muting hosts will be assigned the same IP to reach the same destination (collision probability). The equation $\forall_{i,j\in N} P_t.h_i \neq P_{t+1}.h_i$ means that the address assigned to a communication pair will be different in two consecutive intervals.

3.5.3.3 Temporal IP Mutation

Lets assume, user wants to mutate IP for hosts h_1, h_2, ... h_n, where h_1 can be a web server running as *www.xyz.com*, h_2 can be a FTP server and so on. The MTD policy is shown in figure 3.6.

Here the first equation (after the $WHILE$ statement) means that at any fixed time the IP of any host in the set $\{h_1, h_2, ...h_n\}$ will be distinct. The second equation means that for any host in $\{h_1, h_2, ...h_n\}$, the IP assigned to a host in an interval will be different from the next interval.

```
Bot pattern detection:
isBotdetected(sig) →
    MUTATE state S of {S.location → [IP_1, IP_2, ..., IP_n]}
    USING random(1..N) ON status
    WHILE
        for all t, $\sum_{i=1}^{N} S_t.IP_i < T_B$;
        minDetectionProb(S.location) > 0.9;
        attackUncertainty(S.location) > 0.8
```

Figure 3.7: Bot Pattern Detection Policy.

3.5.3.4 MTD against Stealthy Bots

Lets assume the user want to mutate the locations of the detecting service S. The MTD policy is shown in figure 3.7. Here service S has a set of detectors, and every detector is located in a specific host. If the status of the detector is UP then it is used for traffic sensing, and is DOWN when it is not used. Mutate the location of detectors is equivalent to change the status of the detectors. T_B is the upper limit of the number of detectors at any given time, and $S_t.IP_i$ is 1 if and only if a detector is located in the ith host at time t. Function *isDetectBot(sig)* decides if the bot pattern changes judged from the signature of bot traffic, *minDetectionProb()* is the function to calculate the lower bound of the probability of detecting bot traffic, given the current detector locations, and *attackUncertainty()* is the function to measure the uncertainty created against the bots with respect to the location of the detectors. The details of the functions can be found in [68].

3.6 Case study and Evaluation

The case study is designed in two different ways to show the following capabilities: 1) MTD techniques and 2) Deception techniques.

3.6.1 Experiment Setup

The ActiveSDN controller in ACD framework is built using OpenDaylight controller in Java and the middleware is developed in python. The framework is launched in an iMac machine having 32GB of RAM and 4 GHz Intel Core i7 processor with macOS Mojave. A physical EdgeCore SDN Switch having Pica8R PicOS with virtual SDN Open vSwitch(v1.3) is used in the network. Mininet [95] is used to create the virtual SDN network and Vagrant [96] for managing dynamic creation of shadow hosts in the network. The ADC middleware run as a different process alongside with the ActiveSDN. The middleware provides web service for the Interface and communicates with the ADC controller through the REST ActiveSDN API(northbound). ActiveSDN controller use OpenFlow protocol to communicate with the SDN network switches.

3.6.2 MTD Case Studies

This section illustrates several MTD case studies, such as IP mutation, Path mutation, etc.

3.6.3 IP Mutation

IP mutation is an MTD action in MTDSynth. IP mutation mutates or changed the source and/or destination IP address to hide the actual/real IP address from the end-users, whereas the mutation itself is transparent to them. Besides, the mutation doesn't hamper any regular communication between hosts. The end host is unaware of the mutation. IP mutation helps to defend against any scanning attack for collecting network information for reconnaissances. Moreover, the scanner gets detected immediately.

The ipMutate function has three configurable parameters: the list of the *real* host IP address called *rIP* list that will get new mutated virtual IP address *vIP*, the virtual IP address space as *vIP* list and the mutation function *f*. For each *rIP* a *vIP* will be chossen from the *vIP* list using the mutation function *f*. For a periodic IP mutation, the event trigger need to be a time interval Δ, so that the *rIP* will be mutated every Δ seconds.

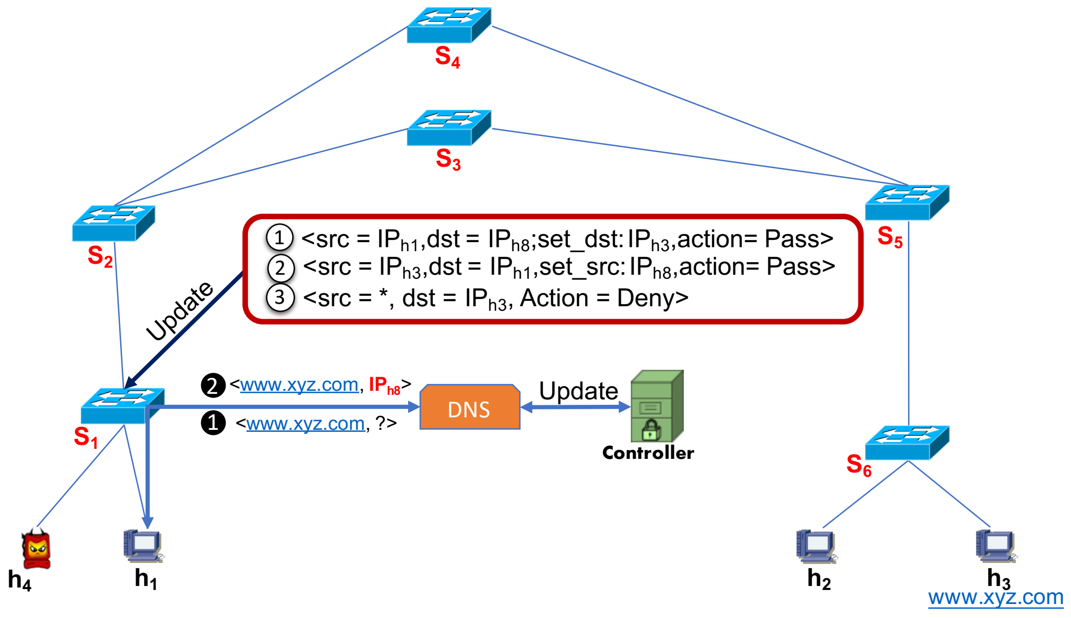

S4
S3
S2
S5
① <src = IPh1,dst = IPh8;set_dst:IPh3,action= Pass>
② <src = IPh3,dst = IPh1,set_src:IPh8,action= Pass>
③ <src = *, dst = IPh3, Action = Deny>
Update
❷ <www.xyz.com, IPh8>
❶ <www.xyz.com, ?>
DNS
Update
Controller
S1
S6
h4
h1
h2
h3
www.xyz.com

Figure 3.8: IP Mutation Example.

Figure 3.8 shows an example of IP Mutation, where h_1, h_2, h_3, h_4 are the the hosts and $s_1, s_2, ..., s_6$ are the switches. A web server *www.xyz.com* is running in h_3 and host h_4 is a scanner. Assume that, IP Mutation is activated with *rIP* as IP_{h_3}, mutation function f as a random function with a range $(1, 10)$ means the the *vIP* list contains a range of ten unused IP addresses (e.g., IP_{h_5} to $IP_{h_{14}}$). If the mutation time interval is Δ, then every Δ seconds, *rIP* IP_{h_3} will get a random *vIP* form the *vIP* list. When IP Mutation starts, in each mutation interval ActiveSDN installs three (the third rule is optional) flow rules into corresponding edge switches, switches that are near to the end hosts and update the DNS entry record. Assume that, for such an interval, the random *vIP* is chosen as IP_{h_8}, the corresponding rules are mentioned in figure 3.8.

Now, when host h_1 try to reach webserver *www.xyz.com*, it makes a DNS query to obtain the IP address of the webserver in step 1. The gateway switch s_1 forwards this request to DNS server. The DNS record is already get updated by the controller and instead of giving the *real* IP address IP_{h_3}, in step 2 the DNS server reply the *vIP* IP_{h_8} as the DNS query request. After achieving the IP address of the webserver, h_1 sends packets setting the destination IP address as IP_{h_8}, and flow rule (1) get matched in s_1. Rule (1) sets the destination IP from IP_{h_8} to IP_{h_3} and forward the packets. When the reply coming back from the webserver to h_1, rule (2) matched that changes back the source IP from IP_{h_3} to IP_{h_8}. Hence, h_1 and h_3 continue their communication withing being known to the mutation.

Scanner h_4, it does not go to the DNS server to obtain IP addresses for any host, rather try to probe by random IP address. When h_4 probe h_3 by IP IP_{h_3} directly, rule (3) matches, and the flow gets dropped. Rule (3) denies any communication that directly forwards to the destination IP address IP_{h_3}. Note that, this rule is optional in IP mutation.

3.6.4 Path Mutation

The path mutation is another MTD action that mutates the *data path* based on a flow between the source and the destination host without interrupting their regular communication. The existing path will be removed, and a new path will be installed depending on the mutation parameter. Path mutation is useful to overcome any link flooding attacks such as the Crossfire Attacks [97] because this technique confused the attacker to fix a specific critical link to flood. Moreover, path mutation provides proactive defense ability into the network restraining attacker permanently from link flooding.

The path mutation API has two main parameters: the flow between a given source-destination IP addresses and a path profile. The path profile is the constraint user made while creating the MTD policy for path mutation. The path profile measures the overlap between current and new mutable path, maximum path length, minimum bandwidth, and the risk a path can have. Besides, any specific *switches* can be included or excluded in the new path. Solving all of these constraints, path profile will generate a mutable path space from where the next mutation path will be chosen. Finally, the event trigger defines how frequently the mutation will happen. If it's a time interval trigger, the path mutation will happen periodically.

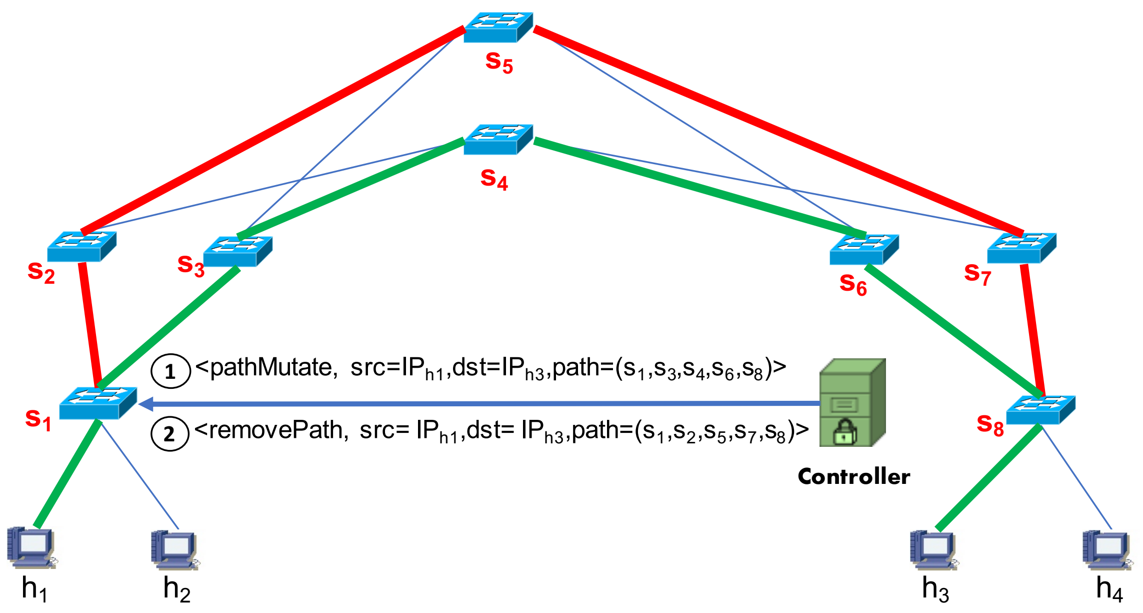

s5
s4
s2
s3
s6
s7
s1
s8
1 <pathMutate, src=IPh1,dst=IPh3,path=(s1,s3,s4,s6,s8)>
2 <removePath, src= IPh1,dst= IPh3,path=(s1,s2,s5,s7,s8)>
Controller
h1
h2
h3
h4

Figure 3.9: Path Mutation Example.

Figure 3.9 is an example of path mutation. Before the mutation starts, source host h_1 communicates with destination host h_3, using path $(s_1, s_2, s_5, s_7, s_8)$. Then after path mutation gets triggered, MTD controler get a mutable path$(s_1, s_3, s_4, s_6, s_8)$ from path profile and configure *pathMutate* API using rule (1) $src = IP_{h_1}$, $dst = IP_{h_3}$ and path $= (s_1, s_3, s_4, s_6, s_8)$. This will install a new path for source h_1 to destination h_3. After mutation, instead of using the old path $(s_1, s_2, s_5, s_7, s_8)$, source h_1 now communicates with destination h_3 using the new mutated path $(s_1, s_3, s_4, s_6, s_8)$. This mutation is transparent to all end hosts while the communication in between them is uninterrupted.

Now the challenge in path mutation is to make uninterrupted communication between the end hosts using the new path instead of the old path. To do so I use *Priority* and *Timeouts* of the flow entries. According to OpenFlow specification [63] there are two fields available in the flow entries, 1) *Flow Priority*: matching precedence of the flow entry, if any packet matches with two different flow entries, the packet will follow the higher priority flow entry and 2) *Idle Timeout*: if it is set, then the flow entry will be expired (removed from the flow table) in the specified number of seconds if any packets are not hitting the entry.

To ensure the uninterrupted property of path mutation, there are two options: 1) explicitly call the *removePath* API that will delete the old path, or 2) install every flow rules in *pathMutation* setting the idle timeout with a specific number of seconds(usually the time interval of the event trigger). In MTDSynth, for each mutation cycle, to delete the previous path and install a new path, a new set of flow rules get deployed to the switches with a given idle timeout but higher flow priority from the previous flow rules. Therefore, if the packets find two sets of matching flow rules but different flow priorities, they follow the higher priority rules. The lower priority flow rules become idle and get removed from the flow table after *timeout* time, which means the old path get deleted.

3.6.5 Deception Case Studies

This section illustrates several case studies related to cyber Deception. The ADC API can be used to achieve various deception goal:

3.6.5.1 Anonymity and Diversity

Anonymity and diversity are effective concealments to hide the true identity of a host [28]. For instance, by randomizing the IP address of critical resources, agile defenses may fail because skilled attackers can identify their target by the static fingerprint of that host (e.g., OS, running services, and their versions). Therefore, deception by *k-anonymization* places $(k-1)$ shadow host with identical fingerprinting of the target host. Moreover, to defeat the static fingerprint problem, *l-diversity* places $(l-1)$ decoy host with fake services of the same software type but different versions/vendors.

The anonymity and diversity can easily be achieved using the *createHoneyNetwork()* API shown in table 3.4. The API dynamically creates a honey network with anonymized shadow hosts and diverse decoy hosts. Listing 2 shows how to invoke

Table 3.4: Deception API: *createHoneyNetwork()*

Param	**Descriptions**
target	The critical resources (hosts, services, links, etc.) to defend.
impact	Impact of the critical resources. (low, medium or high).
k	To anonymize fingerprinting, *k-anonymity* places $(k-1)$ shadow host with identical fingerprinting of the target host.
l	To anonymize configuration, *l-diversity* places $(l-1)$ fake services of same software type but different versions/vendors.
trigger	*activate*: Activate generated honey network.
	deactivate: Deactivate and remove honey network.

Table 3.5: ADC deflection API: *reDirect()* and *reRoute()*

	Param	**Descriptions**
reDirect()	***src***	Source host IP or flow ID.
	dst	Destination host IP or flow ID.
	to	The redirection destination, can be a switch, host, IDS or even the controller.
reRoute()	***src***	Source host IP or flow ID.
	dst	Destination host IP or flow ID.
	to	A new route consist of switches between ***src*** and ***dst*** e.g., s_1, s_2, s_4, s_9.

createHoneyNetwork(). The *target* is the critical resource, *impact* defines the risk of being compromised (high means very critical resources), the k and l are integer value that defines the anonymity and diversity numbers.

3.6.5.2 Deflection by Redirection

The defender can deflect adversaries away from real host to a shadow/decoy host or an inspection environment to let them run in order to discover unknown attack techniques. ADC provides deflection API such as *reDirect()* and *reRoute()* to serve this purpose. Table 3.5 shows the APIs. The *reDirect()* API deflects adversary traffic to an inspection host based on source IP or source flow ID. The *reRoute()* API is useful to deceive any man in the middle listener by changing the active route between two hosts.

Table 3.6: ADC depletion API: *spatioTemporalMutation()*

Param	**Descriptions**
$\boldsymbol{h}$	Target host list for spatial mutation.
$\boldsymbol{eIP}$	List of ephemeral IP addresses. (Optional)
$\boldsymbol{m_i}$	$\boldsymbol{eIP}$ collision rate where $i \in h$
$\boldsymbol{t}$	Lifespan of $\boldsymbol{eIP}$ (temporal period).
$\boldsymbol{how}$	$\boldsymbol{eIP}$ distribution fucntion, can be uniform or random

3.6.5.3 Depletion by Spatio-Temporal Mutation

Depletion means the consumption of adversaries' resources by increasing its computation, confusing towards plausible target identification, and delaying in attack propagation. Therefore, the adversary requires to be more interactive with the system; eventually reveals its identity and hidden tactics. For instance, depleting scanning attacks can be quantified as the total number of probes required to make a hit to the target is higher than a certain threshold. Spatio-temporal mutation distorts the adversary views towards the network [93]. It randomizes the static IP binding to hosts periodically, so that collective reconnaissance information becomes obsolete after that period. Besides, the adversary requires to probe again if they make a further lateral movement. Therefore, Spatio-temporal mutation either makes the adversary hit the wrong target (decoy/shadow in ADF) or increase the total number of probes. The spatial mutation assigns a host multiple ephemeral IP (eIP) addresses so that each of the neighbors uses a different IP address to communicate with that host. Table 3.6 shows the ADC API for *spatioTemporalMutation()*. The parameter h is a list of target hosts of which IP will be mutated by the given list of ephemeral IP, *eIP*. If no eIP is given, ADC selects eIP randomly. Every target host receives a list of eIP; therefore, all neighbor reaches that target host by distinct eIPs. However, a target host may have the same eIP for multiple neighbors, which is defined by the collision rate m_i.

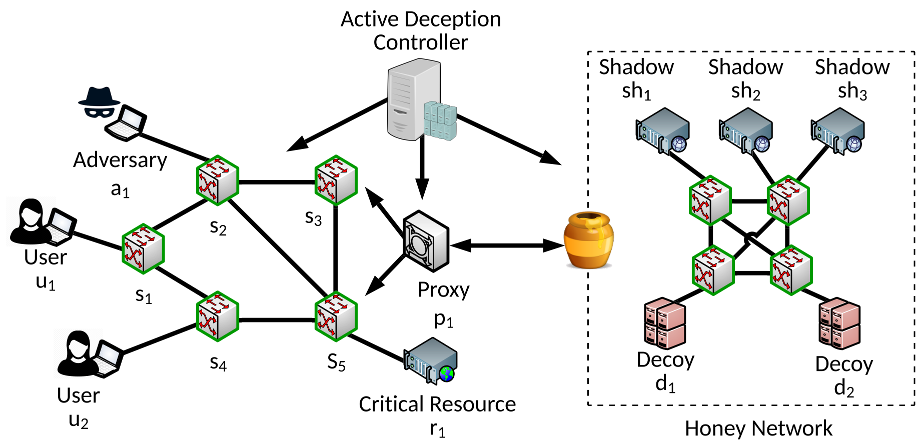


Figure 3.10: Adversary adaptive deception by creating honey network with shadow and decoy hosts.

Table 3.7: ADC SMT solver creates honey configuration for critical host r_1 by 2-anonymity and 3-diversity.

Host	**OS**	**Services**		
r_1	Ubuntu	Vsftpd-2.3.5	Apache-2.2.22	MySQL-5.5.54
d_1	MacOS	CrushFTP 9.3.0	Nginx-1.17.8	Postgresql-10.2
d_2	Win7	FileZilla-0.9.58	IIS 7.5	SQL Server-13
sh_1	Ubuntu	Vsftpd-2.3.5	Apache-2.2.22	MySQL-5.5.54
sh_2	MacOS	CrushFTP 9.3.0	Nginx-1.17.8	Postgresql-10.2
sh_3	Win7	FileZilla-0.9.58	IIS 7.5	SQL Server-13

Listing 5 Automated shadow host (sh_1) configuration script in Vagrant.

```
Vagrant.configure("2") do |config|
   config.vm.define "shadow_1" do |shadow_1|
       shadow_1.vm.box = "hashicorp/precise64"
       shadow_1.vm.network "public_network", bridge: "Ethernet", ip: "10.38.60.2", netmask:"255.255.224.0"
       shadow_1.vm.provision "shell", inline: "sudo apt-get -y install vsftpd=2.3.5"
       shadow_1.vm.provision "shell", inline: "sudo apt-get -y install apache2=2.2.22"
       shadow_1.vm.provision "shell", inline: "sudo apt-get -y install mysql-server=5.5.54"
   ...
```

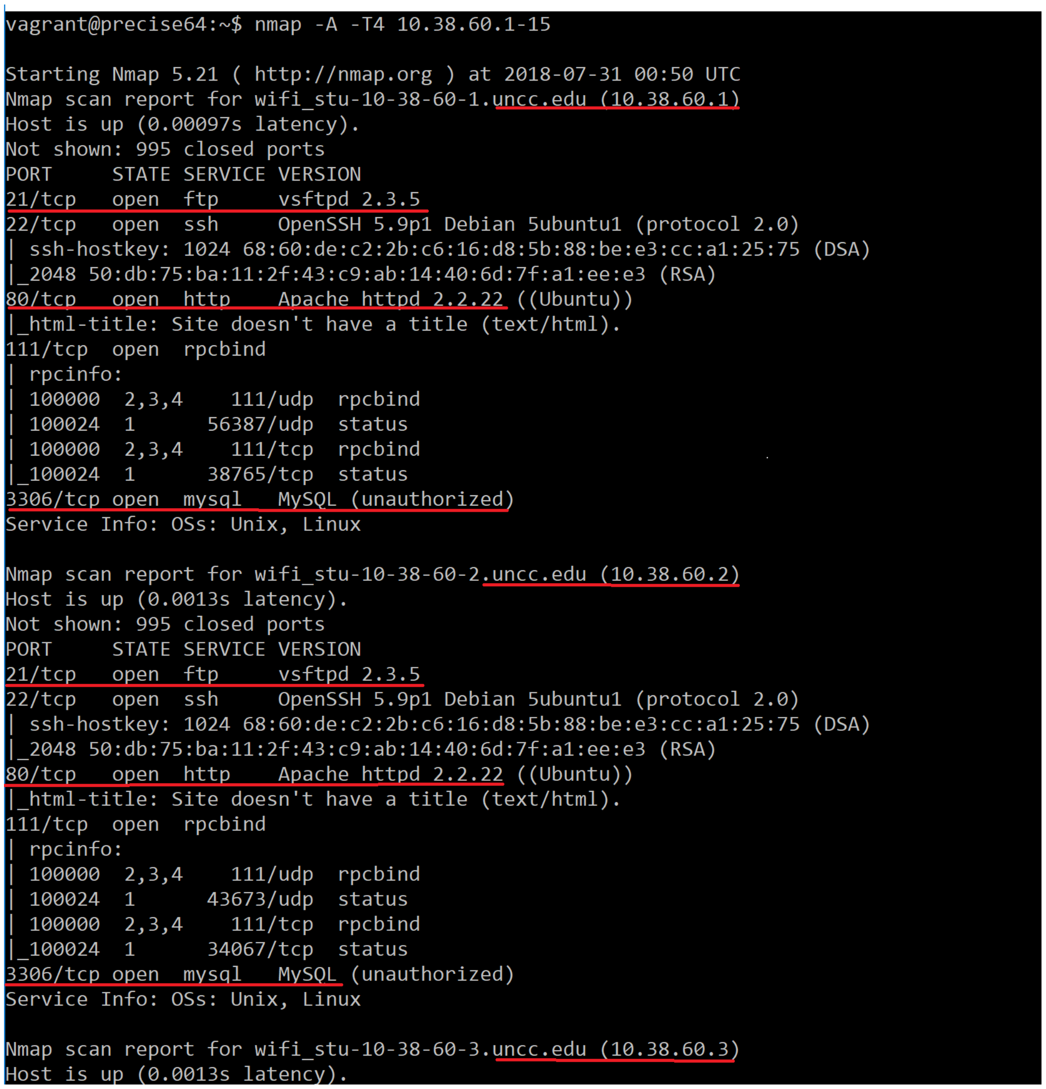
3.6.6 Adversary Distortion by Anonymity and Diversity

The adversary's' views towards the network can be distorted by creating anonymity and diversity for critical resources. Figure 3.10 shows a network, where there is a probability that the adversary starts scanning to identify critical resource r_1. Therefore, the defender launches deception by distortion using ADC API *createHoneyNetwork()*. The API call is shown in listing 2. ADC incorporates ADC solver with the values of k and l to optimized the honey network configuration. Table 3.7 portrays the optimized results. Real host r_1 has OS Ubuntu running on it with three different services. The solver generates two decoy d_1 and d_2 for diversity and three shadows sh_1, sh_2, and sh_3 for anonymity. After that, ADC generates a Vagrant [96] script shown in listing 5 to create the honey network with shadows and decoys. The honey network is connected with proxy p_1. ADC takes a few seconds to create honey networks with different sizes. Figure 3.13 shows that it takes around 3.7 seconds to create a honey network with twenty shadow and decoy hosts.

Table 3.8: Ephemeral IP assignment with real IP.

	Real IP	eIP	
u_1	10.0.0.1	10.0.0.10	10.0.0.11
u_2	10.0.0.2	10.0.0.8	10.0.0.9
r_1	10.0.0.3	10.0.0.6	10.0.0.7

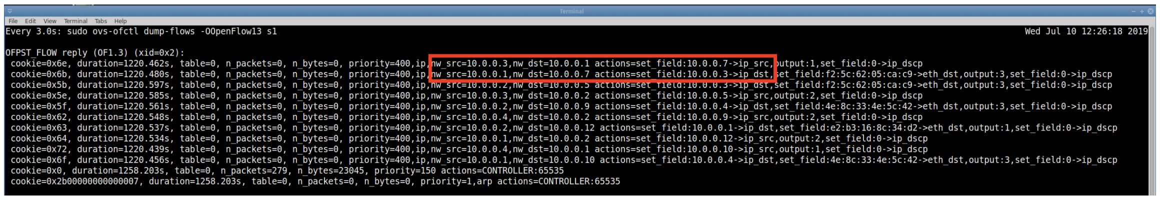
Distortion then Discovery. After the deployment of the *createHoneyNetwork()*, adversary probing will yield a distorted view with many possible targets. A Nmap [98]



```
vagrant@precise64:~$ nmap -A -T4 10.38.60.1-15

Starting Nmap 5.21 ( http://nmap.org ) at 2018-07-31 00:50 UTC
Nmap scan report for wifi_stu-10-38-60-1.uncc.edu (10.38.60.1)
Host is up (0.00097s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.5
22/tcp   open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1 (protocol 2.0)
| ssh-hostkey: 1024 68:60:de:c2:2b:c6:16:d8:5b:88:be:e3:cc:a1:25:75 (DSA)
|_2048 50:db:75:ba:11:2f:43:c9:ab:14:40:6d:7f:a1:ee:e3 (RSA)
80/tcp   open  http    Apache httpd 2.2.22 ((Ubuntu))
|_html-title: Site doesn't have a title (text/html).
111/tcp  open  rpcbind
| rpcinfo:
| 100000  2,3,4    111/udp  rpcbind
| 100024  1      56387/udp  status
| 100000  2,3,4    111/tcp  rpcbind
|_100024  1      38765/tcp  status
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OSs: Unix, Linux

Nmap scan report for wifi_stu-10-38-60-2.uncc.edu (10.38.60.2)
Host is up (0.0013s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.5
22/tcp   open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1 (protocol 2.0)
| ssh-hostkey: 1024 68:60:de:c2:2b:c6:16:d8:5b:88:be:e3:cc:a1:25:75 (DSA)
|_2048 50:db:75:ba:11:2f:43:c9:ab:14:40:6d:7f:a1:ee:e3 (RSA)
80/tcp   open  http    Apache httpd 2.2.22 ((Ubuntu))
|_html-title: Site doesn't have a title (text/html).
111/tcp  open  rpcbind
| rpcinfo:
| 100000  2,3,4    111/udp  rpcbind
| 100024  1      43673/udp  status
| 100000  2,3,4    111/tcp  rpcbind
|_100024  1      34067/tcp  status
3306/tcp open  mysql   MySQL (unauthorized)
Service Info: OSs: Unix, Linux

Nmap scan report for wifi_stu-10-38-60-3.uncc.edu (10.38.60.3)
Host is up (0.0013s latency).
```

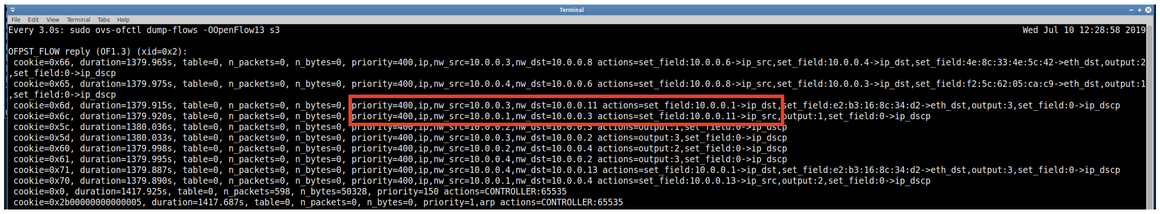
Figure 3.11: Nmap scanning finds new host (sh_1).

scanning result is shown in figure 3.11, where the adversary finds a possible target sh_1, which is actually a shadow host. Therefore, adversaries either land in a decoy/shadow or increase probing to find its real target. Both lead them to engage with the defender, which makes them get detected.



```
File  Edit  View  Terminal  Tabs  Help
Every 3.0s: sudo ovs-ofctl dump-flows -OOpenFlow13 s1                                                                    Wed Jul 10 12:26:18 2019

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x6e, duration=1220.462s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=set_field:10.0.0.7->ip_src,output:1,set_field:0->ip_dscp
 cookie=0x6b, duration=1220.480s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.7 actions=set_field:10.0.0.3->ip_dst,set_field:f2:5c:62:05:ca:c9->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x5b, duration=1220.597s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.2,nw_dst=10.0.0.5 actions=set_field:10.0.0.3->ip_dst,set_field:f2:5c:62:05:ca:c9->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x5e, duration=1220.585s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=set_field:10.0.0.5->ip_src,output:2,set_field:0->ip_dscp
 cookie=0x5f, duration=1220.561s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.2,nw_dst=10.0.0.9 actions=set_field:10.0.0.4->ip_dst,set_field:4e:8c:33:4e:5c:42->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x62, duration=1220.548s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=set_field:10.0.0.9->ip_src,output:2,set_field:0->ip_dscp
 cookie=0x63, duration=1220.537s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.2,nw_dst=10.0.0.12 actions=set_field:10.0.0.1->ip_dst,set_field:e2:b3:16:8c:34:d2->eth_dst,output:1,set_field:0->ip_dscp
 cookie=0x64, duration=1220.534s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=set_field:10.0.0.12->ip_src,output:2,set_field:0->ip_dscp
 cookie=0x72, duration=1220.439s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.4,nw_dst=10.0.0.1 actions=set_field:10.0.0.10->ip_src,output:1,set_field:0->ip_dscp
 cookie=0x6f, duration=1220.456s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.10 actions=set_field:10.0.0.4->ip_dst,set_field:4e:8c:33:4e:5c:42->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x0, duration=1258.203s, table=0, n_packets=279, n_bytes=23045, priority=150 actions=CONTROLLER:65535
 cookie=0x2b00000000000007, duration=1258.203s, table=0, n_packets=0, n_bytes=0, priority=1,arp actions=CONTROLLER:65535
```

(a) r_1(10.0.0.3) reaches u_1(10.0.0.1) through eIP (10.0.0.7) as source IP. Similarly, when u_1(10.0.0.1) replies to eIP (10.0.0.7), the destination changes back to r_1(10.0.0.3) from (10.0.0.7).



```
File  Edit  View  Terminal  Tabs  Help
Every 3.0s: sudo ovs-ofctl dump-flows -OOpenFlow13 s3                                                                    Wed Jul 10 12:28:58 2019

OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x66, duration=1379.965s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.8 actions=set_field:10.0.0.6->ip_src,set_field:10.0.0.4->ip_dst,set_field:4e:8c:33:4e:5c:42->eth_dst,output:2
,set_field:0->ip_dscp
 cookie=0x65, duration=1379.975s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.4,nw_dst=10.0.0.6 actions=set_field:10.0.0.8->ip_src,set_field:10.0.0.3->ip_dst,set_field:f2:5c:62:05:ca:c9->eth_dst,output:1
,set_field:0->ip_dscp
 cookie=0x6d, duration=1379.915s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.11 actions=set_field:10.0.0.1->ip_dst,set_field:e2:b3:16:8c:34:d2->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x6c, duration=1379.920s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=set_field:10.0.0.11->ip_src,output:1,set_field:0->ip_dscp
 cookie=0x5c, duration=1380.036s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:1,set_field:0->ip_dscp
 cookie=0x5d, duration=1380.033s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:3,set_field:0->ip_dscp
 cookie=0x60, duration=1379.998s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:2,set_field:0->ip_dscp
 cookie=0x61, duration=1379.995s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:3,set_field:0->ip_dscp
 cookie=0x71, duration=1379.887s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.4,nw_dst=10.0.0.13 actions=set_field:10.0.0.1->ip_dst,set_field:e2:b3:16:8c:34:d2->eth_dst,output:3,set_field:0->ip_dscp
 cookie=0x70, duration=1379.890s, table=0, n_packets=0, n_bytes=0, priority=400,ip,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=set_field:10.0.0.13->ip_src,output:2,set_field:0->ip_dscp
 cookie=0x0, duration=1417.925s, table=0, n_packets=598, n_bytes=50328, priority=150 actions=CONTROLLER:65535
 cookie=0x2b00000000000005, duration=1417.687s, table=0, n_packets=0, n_bytes=0, priority=1,arp actions=CONTROLLER:65535
```

(b) u_1(10.0.0.1) reaches r_1(10.0.0.3) through eIP (10.0.0.11) as source IP. Similarly, when r_1(10.0.0.3) replies to eIP (10.0.0.11), the destination changes back to u_1(10.0.0.1) from (10.0.0.11).

Figure 3.12: Flow rules for Spatio-temporal mutation.

Table 3.9: Forwarding entry mapping with real IP and *eIP*.

	u_1(10.0.0.1)	u_2(10.0.0.2)	r_1(10.0.0.3)
u_1(10.0.0.1)	-	10.0.0.10	10.0.0.11
u_2(10.0.0.2)	10.0.0.8	-	10.0.0.9
r_1(10.0.0.3)	10.0.0.7	10.0.0.6	-

3.6.7 Adversary Depletion using Spatio-temporal Mutation

The anonymity and diversity distorted the adversary view towards the network. However, the defender can deplete adversaries by calling the ADC API *spatioTemporalMutation()*. Assume that the defender wants spatial mutation for user u_1, u_2, and critical resource r_1. After the mutation API gets called, ADC assigns *eIP* for these hosts shown in table 3.8. Row 2 in table 3.8 means, u_1 has real IP (10.0.0.1) and two ephemeral IP (10.0.0.10) and (10.0.0.11). Table 3.9 shows the forwarding rules of these host through *eIP*. For instance, Row 2 in table 3.9 means, u_1(10.0.0.1) communicates with u_2(10.0.0.2) and r_1(10.0.0.3) through *eIP* (10.0.0.10) and (10.0.0.11) respectively instead of its real IP (10.0.0.1).

ADC installs the corresponding flow rules into the SDN switches shown in figure 3.12. The flow rules in figure 3.12a shows, how r_1 sends packets to u_1 using *eIP* and gets reply. For instance, r_1(10.0.0.3) reaches u_1(10.0.0.1) through eIP (10.0.0.7) as source IP instead of its real IP (10.0.0.3). Similarly, when u_1(10.0.0.1) replies to eIP (10.0.0.7), the destination changes back to (10.0.0.3) from (10.0.0.7) and the replies delivered to the real recipient r_1(10.0.0.3). The rules in figure 3.12b similarly shows how u_1(10.0.0.1) reaches r_1(10.0.0.3). After time interval t, the *eIP*s in table 3.8 changes to a new sets of *eIP*s. ADC incurs limited overhead into the system for deploying spatial mutation. The cost depends on the total number of *eIP*. For a spatial mutation with fifty *eIP*, ADC requires 2.7 seconds to install all necessary flow rules into the network shown in figure 3.14.

Depletion then Discovery. Following the deployment of *spatioTemporalMutation()*, the real IPs for all hosts become obsolete to communicate with each other. Therefore, adversaries need to probe every after t times for reconnaissance. These raise the total number of probing significantly, which consumes the adversary resources, makes them less stealthy, and causes the attack costly. If the adversary directly probes the real IPs to any of the hosts, ADC marks that host as a potential scanner and redirect him to a shadow/decoy for further inspection.

3.6.8 Adversary Deflection by redirection

Distortion and depletion make stealthy attackers more interactive with the system, which increases their exposure. When adversaries frequently engage with the system, for instance, it increases the number of probes to identify targets, and the ADC sensor observes such unusual activities. Therefore, ADC can whitelist benign users and mark potential adversaries. ADC deflects such adversaries by redirection using API *reDirect()*. The *reDirect()* API installs the following rules:

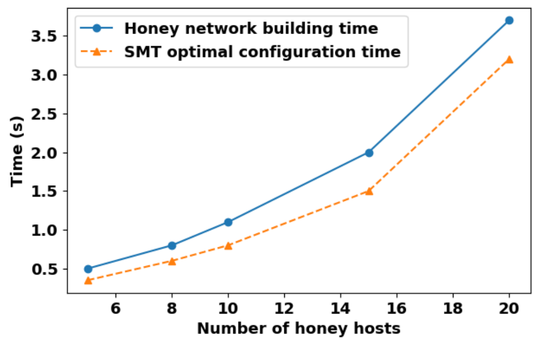


Figure 3.13: Honey network creation overhead.

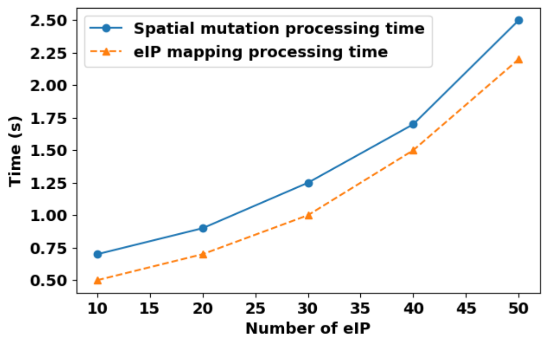


Figure 3.14: Spatial mutation overhead.

1. *(src=*, dst=IP_{r_1}, set_ dst:IP_{p_1})*

2. *(src=$IP_{attacker}$, dst=IP_{r_1}) $\rightarrow$ (src=IP_{p_1}, dst=IP_{d_1})*

3. *(src=IP_{d_1}, dst=IP_{p_1}) $\rightarrow$ (src=IP_{r_1}, dst=$IP_{attacker}$)*

The first rule redirects all traffic to the proxy that is direct probing to critical resource r_1, assuming there is a probability that the source is a potential attacker. Because, after mutation gets started, no benign user probes r_1 directly with its real IP. Therefore, the proxy redirects the traffic to a decoy to inspect the adversary activities. The next two rules are to make the adversary blind follower. For instance, second rules in the p_1 keep track of the source IP address while redirecting it to a decoy (d_1). When the decoy replies, p_1 reverse the source IP back to r_1 using the third rule. Therefore, the adversaries assume that they reach their target r_1, but the response is actually coming from a decoy (d_1).

3.7 Summary

This chapter presented an active cyber defense (ACD) framework to develop and deploy various deception and MTD techniques fast and safely leveraging the open programming capability of SDN. A formal ontology is introduced and a MTD language for agility is provided. Several MTD language examples are showed for temporal and spatial IP mutation, path mutation, etc. The ACD framework enables an open environment for developing sophisticated cyber deception and MTD applications. ACD framework facilitates swift, safe, and effective cyber deception and MTD deployment into the SDN network. ACD leverages an extensive rich API that can be used to build multi-strategy deception and MTD policies. ACD provides sensors that observe adversary activities in real-time, helping to interact with adaptive adversaries to make active deception and MTD techniques. In addition, the management API helps to conduct low-level network configurations without human intervention. ACD framework is flexible; its API can be extended based on requirements. ACD framework is evaluated in the SDN network showing different case studies by developing various goal-oriented deception and MTD strategies. ACD framework incurs very little system overhead while providing proactive defense by deception and agility.

CHAPTER 4: Chimera: Autonomous Planning and Orchestration for Malware Deception

4.1 Motivation

Cyber deception is a paradigm that aims to work beyond traditional detect-then-prevent approaches. In cyber deception, the defender intentionally conceals or falsifies the real configuration of the system's parameters (e.g., network topology, IP addresses, hardware IDs, registry keys, and more) to create uncertainty and confusion for the adversary to mislead their perceptions and decision processes [9]. The state of the art of cyber deception mainly focuses on developing high fidelity decoy systems that work as a standalone sandbox or virtual machine (VM) [54, 99, 100]. These decoys have fake files, user accounts, credentials, and more. If the adversary interacts or exfiltrates such honey resources, the defender gets alerts.

However, the goal of cyber deception is beyond just to catch attackers into VMs by setting up several traps. For example, if the adversary has already penetrated the real system or fingerprinted the VMs to avoid, then static and decoy VM based deception techniques become ineffective [101–103]. Therefore, it is necessary to understand the distinct goals of deception to design an augmented reality that can be embedded with real systems. Firstly, cyber deception can be used to *divert* the adversary away from the real target to a false or no target when the adversary is already in the system—e.g., providing honey files [12] while the attacker searches for sensitive files. Secondly, the defender can *distort* the adversary's perception of the infrastructure by adding ambiguity into the system, e.g., running fake services with obvious vulnerabilities (called honey patches [13]). Thirdly, cyber deception can *deplete* adversary's computational power and resources to delay the attack propagation—for example, honey encryption [14] of the credential files, which the adversary needs to decrypt. Finally, the defender can *discover* new attack tactics, techniques, and procedures (TTPs) by letting them execute different attack actions in contained honey resources.

Existing deception techniques are mainly developed to stop attacker at a particular kill-chain-phase. For instance, network-level deception techniques like redirecting malicious traffic to decoys [104], or generating a mystified response to probe [57] can protect against reconnaissance and lateral movement. However, very few deception frameworks provide a deception composition strategy to defend APT actors in every kill chain phase [28, 105]. Nevertheless, most honeypots and decoy systems are left behind with static deployment and configurations [28, 106], which skilled attackers can easily evade. Moreover, static planning cannot cope with dynamic APT actors who can counter-deceive the defenders' techniques.

To address these limitations, an autonomous framework called *Chimera*[1] is developed that computes an optimal cyber deception plan dynamically in real-time. To do this, Chimera relies on observing the attacker behavior based on MITRE ATT&CK [107] framework that classifies adversary actions into tactics, techniques, and procedures (TTP), which form a chain called TTP kill-chain. An example of a TTP kill-chain is shown in Fig. 4.1. Utilizing the knowledge base of existing APT behaviors from MITRE, Chimera designs a deceptive environment through composing deception techniques, that achieves 4D deception goals (*divert*, *distort*, *deplete* and *discover*). Chimera aims to deceive the APT actor at every kill-chain phase. Although Chimera can be used to compose deception strategies for sandboxes or VMs, the novelty of the framework is to design deception environments that can be embedded with production machines to deceive APT attacks in real-time.

[1](In Greek mythology) Chimera is a fire-breathing female monster with a lion's head, a goat's body, and a serpent's tail. However, I use Chimera to depict an augmented reality into adversaries' minds, "a thing that is hoped or wished for but in fact is illusory or impossible to achieve."

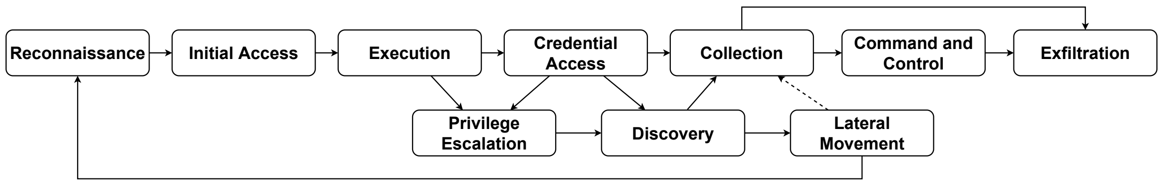


Figure 4.1: APT *kill-chain* model.

Developing such an embedded deception framework is challenging. First, APT actors are adaptive, who can detect and evade/counter deceive existing static deception techniques. Therefore, there is a risk associated with the failure of defense actions in an embedded deception environment, leading to a potential compromise of the system. Second, APT attackers are strategic, who can follow different attack paths to achieve their goal. Therefore, deception planning needs to have the capability to react to the current adopted attack approach. Third, the planning should be correct by construction, which means, if we lie at the reconnaissance phase (e.g., filename f), we must lie believably until the end of the exfiltration phase (e.g., honey content on that filename f). Finally, the deception planning needs to minimize the cost and system overhead while maximizing the achievement of deception goals.

Consequently, a deception action also depends on the attack action. Therefore, Chimera considers the strategic reasoning between the attacker and the defender by integrating the uncertain attack behavior into decision-making. To achieve that, I introduced a new type of attack propagation graph, called a deception graph (Figure 4.3). The deception graph models the dependency between the attacker's action and the defender's deception by embedding two sets of state spaces named attack states (states where attacker lands by executing a successful attack action) and deception states (states where attacker lands if the deception action is successful). Therefore, Chimera models the attack progression over time using the deception graph. Unfortunately, a deception graph for a particular APT such as information stealer can have many states and transitions because of different attack techniques.

Therefore, the defender needs to choose the optimal deception actions regarding effectiveness, costs, and overhead from an enormous set of choices to achieve his goal. I formulate the problem of selecting the optimal deception action using Partially Observable Markov Decision Process (POMDP), that optimizes deception planning by considering the dynamic attack and environment behaviors.

An embedded deception environment is implemented by hooking system-level APIs for Windows OS. First, a manual mapping of 50 ATT&CK techniques with 191 distinct Windows OS APIs is done. Then, hooks are created for all of these APIs to perform the optimal deception actions (e.g., generating a deceptive response) when the adversary calls them. The evaluated of Chimera is done in real-time with three different classes of APTs: Information Stealer, Ransomware, and Remote Access Trojan (RAT). Chimera has high efficiency in deceiving malware in embedded systems and incurs a very low system overhead. The deception plan generated by Chimera is better in achieving distinct deception goals compared with state of the art such as Cuckoo sandbox or Any.run online malware analysis tools.

4.2 Problem Statement

Given the logical topology of the network and systems, assets, deception goal, and ATT&CK techniques, the goal of Chimera is to generate an optimal sequence of deception actions planning that will be orchestrated automatically and deployed with minimal management overhead into the system in real-time.

4.3 Related Work

The state of the art of cyber deception focuses on designing virtual machines (VM) and sandboxes as a decoy [54, 99, 100]. These decoys have fake resources regarding files [108], software [13], user accounts, passwords [109], credentials [109], web pages [12], services, processes, servers, email accounts, and more. If the activity is seen to interacts with these honey resources, an alert is sent to the defender. However, skilled adversaries found unique ways to uncover such decoy VMs [101–103]. Usually, adversaries realize that isolated decoy machines do not initiate traffic, often responding late because they create complicated deception responses, making them easy to identify [9].

Research has been done to create network-level deception by malicious packet redirection [103] and a falsified probe response [57] to defense reconnaissance and lateral movement. System and data level deception such as honey password creation, decoy file generation, vulnerable software patching, etc., protects the system from internal attacks [14,99,104]. Many research has been done to measure the efficacy of cyber deception [105]. Static planning and deception framework composing various deception actions is more efficient to deceive APT actors [28, 106, 110]. To make the attack-defense battle dynamic, many game-theoretic and probabilistic deception model has been introduced [87, 111].

4.4 Threat Model and Scope of the Work

Chimera aims to design a dynamic deception environment that triggers appropriate deception actions in real-time to deceive the attacker's activity in every phase of the kill chain. For instance, Chimera can be used to set up a production machine against information-stealing APT so that defenders can analyze malware (divert, deplete or discover TTPs) that exfiltrates sensitive information. Chimera has two significant aspects: 1) it can deceive malware already running into the production system because of the hooking deception techniques that work at the system API level. 2) APTs with decoy/VM detection capabilities are ineffective as Chimera provides embedded deception integrated with the real system.

It is important to note that the objective of Chimera is to deceive malware in a particular way such that defenders can achieve certain deception goals. Therefore, Chimera does not safelist processes between benign or malicious but deceives a given process based on the goal. For that, Chimera provides optimal planning, a sequence

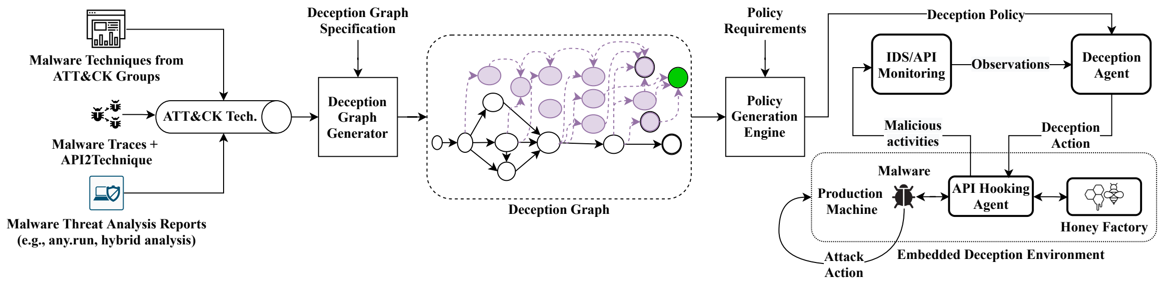


Figure 4.2: Chimera Architecture.

of deception actions based on adversary activity, and honey resources which can be pre-created honey files, credentials, and more. A tremendous amount of research has been done on creating high fidelity honey resources (such as decoy files [12, 108], network traffic [57, 104], credentials [109], and systems [28, 106]). Therefore, in this work, I do not describe the creation of honey resources, but focused on when and how to use them efficiently.

4.5 Chimera System Design

The architecture of Chimera is illustrated in Fig. 4.2. Chimera takes input as APT techniques from MITRE ATT&CK, threat reports, and malware API traces from sandbox (Cuckoo), and other analyzing tools like Hybrid Analysis, Any.run, and more. Chimera maps API call traces to ATT&CK techniques which are later used to generate a deception graph. The graph generator requires specification, such as deception action state space that can defeat certain attack actions. The deception graph is then used to generate optimal deception action planning, where policy definition is given as input, e.g., APT type (Information Stealer or Ransomware). Chimera has a deception agent in the production machine and monitors (IDS/API monitoring) to observe APT actor's activity. The sensor alert triggers the agent to choose an optimal deception action to deceive the attacker's next action. The deception actions in Chimera are implemented in system-level API hooking that fetches honey resources from a Honey Factory to deceive the attacker. The following sections

Table 4.1: ATT&CK technique to Windows API mapping. Columns *Adversary Action* and *Deception Action* are names given by us to represent corresponding technique. These action names are used to describe state transitions in the Deception Graph (Fig. 4.3).

MITRE ATT&CK Technique	**Windows API Call Sequence**	**Adversary Action**	**Deception Action**
Command and Scripting Interpreter	*CreatePipe, CreateProcess, CreateFile, ReadFile, CloseHandle*	execute	migrateInHE
Modify Registry	*RegCreateKeyA, RegSetKeyValueA, RegCloseKey*	addToRegistryRunKeys	doNothing
Query Registry	*RegOpenKey, RegQueryValue, RegCloseKey*	queryRegistry	honeyRegistry
Process Discovery	*1) CreateToolhelp32Snapshot, Process32First, Process32Next* *2) EnumProcesses*	tasklist	honeySwList
File and Directory Discovery	*GetCurrentDirectory, CreateFile, ReadFile, CloseHandle, FindFirstFile, FindNextFile, FindClose*	listDir	redirectToHoneyDir
Clipboard Data	*OpenClipboard, GetClipboardData*	copy	honeyCopy
Input Capture: Keylogging	*1) GetAsyncKeyState, GetKeyState, GetKeyboardState* *2) SetWindowsHookEx, GetKeyState, GetKeyNameText*	copy	honeyCopy

describes each component in detail.

4.5.1 API Sequence to MITRE ATT&CK Technique Mapping

MITRE ATT&CK [107] illustrates the APT lifecycle regarding tactic, technique, and procedure (TTP). A tactic defines attack objective/goal, whereas, techniques are actions to accomplish that goal. An attack technique describes the high-level context explaining why it is executed, what adversary gains from it, and how it is being performed. Such context helps the defender to design necessary deception actions. However, the APT actor interacts with the system using low-level procedures, a sequence of system API calls, to perform an attack technique. For instance, attack techniques such as sensitive files and directories search can be done by following shell commands: `dir`, `tree`, `ls`, `find`, `locate`, etc.

The defender can collect system log events from an ongoing APT campaign. However, to understand the attack context, it is necessary to map the API call traces to high-level attack techniques. Unfortunately, very few works have been done in mapping API calls to ATT&CK techniques and they are very limited [3, 112]. Therefore, a mapping is done for the most essential 50 ATT&CK techniques frequently used in Information Stealer, Ransomware, and RAT with 191 windows API. From 4,578 malware samples, more than 37000 API traces are collected from Cuckoo sandbox [113]

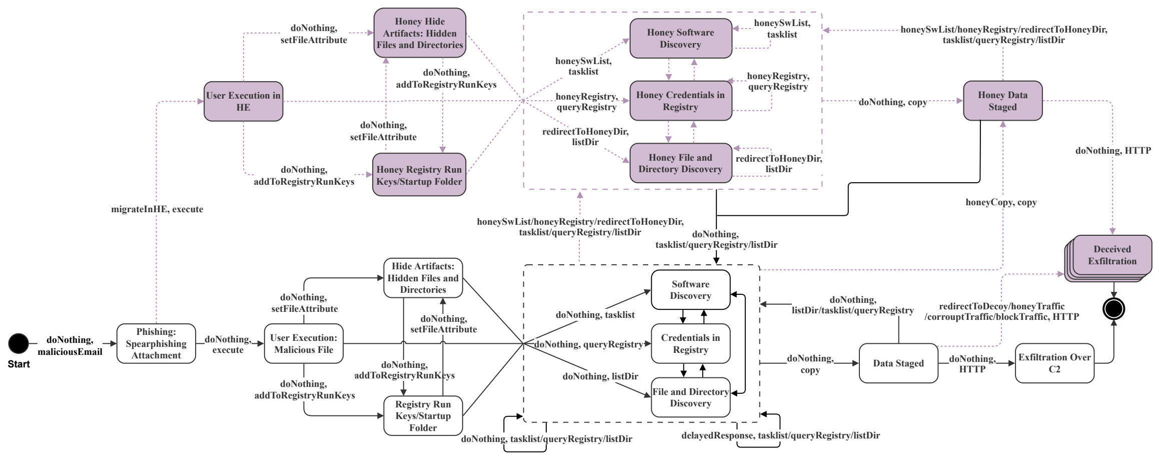

Honey Hide Artifacts: Hidden Files and Directories
doNothing, setFileAttribute
User Execution in HE
doNothing, addToRegistryRunKeys
doNothing, setFileAttribute
doNothing, addToRegistryRunKeys
Honey Registry Run Keys/Startup Folder
honeySwList, tasklist
honeyRegistry, queryRegistry
redirectToHoneyDir, listDir
Honey Software Discovery
Honey Credentials in Registry
Honey File and Directory Discovery
honeySwList, tasklist
honeyRegistry, queryRegistry
redirectToHoneyDir, listDir
honeySwList/honeyRegistry/redirectToHoneyDir, tasklist/queryRegistry/listDir
doNothing, copy
Honey Data Staged
doNothing, HTTP
honeyCopy, copy
Deceived Exfiltration
migrateInHE, execute
honeySwList/honeyRegistry/redirectToHoneyDir, tasklist/queryRegistry/listDir
doNothing, tasklist/queryRegistry/listDir
Start
doNothing, maliciousEmail
Phishing: Spearphishing Attachment
doNothing, execute
User Execution: Malicious File
doNothing, setFileAttribute
Hide Artifacts: Hidden Files and Directories
doNothing, setFileAttribute
doNothing, addToRegistryRunKeys
doNothing, addToRegistryRunKeys
Registry Run Keys/Startup Folder
doNothing, tasklist
doNothing, queryRegistry
doNothing, listDir
Software Discovery
Credentials in Registry
File and Directory Discovery
doNothing, tasklist/queryRegistry/listDir
delayedResponse, tasklist/queryRegistry/listDir
doNothing, listDir/tasklist/queryRegistry
doNothing, copy
Data Staged
redirectToDecoy/honeyTraffic /corrouptTraffic/blockTraffic, HTTP
doNothing, HTTP
Exfiltration Over C2

Figure 4.3: Deception Graph mapping with MITRE ATT&CK techniques.

and their high-level behavior from tools like Any.run [114], Hybrid-analysis [115] and Malware Behavior Catalog (MBC) [112]. Finally, by going through the MITRE ATT&CK techniques description a manually map is made with the API call sequences to ATT&CK techniques. Table 4.1 shows a subset of the mapping. The complete mapping is published in GitHub to stimulate future research in this area [116, 117].

4.5.2 Deception Graph

The deception graph in Chimera is a dependency graph to model the adversary propagation in the system over time. It is similar to attack graphs, where each node represents a distinct adversary position, and each edge represents the transition of attack propagation. A fragment of a deception graph generated for Information Stealer is shown in Fig. 4.3. Unlike the attack graph, the state transition in the deception graph depends on the interaction between attack and deception action. The attacker moves to a state by successfully executing a specific sequence of actions or getting deceived due to deception actions. For instance, in Fig. 4.3, the attacker has to successfully act *execute*, *setFileAttribute*, and *queryRegistry* actions to move from initial position *Phishing* to position *Credentials in Registry.* This research considers a distinct adversary position as a distinct *state* that comprises state space of POMDP. The deception graph has two different types of states: (1) honey state (grey in color) where the attacker reaches after being deceived, and (2) real state (white in color) where the attacker reaches after successful execution of an attack technique. For example, in Fig. 4.3, the defender may redirect the attacker's traffic to a fake (decoy) C2 server, which takes the attacker to a honey state named *Deceived Exfiltration*. Whereas, by successfully exfiltrating data towards real C2 server, the attacker moves to real state *Exfiltration Over C2*.

The state transitions are represented as $(a_{d_1}/a_{d_2}/.../a_{d_n},\ a_{v_1}/a_{v_2}/.../a_{v_m})$, where a_{d_i} is a deception action and a_{v_j} is an attack action separated by ",". Each pair (a_{d_i}, a_{v_j}) represents a distinct transition. Multiple states inside a dotted box represent a superstate. A transition to a superstate delegated for its appropriate sub-state only. For instance, the *(doNothing, tasklist)* transition means if the defender has no deception and the adversary does a *tasklist*, it will jump to the real state *Software Discovery*. Similarly, a *(doNothing, queryRegistry)* will lead him to *Credentials in Registry* state.

4.5.3 Honey Factory

Chimera presents all the deceived responses to the attacker in an Honey Factory (HF). The HF consist of honey files, credentials, passwords, decoy user accounts, email accounts, web pages, software with honey patches, decoy process lists, registry files, honey traffic, decoy servers, VMs, and more. In Chimera, the honey resources in HF can be created offline in remote machines. In the evaluation, three remote VMs are used as part of HF. Therefore, the deception agent can delegate specific attack API calls (e.g., download a secondary piece of malware code command) from the production machine to one of the VMs through API hooking. However, HF resources can be created online inside the production machine. For instance, a decoy process list (honeySwList) can be shown in response to the adversary's tasklist command through hooking the APIs related to the process discovery technique (see Table 4.1 for deceiving process discovery API lists).

4.6 Deception Planning

This section describes how Chimera computes its deception planning to execute optimal deception action.

4.6.1 Deception Decision-making

To achieve the 4D objectives cost-effectively, Chimera computes a policy that recommends an optimal deception action considering the current attack position and behavior.

Formulating Deception Decision-making: The optimal deception strategy at the current time-sequence t depends on the current adversary position. However, it is hard to infer the exact sequence of adversary positions from the beginning (at $t = 0$) due to dynamic environment and attack behavior. Therefore, I formulate the decision-optimization problem as Sequential Decision Process (SDP) [118]. The environment is stochastic due to non-deterministic deception consequences induced because of uncertain attack behavior.

Due to uncertain attack behavior, Chimera cannot certainly know the next attack action/move from a particular adversary position. For example, from the *Data Staged* position (in Fig. 4.3), the attacker may try to discover more credentials or exfiltrate data. Without knowing the next attack action certainly, there is a possibility that the deployed deception may be irrelevant to defend the current attack action. For example, let assume that Chimera decides to execute *honeyCopy* (i.e., replacing discovered data with garbage data) to take the attacker to *Honey Data Staged* position in Fig. 4.3. However, if the attacker exfiltrates data instead of *Copy*, *honeyCopy* is irrelevant. Against unknown attack processes, Chimera can only probabilistically know the next attack move. Besides, sophisticated attackers may adapt action plans based on their observations about previous attack consequences, making any static action planning ineffective. Chimera formulates the SDP environment considering such deception failure probability.

In the SDP, Chimera executes a deception action and analyzes its consequences based on recent observations. Each observation specifies a distinct set of APIs called by the subjected process, based on which, Chimera also infers the current adversary position. However, the monitored set of APIs cannot be certainly mapped to a specific adversary position due to constrained monitoring of limited APIs. Hence, the environment is only partially observable with incomplete and imperfect information.

To address the partial observability, Chimera formulates the decision-making problem as Partially Observable Markov Decision Process (POMDP) [118]. POMDP is a sequential decision process of an agent who acts and receives feedback from the environment synchronously, through addressing uncertainties related to partial observability. It is a tuple of $< S, A, T, \Omega, O, R, \gamma >$ where:

- S, A, and Ω are the state space, deception action space, and observation space, respectively,
- T and O represent the state transition function and observation function, respectively,
- γ is the discount factor.

By solving the POMDP model, Chimera computes an optimal policy that recommends the optimal deception action for the current belief (i.e., probabilistic adversary position).

Among these POMDP parameters, state spaces S are MITRE ATT&CK techniques represented as real state and honey state in the deception graph (section 4.5.2), and deception action space, A, consists of unique deception actions considered for this research. Discount factor γ regulates how far in future Chimera looks to understand the current action consequences into future, which is static in this work.

The following subsections describe the rest of POMDP parameters (Section 4.6.2 to 4.6.4), belief update (Section 4.6.5), and policy generation (Section 4.6.6).

4.6.2 State Transition Matrix

State transition matrix is a POMDP parameter that contains transitional probabilities among states for all considered deception actions. To clarify, it consists of $p(s'|s, a_d)$ that specifies the probability of transition from current state (i.e., adversary position) s to next state s' for the deception/defense action a_d, for all possible $s \in S$, $s' \in S$, and $a_d \in A$. While optimizing policy, POMDP solution approaches consider this parameter mainly to understand action consequence on the environment [118]. Chimera determines $p(s'|s, a_d)$ using the following equation:

$$p(s'|s, a_d) = \sum_{a_v \in V} p(s'|s, a_d, a_v) \times p(a_v|s) \tag{4.1}$$

where, V is the attack space, and $p(s'|s, a_d, a_v)$ is the system behavior that defines the probability of transition from s to s' when the attacker executes a_v in response to a_d. Notably, V consists of unique attack actions. Fig. 4.3 shows some examples of attack actions by the second parameter across edges.

In Eqn. 4.1, Chimera integrates the expected attack behavior into its decision-model, in order to formulate the environment from defender's perspective. This reduces the problem from Partially Observable Stochastic Game (POSG) to POMDP. Thus, Chimera addresses the limitation of POSG in approximating a solution for two players with different payoffs, which improves the scalability significantly.

To determine the system behavior, the agent uses the following equation:

$$p(s'|s, a_d, a_v) = \begin{cases} 1, & if \quad (s', s, a_d, a_v) \quad is \quad valid \\ 0, & Otherwise \end{cases} \tag{4.2}$$

In Eqn. 4.2, the first factor, (s', s, a_d, a_v) is valid if deception a_d is relevant to attack a_v, and (s', s) is relevant to (a_d, a_v). A defense action a_d is relevant to attack action a_v if a_d can defeat/deceive a_v. For example, in Fig. 4.3, *honeySwList* deceives the adversary action *tasklist*; hence, *honeySwList* is relevant to *tasklist.* The combination (s' =*Honey Software Discovery*, s =*User Execution: Malicious File*) is relevant to (a_d =*honeySwList*, a_v =*tasklist*), because the adversary moves to *Honey Software Discovery* from *User Execution: Malicious File* for executing *tasklist* against *honeySwList.* Importantly, the attacker always remains at the same state when he does nothing regardless of other factors.

In Eqn. 4.1, the second factor, $p(a_v|s)$ specifies the probability of executing action a_v at the current state s. This values come from the deception graph.

4.6.3 Observation & Observation Matrix

An observation is a distinct set of API calls that Chimera monitors to infer the current adversary position (state) in the attack chain. However, it cannot certainly specify the underlying state due to partial observability. Here, the observation space Ω is same as state space S. Each observation $o \in \Omega$ is highly correlated with one state $s \in S$ while having low correlations with other states. For example, based on recent observed set of API calls, there is high likelihood that the attacker performed *Software Discovery.* However, due to not monitoring all API calls, Chimera is not certain that the attacker has not performed other actions. The probabilities $p(s|o)$ defining the likelihood of s for the recent observation o, which is computed based on historical data of malware reports.

Composing Observation Matrix: Observation matrix O is a POMDP parameter that specifies correlations among states and observations. Chimera composes O to understand the current state from recent observation $o \in \Omega$. It contains probabilities $p(o|s)$ that specifies the probability of observing o when the state is s, for all possible $o \in \Omega$ and state $s \in S$.

Chimera determines $p(o|s)$ based on recent observation $o \in \Omega$ and prior probabilities $p(s|o)$, using the following equation:

$$p(o|s) = \frac{p(s|o) \times p(o)}{\sum_{x \in \Omega} p(s|x) \times p(x)} = \frac{p(s|o)}{\sum_{x \in \Omega} p(s|x)} \tag{4.3}$$

where, the probability of observing a symptom $p(o)$ is same for all observations.

4.6.4 Reward

Alongside state transition matrix T, POMDP considers reward to optimize the policy through understanding the consequences of defense/deception actions for all possible scenarios. For all possible current state $s \in S$, next state $s' \in S$, and deception action $a_d \in A$, Chimera quantifies $R(s', s, a_d)$ that defines the payoff of action a_d when the state transits from s to s'. Higher reward due to a_d motivates policy to execute a_d. Chimera formulates $R(s', s, a_d)$ using the following equation:

$$R(s', s, a_d) = -q(s') + \sum_{i \in \mathcal{G}} w_i \times Z_i^d - C_d \tag{4.4}$$

where, $q(s)$ is the risk of data exfiltration imposed due to attacker's reaching at state s', and $\mathcal{G}$ is the 4D deception goal that consists of *Diversion*, *Distortion*, *Depletion*, and *Discovery*. Additionally, Z_i^d defines whether a_d achieve the goal $i \in \mathcal{G}$ or not, and C_d is the installment cost of a_d.

In Eqn. 4.4, $q(s)$ depends on two factors: (1) $\rho(s)$ that defines the probability of reaching to *Exfiltration Over C2* from s, and (2) L that is the loss (in dollars) due to data exfiltration. Notabaly, *Exfiltration Over C2* is actually the attack goal state s_g. The first factor, $\rho(s)$, depends on available paths to reach s_g from s, which are obtained from the deception graph at Fig. 4.3 (without considering the defense action). For example, from *Software Discovery*, the attacker can reach to s_g by the path with actions: (*copy*, *HTTP*) or by the path with actions: (*tasklist*, *copy*, *HTTP*).

The second factor, L, is same for all possible scenarios and considered as user-input.

There are multiple available paths to reach s_g from s, and the attacker reaches s_g if he successfully executes all actions of any of available paths. This intuition is formulated using the following equation:

$$\rho(s) = 1 - \prod_{l_j \in \mathcal{L}} (1 - \rho_j(g)) \tag{4.5}$$

where, $\mathcal{L}$ are available paths to reach to s_g from s, and $\rho_j(s)$ is the likelihood of reaching s_g through the path l_j.

$\rho_j(s)$ depends on probabilities of executing attack actions following the sequence in l_j. Understandably, $\rho_j(s)$ gets lower with more required actions to reach s_g, that makes the reward higher. To clarify, from a honey state, he has to repeat previous actions or execute more actions; thus, it reduces the risk of exfiltration and provides incentive to Chimera.

The second term in Eqn. 4.4 provides incentives to a_d for achieving specific deception goals using w_i. Notably, each deception action offers diversified benefits regarding 4D goals. For instance, *redirectToHoneyDir* provides diversion and discovery but not distortion and depletion, whereas, *Honey Data Staged* helps to discover attack behavior. Therefore, by w_i, the user can emphasize on actions that are more inclined to his objectives or mission. The last term, C_d, defines deployment cost of a_d due to required configurations, operations, and others. In evaluation, w_i and C_d are considered as user-inputs.

4.6.5 Belief

Belief b_t is the probabilistic distribution across all states $s \in S$, that probabilistically specifies the current state during time-sequence t. For instance, b_t(*Software Discovery*) defines the likelihood of *Software Discovery* as the current state. Chimera determines next optimal action based on b_t that addresses the imperfect and incomplete observability of the environment.

To address uncertainties related to observations, Chimera determines b_t considering the recent observation o, probable state transitions (state transition matrix), and correlations among states and recent observation (observation matrix). Using the traditional belief update approach [118], Chimera formulates b_t using the following equation:

$$b_t(s) = \frac{p(o|s)\sum_{s''\in S} p(s|s'', a)b_{t-1}(s'')}{\sum_{w\in S} p(o|w)\sum_{s''\in S} p(w|s'', a)b_{t-1}(s'')} \tag{4.6}$$

where, $p(o|s)$ is the probability of observing o at s'.

4.6.6 POMDP Policy Generation

Chimera computes the optimal deception policy by solving the composed POMDP model. It recommends the optimal deception action a_d^* for current belief b_t. To address uncertainties associated with the environment, the composed POMDP model considers not only the probable attack behavior (by state transition matrix), but also correlations of observations (API traces) with probable attack positions (observation matrix). Chimera applies Heuristic Search Value Iteration (HSVI) to solve the POMDP model, which approximates the policy with a bounded (user-given) regret rate [119]. Regret rate defines the precision of HSVI, and the approximated policy moves closer to the optimal policy for lowering its value.

HSVI takes an initial belief as the initial probabilistic attack position, in order to prune irrelevant belief space unreachable from initial attack position. The optimal deception action recommended by the computed policy maximizes the accumulated reward for the current belief b_t, considering not only the current attack position but also the probable attack propagation in future. The following equation formulates the intuition:

$$V^{\pi}(b_t) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s', s, a_t) | b_t, \pi\right]$$

$$\pi^* = \arg\max_{\pi} V^{\pi}(b_t)$$

where, π^* is the computed policy, $V^{\pi}(b_t)$ is the expected reward value considering future decision-horizon (defined by γ) by following policy π, and $R(s', s, a_t)$ is the expected reward for transition from s to s' by defense a_t at time t. Notably, γ^t quantifies the importance of future payoff.

4.7 Implementation

The POMDP model is solved by ZMDP (POMDP solver) [119]. I used python to create the deception graph from MITRE ATT&CK APT reports. I use API-to-MITRE (section 4.5.1) mapping to include techniques in deception graphs from malware traces. For implementing the embedded Honey Factory (HF), system-level API hooking is used. To scale the deception actions with HF resources, I classified all deception strategies into four categories; 1) *Fake Failure*: this strategy always denies any API calls indicating a failure status, e.g., malware wants to compress a file but will get denied stating compression mechanism not found. 2) *Fake Success*: always return a successful response without performing the task. For instance, Ransomware wants to encrypt a file; it will return encryption successfully without any encryption. 3) *Honey Execute*: this strategy delegates the API calls to remote VM that executes the command, and the VM's response goes back to the attacker. 4) *Native Allow*: always allow any API calls.

Embedded Deception-API Hooking: The idea of embedded deception [120] strategy is implemented using EasyHook, a free, open-source hooking library for 32-bit and 64-bit Windows processes released under the MIT license. EasyHook provides a generic template for APIs hooking. In addition, EasyHook ensures thread safety

Table 4.2: Datasets. 4,578 malware samples in total: 3,396 Information Stealers, 1,030 Ransomware, and 152 RATs.

Family	**InfoStealer**													**Ransomware**				**RAT**		
Sub Family	LokiBot	Pony	Khalesi	Kegotip	Rammit	Grozlex	Occamy	FareIt	Floxif	Emotet	Raccon	Generic.PWS	Trojan.PWS	WannaCry	Ryuk	Cerber	GandCrab	Gh0st	Pupy	Quasar
Samples	673	294	207	119	385	77	65	42	106	486	54	552	336	256	161	139	159	53	64	35

by using a thread deadlock barrier. When the deception agent in Chimera decides to deceive the attacker's next movement (technique), the agent chooses a deception strategy. Let us assume the attack technique is File and Directory Discovery, and the defender's strategy is Honey Execute, which delegates every relevant API call to remote HF. From the API-to-MITRE mapping, the agent identifies the malware will invoke the following sequence of API: *GetCurrentDirectory* - *send* - *recv*. After identifying the relevant APIs, the agent uses Easyhook to inject a Dynamic Link Library (DLL) into the malware process including the strategy Honey Execute. According to our example, when the malware calls *GetCurrentDirectory* API, the current working directory was supposed to be copied into the parameter *lpBuffer*. However, the hook forwards the call to HF, and HF reports back a deceptive directory list, which gets copied back into the *lpBuffer*. Eventually, the malware receives the deceptive directory listing instead of the real one.

4.8 Evaluation

Chimera is evaluated with 4,578 real malware samples from three different families: Information Stealer, Ransomeware, and Remote Acces Trojan (RAT). Our key evaluation criteria were how efficiently Chimera deceives malware to obtain distinct deception goals in real-time. Further, I evaluate Chimera's effectiveness of the quality of deception (in terms of discovery, depletion, and diversion), optimal policy generation, and overhead due to deception action deployments. Finally, Chimera is run in a real production machine and discover new TTPs compared to existing APT analysis tools, such as Cuckoo [113] and Any.run [114].

4.8.1 Dataset

Table 4.2 summarizes the datasets used in the evaluation. A collection of 4,578 malware samples from VirusTotal and MalShare are used. I choose 3,396 samples of Information Stealer from 13 families, 1,030 samples of Ransomware from 4 families, and 152 samples of RAT from 3 different families. More than 37,000 API traces are collected from Cuckoo sandbox, DogeTron, and Hybrid-Analysis tool from these malware samples. Furthermore, the lifecycle of 27 Information stealing APTs, 17 Ransowmare APTs, and 19 RAT APTs from MITRE ATT&CK are used.

4.8.2 Experiment setup

80% of the malware sample data is used randomly to build the model for deception action planning, and the rest is used for testing. The training malware samples are run using analyzers such as Cuckoo, DodgeTron, and Any.run to collect traces. From these traces and API-to-MITRE mapping (Table 4.1), I build three different deception graphs for Information Stealer, Ransomware, and RAT. Existing works such as [3, 121] give us insight into calculating the likelihood of adversary transition from a given state to the next state. Further, I quantify the deception action cost and effectiveness as low, medium, and high. Chimera then set up into a production machine with some vulnerable software. A remote Honey Factory with three VMs is created. Honey resources such as honey files, credentials, registry keys, passwords, decoy user accounts, process list, honey traffics, etc., are used in the factory VMs.

4.8.3 Deception Efficiency

A random selection of 916 malware samples from the dataset are used to assess Chimera's efficiency in deceiving attackers. The results are illustrated in Fig. 4.4. Chimera successfully deceived 879 malware samples (95.93%) that run to completion (e.g., exfiltration). Out of them, 781 malware samples took the baits and exfil-

Table 4.3: Number of techniques (T) and procedures (P) discovered by Chimera compared to Cuckoo sandbox and Any.run.

Family	**Malware Family**	**Discovery**	**Cuckoo**	**Any.run**	**Chimera**
InfoStealer	Fareit	T	8	7	8
		P	39	126	149
	LokiBot	T	7	2	11
		P	21	173	243
	Pony	T	8	4	17
		P	231	191	582
	Racoon	T	7	8	16
		P	45	23	51
Ransomware	Ryuk	T	6	3	6
		P	27	32	102
	GandCrab	T	8	10	10
		P	192	109	245
RAT	Gh0st	T	2	2	6
		P	4	57	69
	VanilaRat	T	1	0	12
		P	1	0	12
	Quasar	T	5	2	13
		P	14	4	16

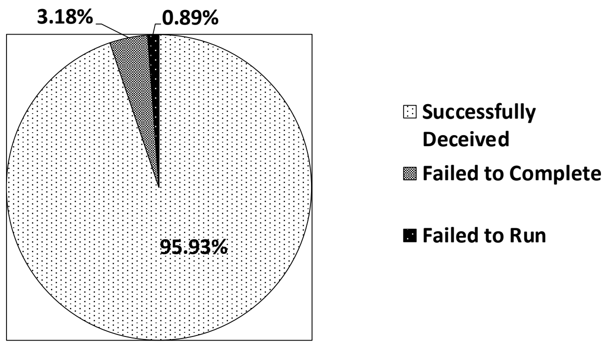


Figure 4.4: Chimera deception efficiency.

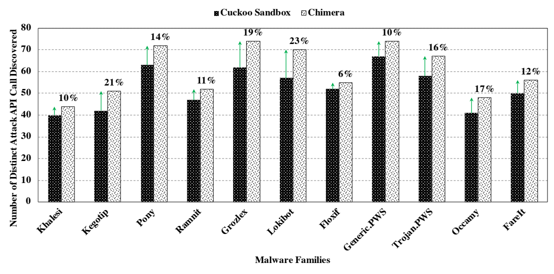


Figure 4.5: Unique attack API discovered.

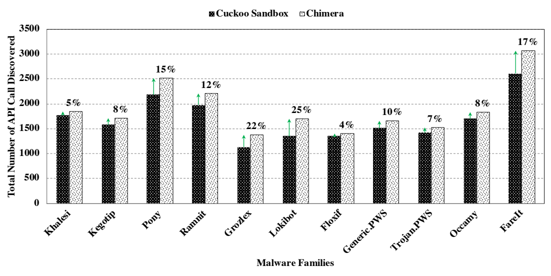


Figure 4.6: Total trace API coverage.

trated the honey resources, which observed by inspecting the plain Command and Control (C2) communication. However, 98 malware samples send encrypted traffic to C2, which got redirected to the decoy server because I cannot verify whether they took honey resources or not. I failed to run 8 malware samples (0.89%) because of either not having the C2 server, or the malware was unable to comply with system requirements. For instance, I could not run *VanillaStub.exe* (MD5: 185526401b0a3a083c797cac3598051a) RAT client for not having the master. The remaining malware samples did not run to completion due to expecting specific C2 commands/responses encrypted with specific keys.

4.8.4 Quality of Deception

I quantify deception quality by discovering new TTPs, unique API calls, distinct API call traces, and malware depletion to delay the attack propagation. I maintain the key criteria of successful deception: running malware to completion to reach its goal. The outcomes are compared with existing popular APT analyzers such as Cuckoo sandbox and Any.run.

TTP Discovery: Table 4.3 shows Chimera's performance in discovering technique (T) and procedure (P). Clearly, Chimera discovers more attack techniques and procedures than Cuckoo and Any.run. For instance, when I run the malware sample with MD5: 5ce9945d6999c9636c1f49e270382d6b in Chimera, I observed it search for files "C: \Users \admin\AppData \Roaming\Mozilla\Firefox\Prof-iles\qldyz51w.default \pkcs11.txt" by calling following APIs: *CreateFile, ReadFile, CloseHandle.* This procedure resembles the File and Directory Discovery technique. Later, I discover another technique "Application Layer Protocol" due to calling following APIs: *InternetOpen, InternetConnect, HttpOpenRequest, HttpSendRequest.* Cuckoo or Any.run cannot detect the later technique and corresponding procedures due to not having a C2 server setup. With decoy C2 server, Chimera discovers more techniques and procedures. For the same reason, Chimera outperforms the other tools in discovering

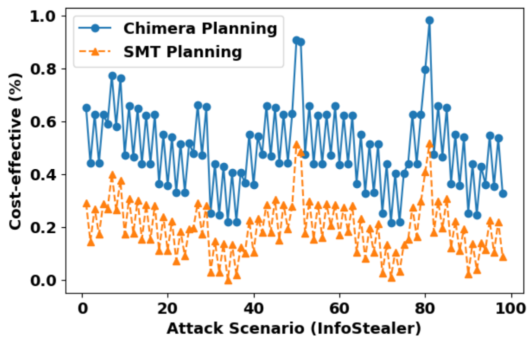


Figure 4.7: Optimal policy over cost effectiveness.

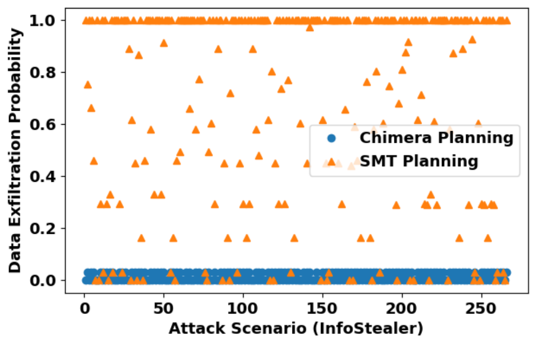


Figure 4.8: Optimal policy over data exfiltration probability.

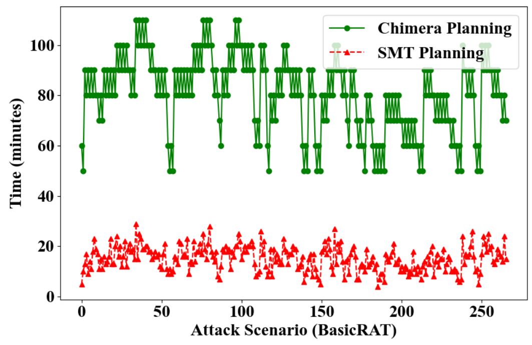


Figure 4.9: Depletion time against BasicRAT.

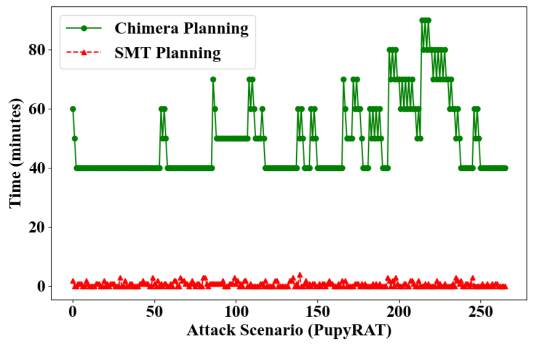


Figure 4.10: Depletion time against PupyRAT.

RATs.

Unique API Call and Trace Discovery: Fig. 4.5 shows the number of unique APIs discovered by Chimera compared to Cuckoo. On average, Chimera discovered 14.45% more unique API calls than Cuckoo. In total, Chimera found 78 distinct API calls that are not even monitored by Cuckoo at all. Fig. 4.6 shows distinct API trace coverage comparison. Chimera is ahead of 12.09% average trace coverage than Cuckoo. A trace is considered as distinct by taking the longest common subsequence from both Chimera and Cuckoo.

4.8.5 Optimal Policy

From the TTP discovery results, it is observed that the deception policy generated by Chimera is efficient enough to make notable results compared to the state-of-

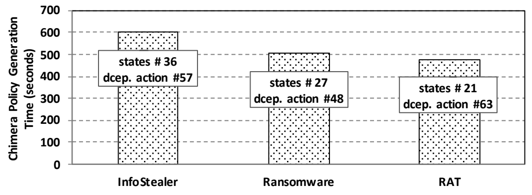


Figure 4.11: Policy generation overhead for different APT.

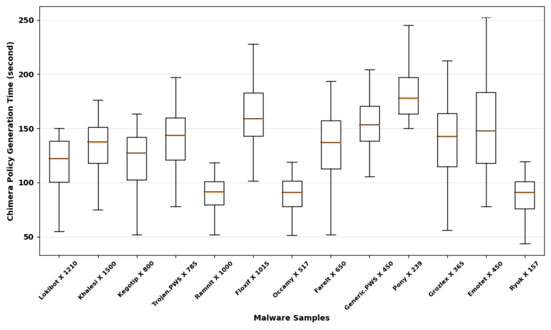


Figure 4.12: Policy generation overhead for different malware samples.

the-art tools. However, to measure the how optimal Chimera's policy is, a static planning is created using the same attack and defense action spaces over the deception graph. Satisfiability Modulo Theories (SMT) is used to create the static planning. Fig. 4.7 shows that, while running different attack scenarios of an Information Stealer, Chimera always best performs choosing the most cost-effective actions, meaning Chimera chooses low cost actions with high efficiency in deceiving the attacker. Similarly, Fig. 4.8 shows that, the SMT planning mostly fails to prevent data exfiltration, whereas in Chimera planning, data exfiltration probability is nearly zero.

The scenario (deplete the attacker to delay its progression) is depicted with two different malware, BasicRat that is called naive RAT, as it does not employ defense evasion techniques, and PupyRat, which can check the system to discover container environments (sandbox/VM). Fig. 4.9 shows that the SMT planning is capable of engaging with the attacker for 10 minutes on average before the BasicRat client quits because of not receiving the appropriate (deceptive) response. On the other hand, PupyRat detects the deception environment because of static planning in the minute mark (Fig. 4.10). In both cases, Chimera deception planning was able to engage with the attacker for more than an hour.

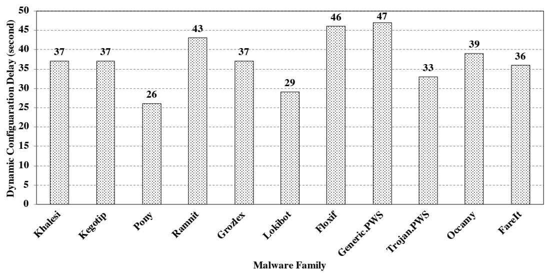


Figure 4.13: Deception delay.

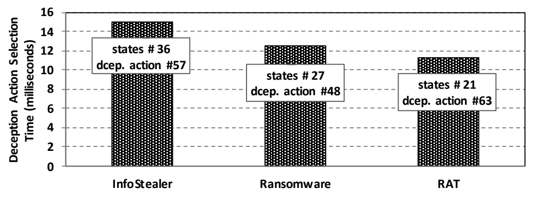


Figure 4.14: Optimal deception action selection time (online).

4.8.6 Policy generation overhead

The overhead of Chimera is measured in two aspects, 1) deception policy generation overhead and 2) delay due to deception action orchestration and deployment.

Offline Policy Generation Delay: From our datasets, I created three deception graphs for Information Stealer, Ransomware, and RAT, each having 36, 27, 21 states, respectively. Fig. 4.11 shows that the maximum policy generation overhead is 600 seconds for Information Stealer, having 37 states. The results also shows that the number of actions in deception graphs has little influence on the overhead. Fig. 4.12 depicts the deception policy generation delay of individually malware sub-family. The average policy creation delay is around 250 seconds.

Online Action Selection Delay: From the deception policy, Chimera deception agent selects an optimal deception action in run-time. Fig. 4.14 shows that the maximum delay to chose a deception action is 15 milliseconds for Information Stealer.

Dynamic Deception Delay: The dynamic deception delay is calculated by running a malware sample (to completion) into a machine without Chimera. Then Chimera is deployed into that machine and run the same malware. Time difference is calculated as system overhead due to deception action orchestration and deployment. Fig. 4.13 shows that the maximum orchestration delay is 47 seconds, which is insignificant compared to an APT campaign running time.

4.9 Summary

In this chapter, I introduced a framework named Chimera that provides an optimal deception plan to deceive APT attacks and achieve the 4D deception goals: diversion, depletion, distortion, and discovery in real-time. I use POMDP to obtain the optimal deception action planning. A deception environment is developed using Windows API hooking, where malicious API calls can be redirected to VM or responded with crafted honey content. Because of API level deception, Chimera can be used embedding with the production machine. Chimera is evaluated with 4,578 malware samples from three different families: Information Stealer, Ransomware, and RAT. Chimera deceives those malware samples with high efficiency (95.93%) and low system overhead (47s). The limitation in Chimera is that the deception is done through Windows System API hooking. If the malware can bypass API hooking or detect it, Chimera cannot deceive them.

CHAPTER 5: Conclusion

This dissertation focuses on the problems of dynamic orchestration and optimal planning of deception actions and moving target defense (MTD) techniques. Cyber deception and MTD enable active cyber defense (ACD) to resilient the system against advanced persistent threats (APT). ACD is a cyber resiliency capability that dynamically orchestrates security architectures to prevent attacks proactively.

To overcome these challenges, this dissertation focuses on developing an autonomous resilient ACD framework, having the following objectives: (1) evolving multistrategy ACD policies that leverage dynamic composition of various MTD and deception techniques, (2) a specification language to design various MTD and deception policies and an extensible rich API integrated with a synthesis engine for deploying security solutions without consulting the low-level network and system configuration management, (3) a theoretical framework and implementation for an autonomous goal-oriented cyber deception planner that optimizes deception decision-making. This section summarizes the overall objectives of the dissertation.

The first objective of this dissertation is to develop a proactive moving target defense against email spear-phishing attacks because the spear-phishing attack is the most utilized and effective adversary tactic that launches the APT campaign. In spear-phishing attacks, adversaries attach malicious links (URL) or files (malware) into the email and send them to targeted victims. Using spear-phishing emails, adversaries can manage to impersonate authoritative identities to incite victims to perform specific actions that help them achieve hacking or financial goals. The current state of the art for detecting spear-phishing emails limits by analyzing email contents which advanced attacker can easily evade by mimicking users' behavior and avoiding bad signatures. To address these limitations, a novel moving target technique called sender email address mutation is introduced in chapter 2 to protect against spear-phishing and spoofing attacks proactively.

The second objective of the dissertation is to develop a framework for automating the creation of deception actions and configuration-based MTD techniques rapidly and safely. Existing MTD and deception frameworks mostly provide static configuration. They often do not support multi-strategy defense composition as it can create conflicts with existing policies and break the system integrity. Moreover, without automatic, system misconfiguration is inevitable that leads to creating new vulnerabilities. In chapter 3, a framework is presented that provides a high-level cyber agility policy language specification and a controller for implementing configuration-based MTD and deception techniques. It also provides an extensible rich API that can be used to observe adversary actions, compose multi-strategy defense plans, and ensure safe yet quick deployment of such plans by automatically managing the network configuration.

The final objective of this dissertation is to provide optimal planning of deception course of actions and their automatic orchestration in the production environment in real-time. Such a framework can help the defender to design specific deception scenarios to deceive APT attacks in order to achieve specific deception goals, like diversion, distortion, depletion and discovery. In chapter 4, a framework is developed that generates an optimal deception action planning based on deception goals and APT groups. The framework leverage MITRE ATT&CK [11] to understand attack techniques and uses Sequential Decision-Making techniques such as Partially Observable Markov Decision Processes (POMDP) to formalize the problem.

Limitations & Future Tasks: Although this dissertation addresses many key challenges to solve the optimization and automation of deception and MTD techniques, it has certain limitations, which opens future research directions.

- In the sender email address mutation technique, the threat model does not consider the compromises of a user machine. In that case, the attacker can access the mutation gateway and launch a phishing email that cannot be detected by the given solution.

- The deception action optimization requires a mapping between high-level attack techniques to low-level system API calls, which has been done manually in this dissertation. In the future, such mapping can be automated using natural language processing tools and machine learning techniques.

- Measuring the effectiveness of deception action is challenging. Because, in most of the cases, the defender could not get a response from the command and control server of the attacker. This dissertation assumes deception actions effectiveness as user input which can be employed from historical data. In the future, techniques can be developed to measure the effectiveness of deception actions.

- The advanced attacker can counter deceive the defender's deception actions. It is challenging to determine whether an attacker takes a bait just because the deception is successful or the attacker is actually counter deceiving the defender. This dissertation does not focus on detecting counter deception.

REFERENCES

[1] "All data breaches in 2019 - 2021: An alarming timeline."

[2] "Fbi internet crime report 2020."

[3] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1137–1152, IEEE, 2019.

[4] "Identity fraud hits record high with 15.4 million u.s."

[5] "Ransomware trends in 2020."

[6] "Colonial pipeline paid hackers nearly $5 million in ransom."

[7] C. Xiao, A. Sarabi, Y. Liu, B. Li, M. Liu, and T. Dumitras, "From patching delays to infection symptoms: using risk profiles for an early discovery of vulnerabilities exploited in the wild," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 903–918, 2018.

[8] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitras, "The attack of the clones: A study of the impact of shared code on vulnerability patching," in *2015 IEEE symposium on security and privacy*, pp. 692–708, IEEE, 2015.

[9] E. Al-Shaer, J. Wei, W. Kevin, and C. Wang, *Autonomous Cyber Deception.* Springer, 2019.

[10] G. Ho, A. Cidon, L. Gavish, M. Schweighauser, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner, "Detecting and characterizing lateral phishing at scale," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1273–1290, 2019.

[11] *Adversarial Tactics, Techniques & Common Knowledge.* `https://attack.mitre.org/wiki/Main_Page`.

[12] P. Karuna, H. Purohit, S. Jajodia, R. Ganesan, and O. Uzuner, "Fake document generation for cyber deception by manipulating text comprehensibility," *IEEE Systems Journal*, 2020.

[13] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, “From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation,” in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 942–953, 2014.

[14] A. Juels and T. Ristenpart, “Honey encryption: Security beyond the brute-force bound,” in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 293–310, Springer, 2014.

[15] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, “Detecting credential spearphishing in enterprise settings,” in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 469–485, 2017.

[16] S. Kitterman, “Sender policy framework (spf),” RFC7208, 2014. `https://tools.ietf.org/html/rfc7208`.

[17] D. Crocker, T. Hansen, and M. Kucherawy, “Domainkeys identified mail (dkim) signatures,” RFC6376, 2011. `https://tools.ietf.org/html/rfc6376`.

[18] M. Kucherawy and E. Zwicky, “Domain-based message authentication, reporting, and conformance (dmarc),” RFC7489, 2015. `https://tools.ietf.org/html/rfc7489`.

[19] H. Hu and G. Wang, “End-to-end measurements of email spoofing attacks,” in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1095–1112, 2018.

[20] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, “Openpgp message format,” tech. rep., RFC 2440, November, 1998.

[21] B. Ramsdell *et al.*, “S/mime version 3 message specification,” tech. rep., RFC 2633, June, 1999.

[22] “Multi-factor authentication.” `https://en.wikipedia.org/wiki/Multi-factor_authentication`, 2020.

[23] S. Ruoti, J. Andersen, S. Heidbrink, M. O’Neill, E. Vaziripour, J. Wu, D. Zappala, and K. Seamons, ““we’re on the same page" a usability study of secure email using pairs of novice users,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 4298–4308, 2016.

[24] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland, “Why johnny still can’t encrypt: evaluating the usability of email encryption software,” in *Symposium On Usable Privacy and Security*, pp. 3–4, ACM, 2006.

[25] I. Thomson, “Who’s using 2fa? sweet fa. less than 10% of gmail users enable two-factor authentication,” *The Register*, 2018.

[26] J. Müller, M. Brinkmann, D. Poddebniak, H. Böck, S. Schinzel, J. Somorovsky, and J. Schwenk, "“johnny, you are fired!”–spoofing openpgp and s/mime signatures in emails," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1011–1028, 2019.

[27] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, 2008.

[28] Q. Duan, E. Al-Shaer, M. Islam, and H. Jafarian, "Conceal: A strategy composition for resilient cyber deception-framework, metrics and deployment," in *2018 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2018.

[29] E. Al-Shaer and M. N. Alsaleh, "Configchecker: A tool for comprehensive security configuration analytics," in *Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on*, pp. 1–2, IEEE, 2011.

[30] "Pomdp background." `http://www.pomdp.org/tutorial/pomdp-background.html`.

[31] "Spear-phishing email reports.." `https://www.phishingbox.com/`, 2020.

[32] "Business email compromise: The $26 billion scam." `https://www.ic3.gov/media/2019/190910.aspx`, 2019.

[33] Verizon, "2018 data breach investigations report." `https://enterprise.verizon.com/resources/reports/DBIR_2018_Report_execsummary.pdf`, 2018.

[34] J. Hong, "The state of phishing attacks," *Communications of the ACM*, vol. 55, no. 1, pp. 74–81, 2012.

[35] B. Parmar, "Protecting against spear-phishing," *Computer Fraud & Security*, vol. 2012, no. 1, pp. 8–11, 2012.

[36] V. Ramanathan and H. Wechsler, "Phishing detection and impersonated entity discovery using conditional random field and latent dirichlet allocation," *Computers & Security*, vol. 34, pp. 123–139, 2013.

[37] S. Aggarwal, V. Kumar, and S. Sudarsan, "Identification and detection of phishing emails using natural language processing techniques," in *Proceedings of the 7th International Conference on Security of Information and Networks*, p. 217, ACM, 2014.

[38] H. Gascon, S. Ullrich, B. Stritter, and K. Rieck, "Reading between the lines: content-agnostic detection of spear-phishing emails," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 69–91, Springer, 2018.

[39] X. Hu, B. Li, Y. Zhang, C. Zhou, and H. Ma, "Detecting compromised email accounts from the perspective of graph topology," in *Proceedings of the 11th International Conference on Future Internet Technologies*, pp. 76–82, 2016.

[40] S. Duman, K. Kalkan, M. Egele, W. Robertson, and E. Kirda, "Emailprofiler: Spearphishing filtering with header and stylometric features of emails," in *IEEE 40th COMPSAC*, vol. 1, pp. 408–416, IEEE, 2016.

[41] G. Stringhini and O. Thonnard, "That ain't you: Blocking spearphishing through behavioral modelling," in *Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 78–97, Springer, 2015.

[42] M. Khonji, Y. Iraqi, and J. Andrew, "Mitigation of spear phishing attacks: A content-based authorship identification framework," in *2011 International Conference for ITST*, pp. 416–421, IEEE, 2011.

[43] M. M. Islam, E. Al-Shaer, and M. A. B. U. Rahim, "Email address mutation for proactive deterrence against lateral spear-phishing attacks," in *International Conference on Security and Privacy in Communication Systems*, pp. 1–22, Springer, 2020.

[44] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pp. 60–69, 2007.

[45] J. T. Goodman, P. S. Rehfuss, R. L. Rounthwaite, M. Mishra, G. J. Hulten, K. G. Richards, A. H. Averbuch, A. P. Penta, and R. C. Deyo, "Phishing detection, prevention, and notification," Dec. 15 2009. US Patent 7,634,810.

[46] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: a literature survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2091–2121, 2013.

[47] M. M. Islam, Q. Duan, and E. Al-Shaer, "Specification-driven moving target defense synthesis," in *Proceedings of the 6th ACM Workshop on Moving Target Defense*, pp. 13–24, 2019.

[48] M. Quadrant, "Magic quadrant for secure email gateways," 2014.

[49] "Email security gateways.." `https://www.expertinsights.com/insights/top-11-email-security-gateways/`, 2020.

[50] D. Crocker, "Rfc0822: Standard for the format of arpa internet text messages," 1982.

[51] J. Klensin *et al.*, "Simple mail transfer protocol," tech. rep., rfc 2821, April, 2001.

[52] Django, "Django." `https://www.djangoproject.com/`, 2020.

[53] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[54] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 80–85, 2018.

[55] J. H. Jafarian, A. Niakanlahiji, E. Al-Shaer, and Q. Duan, "Multi-dimensional host identity anonymization for defeating skilled attackers," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pp. 47–58, ACM, 2016.

[56] S. Jajodia, V. Subrahmanian, V. Swarup, and C. Wang, *Cyber deception.* Springer, 2016.

[57] S. Jajodia, N. Park, F. Pierazzi, A. Pugliese, E. Serra, G. I. Simari, and V. Subrahmanian, "A probabilistic logic of cyber deception," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2532–2544, 2017.

[58] K. Horák, Q. Zhu, and B. Bošanskỳ, "Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security," in *International Conference on Decision and Game Theory for Security*, pp. 273–294, Springer, 2017.

[59] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using sdn-based virtual topologies," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1098–1112, 2017.

[60] K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major, "Game theory for adaptive defensive cyber deception," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pp. 1–8, 2019.

[61] "Deception technology market research report- forecast 2022."

[62] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pp. 1–6, IEEE, 2014.

[63] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[64] L. de Moura and N. Bjørner, "Satisfiability modulo theories: An appetizer," in *SBMF '09, Brazilian Symposium on Formal Methods*, 2009.

[65] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *SecureComm*, vol. 106, pp. 310–327, Springer, 2012.

[66] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, (New York, NY, USA), pp. 69–78, ACM, 2014.

[67] Q. Duan, E. Al-Shaer, and J. H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *CNS'13*, 2013.

[68] 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*, MTD '16, (New York, NY, USA), pp. 37–46, ACM, 2016.

[69] E. Al-Shaer, "Mutable networks, National cyber leap year summit 2009 participants ideas report," tech. rep., Networking and Information Technology Research and Development (NTIRD), August 2009.

[70] E. Al-Shaer, "Toward network configuration randomization for moving target defense," in *Moving Target Defense* (S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, eds.), vol. 54 of *Advances in Information Security*, pp. 153–159, Springer New York, 2011.

[71] D. Kewley, R. Fink, J. Lowry, and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering," *DARPA Information Survivability Conference and Exposition*, vol. 1, p. 0176, 2001.

[72] J. T. Michalski, "Network security mechanisms utilising network address translation," *International Journal of Critical Infrastructures*, vol. 2, no. 1, pp. 10–49, 2006.

[73] J. Michalski, C. Price, E. Stanton, E. Lee, C. K. Seah, Y. H. TAN, and C. Pheng., "Final report for the network security mechanisms utilizing network address translation ldrd project. technical report sand2002-3613," tech. rep., Sandia National Laboratories, 2002.

[74] M. Atighetchi, P. Pal, F. Webber, and C. Jones, "Adaptive use of network-centric mechanisms in cyber-defense," in *ISORC '03: Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, (Washington, DC, USA), p. 183, IEEE Computer Society, 2003.

[75] R. Morehart, "Evaluating the effectiveness of ip hopping via an address routing gateway," Master's thesis, AIR FORCE INSTITUTE OF TECHNOLOGY, 2013.

[76] M. Sifalakis, S. Schmid, and D. Hutchison, "Network address hopping: a mechanism to enhance data protection for packet communications," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 3, pp. 1518–1523 Vol. 3, May 2005.

[77] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pp. 127–132, ACM, 2012.

[78] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "A framework for reliable routing in mobile ad hoc networks," in *IEEE INFOCOM*, pp. 270–280, 2003.

[79] L. M. Marvel, S. Brown, I. Neamtiu, R. Harang, D. Harman, and B. Henz, "A framework to evaluate cyber agility," in *MILCOM 2015-2015 IEEE Military Communications Conference*, pp. 31–36, IEEE, 2015.

[80] D. Bodeau and R. Graubart, "Cyber resiliency engineering framework," *MTR110237, MITRECorporation*, 2011.

[81] J. D. Mireles, E. Ficke, J.-H. Cho, P. Hurley, and S. Xu, "Metrics towards measuring cyber agility," *IEEE Transactions on Information Forensics and Security*, 2019.

[82] A. Dutta and E. Al-Shaer, "“what”,“where”, and “why” cybersecurity controls to enforce for optimal risk mitigation," in *2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 160–168, IEEE, 2019.

[83] A. Dutta and E. Al-Shaer, "Cyber defense matrix: a new model for optimal composition of cybersecurity controls to construct resilient risk mitigation," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pp. 1–2, 2019.

[84] A. Schlenker, O. Thakoor, H. Xu, F. Fang, M. Tambe, L. Tran-Thanh, P. Vayanos, and Y. Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 892–900, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[85] T. E. Carroll and D. Grosu, "A game theoretic investigation of deception in network security," *Security and Communication Networks*, vol. 4, no. 10, pp. 1162–1172, 2011.

[86] E. Al-Shaer, J. Wei, K. W. Hamlen, and C. Wang, "Dynamic bayesian games for adversarial and defensive cyber deception," in *Autonomous Cyber Deception*, pp. 75–97, Springer, 2019.

[87] E. Miehling, M. Rasouli, and D. Teneketzis, "A pomdp approach to the dynamic defense of large-scale cyber networks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2490–2505, 2018.

[88] M. Al Amin, S. Shetty, L. Njilla, D. Tosh, and C. Kamouha, "Attacker capability based dynamic deception model for large-scale networks," *EAI Endorsed Transactions on Security and Safety*, vol. 6, no. 21, 2019.

[89] K. Chung, C. A. Kamhoua, K. A. Kwiat, Z. T. Kalbarczyk, and R. K. Iyer, "Game theory with learning for cyber security monitoring," in *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pp. 1–8, IEEE, 2016.

[90] O. Hayatle, H. Otrok, and A. Youssef, "A markov decision process model for high interaction honeypots," *Information Security Journal: A Global Perspective*, vol. 22, no. 4, pp. 159–170, 2013.

[91] J. H. Jafarian and A. Niakanlahiji, "A deception planning framework for cyber defense," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[92] C.-Y. J. Chiang, Y. M. Gottlieb, S. J. Sugrim, R. Chadha, C. Serban, A. Poylisher, L. M. Marvel, and J. Santos, "Acyds: An adaptive cyber deception system," in *MILCOM 2016-2016 IEEE Military Communications Conference*, pp. 800–805, IEEE, 2016.

[93] 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*, pp. 69–78, 2014.

[94] M. M. Islam, E. Al-Shaer, A. Dutta, and M. N. Alsaleh, "Clips/activesdn for automated and safe cybersecurity course-of-actions orchestration," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pp. 1–3, 2019.

[95] "Mininet: An instant virtual network on your laptop (or other pc)." `http://mininet.org/`.

[96] "Vagrant: Development environments made easy." `https://www.vagrantup.com/`.

[97] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *2013 IEEE symposium on security and privacy*, pp. 127–141, IEEE, 2013.

[98] "Namp."

[99] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[100] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, p. 102288, 2021.

[101] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[102] J. Rrushi, "Honeypot evader: Activity-guided propagation versus counter-evasion via decoy os activity," in *Proceedings of the 14th IEEE International Conference on Malicious and Unwanted Software*, 2019.

[103] L. Alt, R. Beverly, and A. Dainotti, "Uncovering network tarpits with degreaser," in *Proceedings of the 30th Annual Computer Security Applications Conference*, pp. 156–165, 2014.

[104] K. Borders, L. Falk, and A. Prakash, "Openfire: Using deception to reduce network attacks," in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*, pp. 224–233, IEEE, 2007.

[105] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman, "Examining the efficacy of decoy-based and psychological cyber deception," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[106] M. S. I. Sajid, J. Wei, M. R. Alam, E. Aghaei, and E. Al-Shaer, "Dodgetron: Towards autonomous cyber deception using dynamic hybrid analysis of malware," in *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2020.

[107] "Mitre att&ck." `https://attack.mitre.org/`.

[108] J. Lee, J. Choi, G. Lee, S.-W. Shim, and T. Kim, "Phantomfs: file-based deception technology for thwarting malicious users," *IEEE Access*, vol. 8, pp. 32203–32214, 2020.

[109] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 145–160, 2013.

[110] M. M. Islam and E. Al-Shaer, "Active deception framework: an extensible development environment for adaptive cyber deception," in *2020 IEEE Secure Development (SecDev)*, pp. 41–48, IEEE, 2020.

[111] A. Dutta, E. Al-Shaer, and S. Chatterjee, "Constraints satisfiability driven reinforcement learning for autonomous cyber defense," *arXiv preprint arXiv:2104.08994*, 2021.

[112] "Malware behavior catalog." `https://github.com/MBCProject`.

[113] "Cuckoo sandbox." `https://cuckoosandbox.org/`.

[114] "Any.run." `https://any.run/`.

[115] "Hybrid analysis." `https://www.hybrid-analysis.com/`.

[116] M. M. Islam, A. Dutta, M. S. I. Sajid, E. Al-Shaer, J. Wei, and S. Farhang, "Chimera: Autonomous planning and orchestration for malware deception," in *2021 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2021.

[117] "Chimera results." `https://anonymous.4open.science/r/Chimera-D656/README.md`.

[118] D. Braziunas, "Pomdp solution methods," *University of Toronto*, 2003.

[119] T. Smith, "Zmdp software for pomdp and mdp planning," 2013.

[120] M. S. I. Sajid, J. Wei, B. Abdeen, E. Al-Shaer, M. M. Islam, W. Diong, and L. Khan, "Soda: A system for cyber deception orchestration and automation," in *Annual Computer Security Applications Conference*, 2021.

[121] R. Al-Shaer, J. M. Spring, and E. Christou, "Learning the associations of mitre att & ck adversarial techniques," in *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, 2020.